Apprentissage de la programmation informatique : analyses et ressources pour accompagner la transition collège-lycée

Matthieu Branthome

COLLEGE DOCTORAL BRETAGNE
EDUCATION, LANGAGES
INTERACTIONS, COGNITION
CLINIQUE, EXPERTISE

Université de Bretagne Occidentale

Région Bretagne

THÈSE DE DOCTORAT DE

L'UNIVERSITE
DE BRETAGNE OCCIDENTALE

ÉCOLE DOCTORALE N° 646
Éducation, Langages , Interactions, Cognition, Clinique, Expertise
Spécialités : Didactique de l'informatique

Par

Matthieu BRANTHÔME

Apprentissage de la programmation informatique : analyses et ressources pour accompagner la transition collège-lycée

TOME 1 : TEXTE

Thèse présentée et soutenue à Rennes, le 11 octobre 2023
Unité de recherche : Centre de Recherche sur l'Éducation, les Apprentissages et la Didactique

Rapporteurs avant soutenance :

Sébastien GEORGE	Professeur des Universités, Le Mans Université
Vanda LUENGO	Professeure des Universités, Sorbonne Université

Composition du Jury :

Présidente :	Patricia MARZIN-JANVIER	Professeure des Universités, Université de Bretagne Occidentale
Examinateurs :	Gilles DOWEK	Directeur de recherche INRIA, professeur attaché ENS Paris-Saclay
	Sébastien GEORGE	Professeur des Universités, Le Mans Université
	Jean-Marie GILLIOT	Maître de conférences, IMT Atlantique
	Vanda LUENGO	Professeure des Universités, Sorbonne Université
	Agathe MERCERON	Professeure des Universités, Berliner Hochschule für Technik
Dir. de thèse :	Ghislaine GUEUDET	Professeure des Universités, Université Paris-Saclay
Co-dir. de thèse :	Cédric FLUCKIGER	Professeur des Universités, Université de Lille

Résumé

Cette thèse porte sur l'enseignement-apprentissage de la programmation informatique à la transition collège-lycée. L'objectif est dans un premier temps d'analyser finement le contexte global de cette transition afin de mettre au jour les discontinuités que peuvent rencontrer les élèves. Dans un second temps, le but est de concevoir et d'évaluer une ressource d'enseignement à même d'accompagner les élèves dans ces différents changements.

Ainsi, lors d'une première phase, nous réalisons une analyse épistémologique de l'informatique en général, puis plus précisément de la programmation informatique. Nous effectuons ensuite une étude approfondie des programmes d'enseignements et des documents d'accompagnement qui s'appuie sur la Théorie anthropologique du didactique. Nous complétons ces travaux par un examen détaillé des différences entre les langages de programmation Scratch et Python qui sont employés respectivement en classe de troisième et de seconde. Nous menons ensuite une enquête en ligne auprès de 480 enseignants visant à connaitre leurs pratiques déclarées d'enseignement de la programmation informatique.

En nous basant sur les résultats issus de ces analyses préalables, nous concevons l'application en ligne Pyrates prenant la forme d'un jeu sérieux d'apprentissage du langage Python. L'environnement de l'application est aménagé afin de faciliter le passage de la programmation par blocs à la programmation textuelle, et les huit niveaux du jeu mettent en jeu les différents concepts fondamentaux de la programmation informatique à travers des situations ludiques conçues dans une perspective constructiviste selon la Théorie des situations didactiques. Nous évaluons ensuite cette application lors d'une première expérimentation dans les classes auprès de 240 élèves de seconde. L'étude des traces d'activités générées nous permet de valider globalement cette conception puis de proposer des pistes d'améliorations.

Enfin, nous mettons en œuvre l'une de ces pistes visant à améliorer l'autonomie des élèves à travers la création d'un système de rétroactions épistémiques automatiques sélectionnées par des modèles d'intelligence artificielle. Ces modèles, qui prédisent la rétroaction la plus adaptée en fonction de l'activité des élèves dans l'application, sont entrainés à partir de données étiquetées par des enseignants lors d'une deuxième expérimentation dans des classes de seconde impliquant 215 élèves.

Mots-clés : enseignement-apprentissage de la programmation, transition blocs-texte, jeux sérieux, rétroactions automatiques, analytique des apprentissages, apprentissage automatique.

Abstract

This thesis focuses on computer programming education at the transition between lower and upper secondary school (from grade 9 to grade 10) in France. The aim is firstly to analyse in detail the global context of this transition in order to enlighten the discontinuities that students may encounter. Secondly, the goal is to design and evaluate teaching materials that can support students through these various changes.

In a first step, we carried out an epistemological analysis of computer science in general, and computer programming in particular. Then we proceeded with an in-depth study of curricula, based on the anthropological theory of didactics. This work is complemented by a detailed investigation of the differences between the Scratch and Python programming languages used in ninth and tenth grades respectively. Subsequently, we conducted an online survey of 480 teachers to find out about their declared practices in teaching computer programming.

Based on the results of these preliminary analyses, we designed the Pyrates online application as a serious game for learning the Python language. The application's environment was tailored to facilitate the transition from block-based programming to text-based programming, and the eight levels of the game are involving different fundamental concepts of computer programming through playful situations conceived in a constructivist philosophy following the theory of didactical situations. We then evaluated this application in classroom experiments including 240 grade 10 students. By analysing the generated activity traces, we were able to validate the overall design and suggest avenues for improvement.

Finally, we implemented one of these avenues in order support students' autonomy through the creation of a system of epistemic feedbacks triggered by students, and selected by artificial intelligence models. These models, which predict the most relevant feedback based on student activity in the application, were trained on data labelled by teachers during a second field experimentation involving 215 grade 10 students.

Keywords: computer programming education, block-to-text transition, serious games, automatic feedback, learning analytics, machine learning.

Publications liées à la thèse

Nous donnons ci-dessous les références de quatre articles qui ont été publiés en lien avec cette thèse. Les classements[1] des revues et conférences mentionnés ci-dessous sont ceux établis par l'Association des Technologies de l'Information pour l'Éducation et la Formation (ATIEF).

Branthôme, M. (2021). Apprentissage de la programmation informatique à la transition collège-lycée. *STICEF - Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, 28(3), 1-35. https://doi.org/10.23709/sticef.28.3.1 (Revue nationale rang A)

Branthôme, M. (2022). Conception et évaluation du jeu sérieux Pyrates : Agencement du milieu didactique pour la transition Scratch-Python. *Didapro 9 - 9ème colloque francophone de didactique de l'informatique*. 86-99. https://hal.science/hal-03674705 (Conférence nationale rang A)

Branthôme, M. (2022). Conception et évaluation du jeu sérieux Pyrates : Aspects didactiques dans la construction des niveaux. *RJC-EIAH - Neuvième rencontres jeunes chercheur en EIAH*. 36-42. https://hal.science/hal-03666645 (Conférence nationale rang B). Prix Martial Vivet 2022.

Branthôme, M. (2022). Pyrates : A Serious Game Designed to Support the Transition from Block-Based to Text-Based Programming. *EC-TEL - European Conference on Technology Enhanced Learning*, 31-44. https://doi.org/10.1007/978-3-031-16290-9_3 (Conférence internationale rang A+)

[1] http://www.atief.fr/ressources/classement

Remerciements

En premier lieu, je tiens à remercier très chaleureusement Ghislaine Gueudet et Cédric Fluckiger qui ont encadré cette thèse. Merci à Ghislaine, qui dirige mes travaux depuis le master, pour sa grande disponibilité, ses relectures minutieuses et ses conseils avisés. Merci à Cédric, qui a accepté de codiriger ma thèse, pour ses relectures, ses remarques et ses conseils toujours pertinents. J'ai grandement apprécié la qualité de nos échanges, la confiance qu'ils m'ont accordé, et leur accompagnement à la fois exigent et bienveillant.

Je remercie ensuite très sincèrement Sébastien George et Vanda Luengo, les rapporteurs de cette thèse, pour leur lecture attentive de mon manuscrit et pour la qualité et la pertinence des remarques qu'ils ont formulées dans leur rapport. Je suis également reconnaissant envers Gilles Dowek, Jean-Marie Gilliot, Patricia Marzin-Janvier et Agathe Merceron qui me font l'honneur de participer au jury.

Je remercie Patricia Marzin-Janvier et Sandra Nogry pour leurs conseils et leurs éclairages dans le cadre du comité de suivi.

Un grand merci aux différents chercheurs de la communauté EIAH qui me sont venu en aide tout au long de cette thèse, je pense notamment à : Marielle Léonard, Bertrand Marne, Mathieu Muratet, Amel Yessad, Sébastien Jolivet, Fahima Djelil et Sébastien Lallé.

J'exprime également ma reconnaissance envers les enseignants et les élèves qui ont participés aux différentes expérimentations sur le terrain, pour le temps et la confiance qu'ils m'ont accordé.

Je remercie mes amis de la « Fox » pour les bons moments que constituent nos différentes retrouvailles annuelles, véritables sas de décompression loin des tracas du quotidien. Je suis particulièrement reconnaissant envers Carole, à la genèse de cette aventure, pour son soutien, son aide et ses conseils éclairés.

Je remercie enfin toute ma famille, plus particulièrement mes parents qui par leur exemple m'ont transmis le gout du travail, la curiosité et l'ouverture d'esprit, mes beaux-parents, toujours présents pour me libérer du temps en accueillant leurs petits-enfants, en rendant service, et en me mijotant de bons petits plats.

Pour terminer, j'adresse un immense merci à ma compagne Dorothée, pour son soutien logistique et moral sans faille et pour ses encouragements. Ces cinq années ne furent pas toujours un long fleuve tranquille, mais elle fait partie de ceux sur qui on peut compter de manière inconditionnelle. Enfin, je remercie Clément et Éloïse, mes enfants, d'avoir accepté mes longues heures d'absence enfermé dans mon bureau ou dans la chambre avec mon ordinateur, d'avoir enduré les repas passé à « raconter ma thèse », et d'avoir servi de cobayes programmeurs en tant que « marins d'eau douce ». En espérant servir à mon tour de modèle d'opiniâtreté dans l'accomplissement de ses objectifs : hissez haut !

Table des matières

Introduction

Au cours des dernières décennies, les technologies numériques ont pris une place prépondérante dans nos vies quotidiennes, notamment à travers l'usage des réseaux sociaux (Abiteboul & Cattan, 2022), et les progrès de l'intelligence artificielle (Abiteboul & Dowek, 2017). Cette tendance, qui semble s'être accélérée depuis la pandémie de COVID-19 (Elimelech et al., 2022; Limone & Toto, 2021), a placé les compétences numériques (van Laar et al., 2017) au premier plan des politiques publiques en matière d'éducation. Il s'agit donc désormais d'instruire des citoyens éclairés capables de recul et d'esprit critique envers ces technologies, mais également de former une main-d'œuvre qualifiée à même de répondre aux besoins de la société.

Dans ce contexte sociétal transformé, la plupart des systèmes éducatifs au niveau mondial ont introduit les compétences numériques dans les curriculums de l'école primaire jusqu'au lycée (European Commission, 2022; Passey, 2017; Storte et al., 2019). La science informatique est un ensemble de concepts et de techniques sous-jacentes à ces compétences numériques, et la programmation informatique y joue un rôle central (Dowek, 2011).

En France, depuis 2016, la programmation informatique fait l'objet d'une découverte à l'école primaire (Vandevelde & Fluckiger, 2020), puis d'une consolidation en mathématiques au collège (Branthôme, 2021). Au lycée, après une adoption faite de hauts et de bas depuis les années 1970 (Baron et al., 2014), la programmation informatique est approfondie par l'intermédiaire d'un enseignement en mathématiques depuis 2009 (Modeste, 2012), et dans des disciplines informatiques dédiées depuis 2019 (Branthôme, 2021).

Cet enseignement en plusieurs étapes à différents moments de la scolarité engendre inévitablement des obstacles pour les élèves lors des transitions entre les différents paliers. C'est en particulier le cas lors de la liaison entre le collège et le lycée au cours de laquelle les élèves connaissent des changements concernant notamment les notions enseignées, les langages de programmation utilisés, et les types d'activités proposés (Branthôme, 2021).

L'objectif de cette thèse est double, il s'agit d'abord d'approfondir l'analyse de l'enseignement de la programmation informatique à la fin du collège et au début du lycée, puis de concevoir une ressource d'enseignement à même d'accompagner les élèves lors du passage de l'un à l'autre.

Dans cette introduction, nous exposons d'abord les questions de recherche que nous allons investiguer, puis nous décrivons la méthodologie générale employée, enfin nous détaillons le plan de ce manuscrit.

Questions de recherche

Dans le but d'accompagner les élèves dans la transition du collège au lycée, il s'agit d'abord d'avoir une compréhension fine des curriculums, en particulier pour les niveaux charnières qui nous intéressent : la troisième et la seconde. Au préalable, cela nécessite une analyse épistémologique portant sur la nature même de l'informatique et en particulier de la programmation informatique. Les langages de programmation prescrits dans les textes officiels en France, Scratch au collège puis Python au lycée,

sont différents à bien des égards. Il est donc également nécessaire de les étudier en détails dans le but d'analyser ces divergences. Ensuite, au-delà des prescriptions curriculaires, il est utile d'interroger les enseignants afin de connaitre leurs pratiques effectives d'enseignement de la programmation informatique dans les classes. Ces analyses épistémologiques et praxéologiques constituent le socle d'une seconde phase plus interventionniste qui consiste à concevoir une ressource d'enseignement de la programmation informatique à la transition collège-lycée. Cette ressource prenant la forme d'une application en ligne basée sur un jeu sérieux doit ensuite être testée sur le terrain afin d'évaluer sa conception. Enfin, cette application peut faire l'objet d'évolution en fonction de ces résultats. Nous parvenons ainsi aux six questions de recherche qui structurent cette thèse que nous présentons ci-dessous.

- Q1 : Qu'est-ce que l'informatique en général, et plus particulièrement la science informatique et la programmation informatique ? Quels sont les concepts fondamentaux de la programmation informatique ?
- Q2 : Quelle est la place de la programmation informatique dans les programmes scolaires français à la transition collège-lycée ? Quelles sont les modalités d'enseignement prescrites ? Quelles sont les conséquences pour les élèves qui passent du collège au lycée ?
- Q3 : Quelles sont les différences entre les langages de programmation Scratch et Python ? Quelles sont les conséquences pour les apprenants qui transitent de l'un à l'autre ?
- Q4 : Quelles sont les pratiques déclarées des enseignants concernant l'enseignement de la programmation informatique en troisième et en seconde ? Quelles répercussions pour la création d'une ressource d'enseignement ?
- Q5 : Comment concevoir une ressource d'enseignement de la programmation informatique sous la forme d'une application en ligne basée sur un jeu sérieux ? De quelle manière les élèves s'emparent-ils de cette application sur le terrain ?
- Q6 : Comment améliorer l'autonomie des élèves vis-à-vis des enseignants par le biais de rétroactions automatisées lors de l'utilisation de notre application ?

Dans les chapitres qui suivent, nous déclinerons chacune de ces six questions principales en sous-questions.

Méthodologie générale de la thèse

Afin de répondre à ces questions de recherche, la méthodologie générale de la thèse est celle de l'ingénierie didactique. Cette méthodologie issue de la didactique des mathématiques est en relation étroite avec la Théorie des situations didactiques (Brousseau, 1998) que nous présenterons en détail dans le Chapitre 5. L'ingénierie didactique a pour objectif d' « ouvrir le champ des possibles de l'enseignement en construisant une niche expérimentale où un objet non observable directement puisse vivre et où certaines caractéristiques de son fonctionnement puissent être analysées » (Artigue, 1988, p. 69). Un aspect fondamental est son mode de validation basé sur la confrontation entre une analyse à priori dans laquelle un certain nombre d'hypothèses sont formulées, et une analyse à posteriori qui s'appuie sur les données issues de

l'observation (Artigue, 2002). Nous détaillons ci-dessous les différentes phases de l'ingénierie didactique (Artigue, 1988).

- (1) Analyses préalables : il s'agit d'une étape préliminaire visant à établir le champ de contraintes dans lequel va se situer la réalisation didactique. Cela comprend l'analyse épistémologique des contenus visés, l'analyse de l'enseignement usuel et de ses effets, des conceptions des élèves, et des difficultés et obstacles qu'ils peuvent rencontrer.
- (2) Conception d'une réalisation didactique : lors de cette phase, le chercheur prend la décision d'agir sur un certain nombre de variables du système concernant l'organisation générale de l'expérimentation et la conception fine des activités d'enseignement proposées aux élèves. L'objectif de cette étape est de « déterminer en quoi les choix effectués permettent de contrôler les comportements des élèves et leur sens » (Artigue, 1988, p. 258).
- (3) Expérimentation dans les classes : c'est la phase de la mise en œuvre effective de la réalisation didactique sur le terrain et de recueil de données empiriques.
- (4) Analyse à posteriori et évaluation : lors de cette dernière étape de mise en regard des faits avec les prédictions a lieu la validation ou l'invalidation des hypothèses engagées dans la recherche. Comme l'affirme Dorier : « L'analyse a posteriori est alors une lecture des observables à la lumière de la grille fournie par l'analyse a priori » (2010, p. 10). Il est ensuite possible de réitérer le processus conception-expérimentation-analyse, à l'aune des résultats obtenus.

Au-delà de cette méthodologie qui régit la démarche générale de la thèse, nous présenterons, localement aux différents chapitres, la méthodologie spécifique employée et les concepts théoriques mobilisés pour répondre à certaines questions de recherche. Dans le même esprit, nous ne proposons pas de revue de littérature générale mais présenterons l'état de l'art relatif à chaque question dans les chapitres qui les traitent.

Plan du manuscrit

Ce manuscrit est composé de six chapitres répondant chacun à une question de recherche, ils s'articulent comme suit.

Les quatre premiers chapitres concernent la phase d'analyse préalable de l'ingénierie didactique. Ainsi, le Chapitre 1 propose une étude épistémologique portant sur l'informatique puis plus précisément sur la science informatique et la programmation informatique, et enfin sur les concepts fondamentaux mis en jeu dans cette activité. Ces analyses s'appuient sur la littérature scientifique et des ouvrages universitaires constituant le « savoir savant » (Chevallard, 1985) pour la discipline informatique.

Le Chapitre 2 présente une analyse curriculaire des textes officiels français concernant l'enseignement de la programmation informatique en classe de troisième et de seconde. Cette étude s'appuie sur les programmes d'enseignement et les documents d'accompagnement émanant du ministère de l'Éducation nationale et mobilise la Théorie anthropologique du didactique (Chevallard, 1992) et ses prolongements.

Le Chapitre 3 est consacré à l'étude des langages de programmation Scratch et Python, et en particulier à l'analyse des différences intrinsèques aux langages et de celles liées à leur environnement d'édition. Ces analyses sont fondées sur des exemples d'utilisation et des travaux préexistants dans la littérature.

Le Chapitre 4 expose les résultats d'une enquête en ligne réalisée auprès de 480 enseignants de troisième et de seconde visant à connaitre leurs pratiques d'enseignement de la programmation informatique.

La suite de la thèse se rapporte aux phases de conception, d'expérimentation et d'analyse de l'ingénierie didactique. De cette manière, le Chapitre 5 évoque la conception de la ressource d'enseignement créée qui prend la forme d'une application en ligne comportant un jeu sérieux d'apprentissage du langage Python. Cette conception s'appuie en partie sur la Théorie des situations didactiques (Brousseau, 1998). Ce chapitre présente ensuite l'évaluation de cette application en situation écologique dans des classes auprès de 240 élèves de seconde.

Le Chapitre 6 décrit une évolution de l'application visant à améliorer l'autonomie des élèves à travers la création d'un système de rétroactions automatisées. Le déclenchement des rétroactions est à l'initiative des élèves, cependant la sélection de la rétroaction à afficher est assurée par des modèles d'intelligence artificielle. Ces modèles ont été entrainés à partir de données étiquetées par des enseignants lors d'une deuxième expérimentation dans les classes impliquant 215 élèves.

Enfin, le Chapitre 7 conclut cette thèse. Il en résume les principaux résultats, expose les contributions pour la recherche et les praticiens, discute les limitations de la méthodologie, et termine par esquisser les perspectives pour de futures recherches.

Chapitre 1

Programmation informatique : analyse épistémologique

Dans ce chapitre, nous menons une analyse épistémologique relative aux concepts visés par l'enseignement de la programmation informatique. Cette analyse des contenus poursuit trois objectifs que nous détaillons ci-dessous.

Il s'agit, premièrement, en suivant le précepte du poète Nicolas Boileau : « ce qui se conçoit bien s'énonce clairement », de définir précisément les termes employés dans ce manuscrit afin de lever toute ambiguïté à leur égard. En effet, comme l'affirme Fluckiger (2019), le terme « informatique » ne possède pas de définition consensuelle :

> Ces questions sémantiques sont à mettre en relation avec la nature même de l'informatique. Comme le rappelle Drot-Delange, « la réponse n'est pas triviale » (Drot-Delange, 2016, p. 39). Les auteurs s'accordent habituellement sur la difficulté d'une définition unifiée, conduisant à des définitions « composites ». (Fluckiger, 2019, p. 18)

Deuxièmement, comme l'explique Artigue (1989), l'analyse épistémologique doit permettre d'aider le chercheur « à mettre à distance et sous contrôle les "représentations épistémologiques" [...] induites par l'enseignement. En aidant à redonner une historicité aux concepts [...] que l'enseignement usuel tend à présenter comme des objets universels, à la fois dans le temps et dans l'espace » (1989, p. 1). C'est avec cet objectif de décontextualisation temporelle et spatiale que nous nous appuyons sur la littérature internationale et que nous présentons, dans la suite, quelques éléments historiques en lien avec les langages de programmation.

Troisièmement, nous nous inscrivons dans une démarche classique en didactique (Alvarez, 2007; Candy, 2020; Crisci, 2020; Jolivet, 2018; Ravel, 2003) consistant à étudier la *transposition didactique* (Chevallard, 1985) d'un ensemble de savoirs. Comme le précise Artigue (1989) :

> L'analyse épistémologique permet également au didacticien de prendre la mesure des disparités existant entre savoir "savant" [...] et savoir "enseigné". En effet, alors que l'École vit sur la fiction consistant à voir dans les objets d'enseignement des copies simplifiées mais fidèles des objets de la science, l'analyse épistémologique, en nous permettant de comprendre ce qui gouverne l'évolution de la connaissance scientifique, nous aide à prendre conscience de la distance qui sépare les économies des deux systèmes. (1989, p. 2)

Il s'agit donc pour nous de constituer un modèle épistémologique du *savoir savant* lié à la programmation informatique. C'est-à-dire de délimiter le corpus des connaissances, reconnues comme pertinentes et valides au sein de la communauté scientifique des chercheurs en informatique. Ceci dans l'objectif d'étudier, dans le Chapitre 2, la transposition qui en est faite dans le *savoir à enseigner*, autrement dit, dans les textes officiels définissant les contenus à enseigner en France à la transition collège-lycée.

Dans ce chapitre, nous commençons par préciser nos questions de recherche et notre méthodologie (section 1.1). Nous procédons ensuite, suivant le modèle de l'entonnoir, en nous intéressant à l'informatique de façon globale que nous caractérisons en quatre niveaux (section 1.2), puis à un niveau en particulier, la science informatique dont nous listons les domaines de connaissance (section 1.3). Par la suite, nous affinons encore le grain d'étude en nous focalisant sur le domaine de connaissance des langages de programmation (section 1.4), et enfin sur les concepts fondamentaux de la programmation informatique (section 1.5).

1.1 Questions de recherche et méthodologie

Dans ce chapitre nous étudions la question de recherche Q1 : Qu'est-ce que l'informatique en général, et plus particulièrement la science informatique et la programmation informatique ? Quels sont les concepts fondamentaux de la programmation informatique ?

Nous déclinons celle-ci selon les sous-questions suivantes :

- Q1.1 : Dans un cadre sociétal général, qu'entend-on par « informatique » au sens large ? Quels sont les différents niveaux qui la composent ?
- Q1.2 : Quel est le périmètre de la science informatique ? Quels sont les domaines de connaissance qui la constituent ?
- Q1.3 : Quelle définition peut-on donner d'un langage de programmation afin d'en capturer les aspects fondamentaux ? Quels sont les différents niveaux d'abstraction des langages de programmation par rapport à la machine ? Quels sont les principaux paradigmes de programmation ? Quelle filiation historique pour les langages contemporains ?
- Q1.4 : Quel est le corpus des concepts fondamentaux mis en jeu dans l'activité de programmation avec un langage impératif ?

Ces sous-questions sont traitées dans les différentes sections qui composent ce chapitre. Notre méthodologie s'appuie sur l'étude de la littérature internationale. Nous citons à la fois la littérature scientifique de la recherche en science informatique mais également des ouvrages généralistes, rédigés par des universitaires et destinés aux étudiants ou aux enseignants, que nous considérons comme reflétant le savoir savant. Nous nous basons également sur des documents prescriptifs nationaux et internationaux à destination des concepteurs de programmes (Computer Science Curricula 2013) ou régissant des concours d'enseignement (programme de l'agrégation d'informatique 2023).

1.2 L'informatique : théorie, techniques, outils et usages.

Il s'agit dans cette section d'établir ce que recouvre le terme courant d'« informatique ». En France, plusieurs auteurs, issus de différents champs scientifiques, se sont attelés à cette tâche avec la contrainte de proposer une caractérisation robuste face aux évolutions rapides du domaine. Les auteurs que nous citons ci-après produisent leurs analyses à des époques et dans des contextes technologiques distincts. Ils semblent cependant converger vers une partition de

l'informatique en quatre niveaux : une théorie, des techniques, des outils et des usages. Chaque auteur y plaçant les frontières à sa manière et articulant singulièrement ces niveaux entre eux.

1.2.1 Différents points de vue

Ainsi, en 1990, le sociologue Philippe Breton publie un ouvrage retraçant l'histoire de l'informatique. Dans une tentative de définition de son objet d'étude, il commence par en énumérer les différents aspects :

> L'informatique a de multiples dimensions : [...] la biographie des inventeurs, leurs sources de créativité, les facteurs économiques et la structure de l'industrie, le poids des innovations techniques, les croyances philosophiques des créateurs, les notions scientifiques de base qui sont disponibles, le contexte social et politique, les contraintes techniques d'utilisation des matériels et des langages, les traditions dont l'informatique hérite et l'idéologie moderniste qui la soutient activement. (Breton, 1990, p. 10)

Son idée est de pouvoir considérer ces nombreux aspects au cours des trois périodes historiques qu'il discerne[2]. Aussi, pour chacune de ces époques, il positionne des « thèmes essentiels » dans un triangle dont les trois sommets représentent :

- les « techniques de base » dans lesquelles il regroupe les techniques matérielles et logicielles, mentionnons par exemple : « lampes à vide », « transistor », « norme ASCII » et « programme enregistré » ;
- les « usages et économie de l'ordinateur » rassemblant les différentes classes d'utilisations des outils informatique (militaires, industrielles et domestiques) dont voici quelques illustrations : « marchés d'État militaires », « compagnies américaines », « filière électronique » et « percée des utilisateurs » ;
- les « fondements de la culture informatique », on y retrouve les théories et paradigmes sous-jacents dans le domaine des sciences et des idées, citons pêle-mêle : « théorie de l'information », « universalité de la logique », « société de communication » et « informatique = liberté » (Breton, 1990, p. 10-13).

Ce « triangle de Breton », permettant de saisir l'essence de l'informatique à chaque période historique, discerne les techniques informatiques des usages des outils informatiques tout en prenant en compte l'aspect culturel, capable de fournir un éclairage sur les soubassements théoriques, conceptuels et idéologiques qui y prennent corps.

La même année, le philosophe Michel Mirabail (1990) constate que le statut des savoirs informatiques a évolué : « d'abord et principalement reliés à la programmation des ordinateurs et aux méthodes algorithmiques de résolution de problèmes [...], ils acquièrent une signification plus large, à la fois technologique et sociale, car liés à l'évolution des matériels et des applications au monde du travail » (1990, p. 11-12). Il poursuit en affirmant que « l'informatique possède donc une triple caractéristique, qui en fait une science et une technologie en voie de constitution, en même temps qu'un agent social et culturel de changement des habitudes et des comportements » (1990, p. 12). Nous retiendrons ici la caractérisation de l'informatique en trois parties : une science liée à la programmation des machines et à la résolution algorithmique de

[2] Première informatique : 1945-1965, deuxième informatique : 1965-1980 et troisième informatique : 1980-1990.

problèmes, une technologie en évolution s'appuyant sur les avancées matérielles, et un facteur de changements sociétaux via l'usage des outils qui en découlent.

Plus tard, alors que les micro-ordinateurs commencent à envahir les foyers et qu'Internet n'est encore que balbutiant, l'informaticien Bernard Lang (1998) avance que :

> L'informatique a de multiples facettes, science théorique et expérimentale objet de recherches d'une grande diversité, technologie donnant lieu à une activité industrielle considérable, et ensemble d'outils de plus en plus intégrés à notre vie quotidienne, familiale ou professionnelle [...]. Beaucoup en sont encore à confondre les aspects fondamentaux et pérennes avec leur expression actuelle dans des outils destinés à évoluer rapidement [...], les disciplines plus anciennes distinguent sans problème ces trois composantes, et nul ne confond la thermodynamique, la technologie des moteurs à explosion et le mode d'emploi d'un véhicule automobile [souligné par l'auteur]. (1998, p. 2)

Il propose donc de définir l'informatique à l'aune de ses usages, en établissant des catégories que nous pouvons résumer dans le triptyque : une science fondamentale et pérenne objet de recherches, une technologie évolutive mobilisée par l'industrie, et des outils employés dans les usages domestiques et sociaux.

Ces dernières années, de nombreux universitaires s'accordent sur le rôle majeur qu'a acquis l'informatique. Lazard et Mounier-Kuhn, dans l'introduction de leur « Histoire illustrée de l'informatique » (2019) dressent deux constats : « d'une part, nous baignons dans une civilisation transformée par l'informatique et nous utilisons tous des appareils numériques dans notre vie quotidienne [...]. D'autre part, ces technologies sont devenues des enjeux économiques et sociaux gigantesques » (2019, p. 13). Abiteboul et Dowek, dans l'ouvrage « Le temps des algorithmes » (2017) traitant des questions sociales et éthiques soulevées par l'utilisation massive des algorithmes, listent les nombreux usages informatiques et analysent les transformations qui en découlent à l'échelle de la société :

> Les algorithmes sont devenus des ingrédients essentiels de nos vies professionnelles, de nos interactions sociales, de notre médecine, de notre industrie, de nos transports, de notre commerce, etc. Ils transforment les sciences exactes et les sciences humaines et contribuent ainsi à enrichir nos connaissances. (2017, p. 6)

C'est dans ce contexte d'expansion des usages que Bruillard (2016) propose un découpage de l'informatique en se plaçant du point de vue de l'activité qu'elle rend possible :

> Prenant la question à partir de la place des concepteurs/utilisateurs de l'informatique et de leurs activités, on peut caractériser l'informatique autour de trois approches complémentaires [...] : (1) algorithmique et traitements automatisés autour du cycle données / traitement / résultats ; (2) interaction continue avec des machines, des artefacts sémiotiques [...]; (3) participation à des interactions sociales avec des agents humains et non humains via les réseaux. La première, celle de l'informatique comme science de calcul, consacre une forme de démarche intellectuelle. La seconde correspond à l'utilisation personnelle des dispositifs informatisés, la troisième à l'informatique sociale. (2016, p. 31)

Bruillard identifie donc trois classes d'usages de l'informatique : les utilisations scientifiques (calculs et traitements de données), les usages personnels par le biais des machines, et les pratiques sociales à travers les réseaux.

Quelques années plus tard, Fluckiger (2019) adopte le point de vue d'une informatique plurielle :

> Il existe donc une certaine convergence vers l'idée d'une pluralité ontologique de l'informatique, et même, plus précisément, d'une triple nature de l'informatique. Il y a certes des divergences, selon que soit considéré comme ligne de fracture principale par exemple la distinction entre logiciel et matériel, ou entre science et application technique de la science, mais il y a bien accord autour de l'idée que l'informatique comprend une dimension sociale. Selon cette approche, relève de l'informatique à la fois ce qui relève de la science et des manières de penser, de la technologie, mais aussi, et c'est là le point crucial, des usages spécifiques. Cela signifie que les usages de l'informatique ne sont pas de simples « utilisations » de produits de la science informatique, ils font partie intégrante de l'informatique elle-même. (Fluckiger, 2019, p. 19)

Il avance donc l'idée d'une « triple nature » de l'informatique qui comprend non seulement la science et la technologie mais également un ensemble d'usages spécifiques.

On assiste, en même temps que les usages de l'informatique se diversifient, à un glissement sémantique : « Aujourd'hui où le terme numérique supplante le mot informatique, l'ordinateur lui-même semble disparaitre sous des couches de plus en plus épaisses de logiciels et de fonctions de communications, photographiques et ludiques » (Lazard & Mounier-Kuhn, 2019, p. 13). Berry (2017), dans un livre regroupant ses cours dispensés au Collège de France, fait le même constat : « on a vu se produire récemment deux glissements de vocabulaire dans les médias, les discours politiques et l'enseignement : "informatique" est devenu "numérique" et "programmation" est devenu "codage" » (2017, p. 6-7). Il propose donc, en s'appuyant sur ce changement de vocabulaire, d'opérer une dichotomie :

> Le mot « informatique » désignera spécifiquement la science et la technique du traitement de l'information, et, par extension, l'industrie directement dédiée à ces sujets. L'adjectif « numérique » peut être accolé à toute activité fondée sur la numérisation et le traitement de l'information : photographie numérique, son numérique, édition numérique, sciences numériques, art numérique, etc. [...]. Le raccourci « le numérique » rassemblant toutes les activités auxquelles on peut accoler l'adjectif « numérique ». Puisque toute information numérisée ne peut être traitée que grâce à l'informatique, celle-ci est le moteur conceptuel et technique du monde numérique. (2017, p. 7-8)

Berry met donc en avant une distinction entre une science informatique mise en œuvre dans des techniques de traitement de l'information (l'informatique) et les usages sociaux permis par ces techniques de l'autre (le numérique).

Fluckiger (2019) abonde dans ce sens en affirmant que le terme numérique désigne davantage la dimension d'usage des outils informatiques plutôt que les dimensions scientifiques et techniques, tout en élargissant sa base technologique au traitement de l'information au-delà du seul ordinateur :

> le terme [numérique] semble permettre un double dépassement : d'une part ne pas parler que d'ordinateur mais d'un ensemble plus vaste de technologies, d'autre part ne pas se focaliser sur les seules dimensions scientifiques et techniques mais aussi sur les usages. *Numérique* renvoie donc à la fois à la volonté de désigner l'ensemble des technologies s'appuyant sur un tel traitement numérique de l'information et non les seuls ordinateurs et à la nécessité de distinguer les dimensions d'usage des outils devenus quotidiens, bien éloignés de la science informatique qui leur a, parfois de manière lointaine, donné naissance. Ainsi, le téléphone, Facebook, le développement du commerce en ligne ou les MOOC relèvent, dans le langage et les représentations, du *numérique*, beaucoup moins nettement de l'*informatique* [mis en valeur par l'auteur]. (Fluckiger, 2019, p. 25)

1.2.2 Découpage retenu

Nous venons d'exposer les considérations de plusieurs auteurs quant à la définition du terme « informatique ». Ils s'accordent tous sur son caractère hybride, sur l'existence de soubassements théoriques, et sur la diversité de ses utilisations. Cependant, ils utilisent un terminologie variée pour désigner chaque aspects : science, technique, outil, technologie, usage, numérique, etc.

Nous clarifions la situation en définissant précisément le vocabulaire que nous allons utiliser dans cette thèse. D'abord, en suivant Fluckiger (2019), nous abandonnons l'appellation « numérique » dans la mesure où il s'agit d'un terme générique :

> Le terme *numérique* ainsi substantivé n'apporte pas de clarification : il peut désigner aussi bien un contexte général induisant de nouvelles pratiques des élèves que l'installation de TNI dans les classes [...]. Il n'y a donc pas de réel gain heuristique à reprendre le terme [mis en valeur par l'auteur]. (Fluckiger, 2019, p. 25)

Ensuite, nous proposons un découpage de l'informatique en quatre niveaux hiérarchiques : une théorie, des techniques, des outils, et des usages. Ce découpage est illustré dans le Figure 1-1, chaque niveau constitue le socle du suivant.

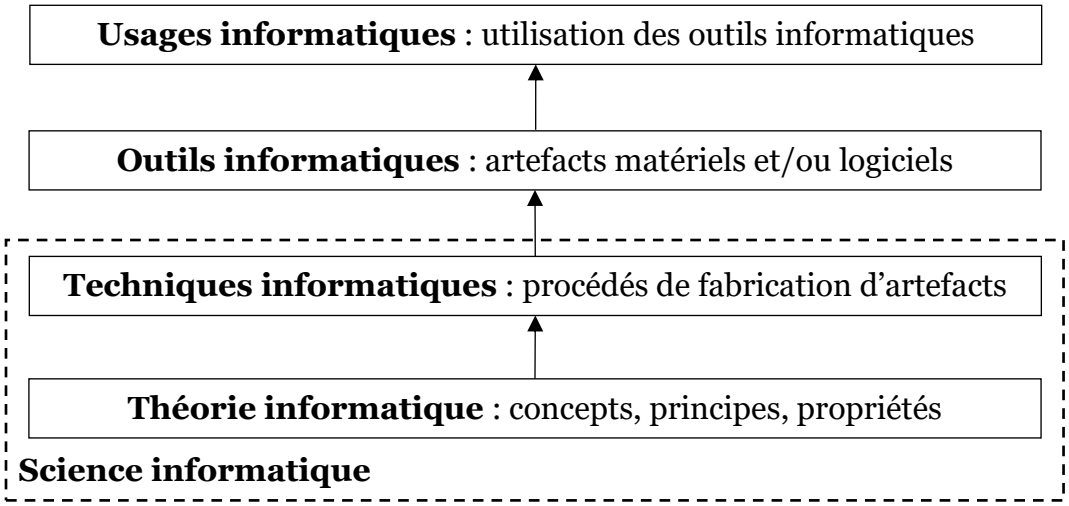


Figure 1-1 – Les différents niveaux hiérarchiques constituant l'informatique

Premièrement, la *théorie informatique* est un ensemble de concepts, de principes et de propriétés issus de la recherche théorique et expérimentale qui constituent les bases pérennes de l'informatique. Deuxièmement, nous considérons les *techniques*

informatiques, qui évoluent continuellement. Nous les définissons, en nous appuyant sur l'étymologie (du grec « technè » : savoir-faire), comme un ensemble de procédés méthodiques fondés sur la théorie informatique et employés dans la production d'artefacts matériels ou logiciels. Troisièmement, les *outils informatiques* sont des moyens mis à disposition de la société. Ce sont des artefacts alliant matériels et logiciels utilisés à de multiples fins par des groupes sociaux différents. Quatrièmement, les *usages informatiques* sont l'ensemble des utilisations des outils informatiques dans la société. La sociologie des usages (Jouët, 2000) décrit ces différentes sphères sociales (personnelle, familiale, amicale, professionnelle, etc.) et ces différentes finalités (divertissement, interaction sociale, productivité, activité artistique, etc.). Nous avons, par ailleurs, cité dans les lignes précédentes de nombreuses incidences sociétales engendrées par les usages informatiques dans différents domaines et à différentes époques.

Enfin, Dowek et ses collègues (2011) avancent que la distinction classique science-technique ne fait pas sens s'agissant de l'informatique :

> L'apparition de l'informatique nous mène aussi à revoir les relations qui existent entre les sciences et les techniques. Nous étions accoutumés à voir les techniques comme des applications des sciences, ce que traduit notre habitude de les appeler « sciences appliquées » […]. Il est probable que l'apparition de l'informatique nous contraigne à abandonner cette hiérarchie. Ignorer que l'informatique étudie des objets – machines, programmes, etc. – mais en fabrique également donne une image complètement déformée de la discipline. Il est certes possible de distinguer une activité scientifique, qui vise à connaître, d'une activité technique, qui vise à fabriquer, mais il n'est plus possible pour le savant d'ignorer l'activité de l'ingénieur. (2011, p. 22)

Pour ces auteurs, la science informatique est constituée à la fois d'un corpus théorique de connaissance et mais également d'un ensemble de techniques visant la fabrication d'artefacts. En cohérence avec eux, nous nommons *science informatique* la conjonction de la théorie et des techniques informatiques.

Notons que dans le monde anglo-saxon, nous retrouvons le même type de considération. La Royal Society (2012, p. 5) précise ces termes, ainsi « computer science » est défini comme « referring to the scientific discipline of Computer Science, covering principles such as algorithms, data structures, programming, systems architecture, design, problem solving etc. ». Ce terme semble recouvrir, de la même manière que « science informatique » en français, les aspects théoriques et techniques de la discipline. L'expression « information technology (IT) » qui désigne « the use of computers, in industry, commerce, the arts and elsewhere, including aspects of IT systems architecture, human factors, project management, etc. » regroupe les outils informatiques et leurs usages[3]. Enfin, la dénomination « computing », qui englobe « the broad subject area », est le pendant du mot français « informatique ».

[3] Remarquons que dans le contexte de l'informatique, le terme « technologie » est utilisé en français, non pas dans son sens originel d'étude des techniques, mais plutôt comme ce que le Robert considère comme un anglicisme et que le Larousse définit comme l'« ensemble des outils et des matériels utilisés dans l'artisanat et dans l'industrie ».

1.2.3 Synthèse de la section

Q1.1 : Dans un cadre sociétal général, qu'entend-on par « informatique » au sens large ? Quels sont les différents niveaux qui la composent ?

En nous appuyant sur différentes auteurs issus de divers champs scientifiques, nous avons défini l'**informatique** à travers un découpage en quatre niveaux hiérarchiques :

- la **théorie informatique**, ensemble de concepts, principes et propriétés stables issus de la recherche scientifique ;
- les **techniques informatiques**, ensemble évolutif de procédés méthodiques fondés sur la théorie informatique et utilisés dans la production d'artefacts matériels et/ou logiciels ;
- les **outils informatiques**, ensemble d'artefacts utilisés à diverses fins par différents groupes sociaux ;
- les **usages informatiques**, utilisation des outils informatiques menant à des incidences sociétales dans de nombreux domaines.

Nous définissons la **science informatique** comme la conjonction de la théorie et des techniques informatiques.

Après avoir dégagé les différents niveaux de l'informatique, penchons-nous plus précisément, dans la section suivante, sur la science informatique.

1.3 La science informatique : domaines de connaissances

Comme nous venons de l'exposer dans la section précédente, nous considérons la science informatique comme la combinaison d'un noyau théorique relativement stable et d'un ensemble évolutif de techniques pouvant mener à la conception d'outils informatiques. Le but de cette section est d'établir le périmètre de cette science.

Afin d'y parvenir, nous prenons comme repère les programmes d'enseignement de fin de premier cycle universitaire (« undergraduate » dans le contexte international). En effet, à ce niveau de diplôme, les étudiants ont acquis un socle scientifique généraliste en informatique leur permettant de se spécialiser lors de leur poursuite d'étude en deuxième cycle. Nous nous appuyons pour cela sur deux documents de référence : l'un international et l'autre français. Il s'agit d'abord du « Computer Science Curricula 2013 » (CSC13) établi par un groupe de travail international regroupant des universitaires et des professionnels de l'informatique (Task force ACM & IEEE-CS, 2013). Ces auteurs sont issus des deux principales associations internationales dédiées à l'informatique : l'Association for Computing Machinery (ACM) et l'Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS). Sous-titré « Curriculum Guidelines for Undergraduate Degree Programs in Computer Science » le rapport fournit des directives pour l'élaboration des programmes de premier cycle de l'enseignement supérieur au niveau mondial. Ensuite, pour la France, ne disposant pas de programmes unifiés pour la licence

d'informatique, nous nous basons sur le Programme de l'Agrégation d'Informatique 2023 (PAI23) dont la première session a eu lieu en 2022 (MEN, 2022a). Comme le précise le rapport du jury (MEN, 2022b), ce programme a été établi par des inspecteurs généraux de l'Éducation nationale et des universitaires.

1.3.1 Computer Science Curricula

Commençons par examiner le contenu du CSC13. Nous détaillons dans le Tableau 1-1 les différents domaines de connaissance identifiés.

N°	Domaine de connaissance	Détails
D1.1	Algorithms and Complexity	Defines the central concepts and skills required to design, implement, and analyze algorithms for solving problems including: strategies, computability and complexity.
D1.2	Architecture and Organization	Develop a deeper understanding of the hardware environment upon which all computing is based, and the interface it provides to higher software layers.
D1.3	Computational Science	The application of computer science to solve problems across a range of discipline, including modeling and simulation.
D1.4	Discrete Structures	Foundational concepts for many other areas in computing, including: set theory, logic, proof techniques, graph theory, and probability theory.
D1.5	Graphics and Visualization	Computer generation and manipulation of images, including: rendering, animation and visualization.
D1.6	Human-Computer Interaction	Designing interactions between human activities and the computational systems.
D1.7	Information Assurance and Security	Processes intended to protect information by ensuring their confidentiality, integrity, and availability, and by providing for authentication and non-repudiation.
D1.8	Information Management	Capture, digitization, representation, organization, transformation of information including: access and update, data modeling and abstraction, physical storage.
D1.9	Intelligent Systems	Set of general and specialized knowledge representation schemes, problem solving mechanisms and learning techniques for Artificial intelligence (AI).
D1.10	Networking and Communication	How the networks behave and the key principles behind the organization and operation of the networks.
D1.11	Operating Systems	Basic knowledge of operating systems (OS) including: interfacing to networks, kernel and user modes, approaches to OS design and implementation.
D1.12	Platform-Based Development	Development of applications on specific software platforms including: web programming, multimedia development, mobile and app development, and robotics.
D1.13	Parallel and Distributed Computing	Fundamental systems concepts such as concurrency and parallel execution, message-passing and shared-memory models, atomicity, consensus, and conditional waiting.
D1.14	Programming Languages	Programming Language (PL) models including: functional, object-oriented, logic, event-driven. PL composition, implementation, translation, static analysis.
D1.15	Software Development Fundamentals	Fundamental programming concepts and data structures, basic software development methods and tools.

D1.16	Software Engineering	Software lifecycle including: requirements elicitation, analysis and specification, design, construction, verification and validation, deployment; and maintenance.
D1.17	Systems Fundamentals	Fundamental systems concepts including: computational paradigms, parallelism, cross-layer communications, state and state transition, resource allocation and scheduling.
D1.18	Social Issues and Professional Practice	Larger societal context of computing to develop an understanding of the relevant social, ethical, legal and professional issues.

Tableau 1-1 – Nom et détails des domaines de connaissance distingués dans le CSC13

Dans ce document, le corps de connaissance (« body knowledge ») de la science informatique est divisé en 18 domaines de connaissance (« knowledge area »). Nous proposons ci-dessous une catégorisation de ces domaines en quatre ensembles.

Un premier ensemble de domaines, plutôt théoriques, est en rapport étroit avec les mathématiques. Nous le nommons *mathématiques pour l'informatique*, et nous le séparons en deux sous-ensembles :

- *calculabilité et complexité* : D1.1-Algorithms and Complexity ;
- *mathématiques discrètes* : D1.4-Discrete structure.

Un deuxième ensemble mélangeant éléments théoriques et techniques compose le *cœur de la science informatique*. Il est possible de le subdiviser en fonction du niveau d'abstraction qu'il propose en partant de la machine vers les applications de plus haut niveau :

- *architectures des machines* : D1.2-Architecture and Organization ;
- *systèmes d'exploitation* : D1.17-Systems Fundamentals et D1.11-Operating Systems ;
- *réseaux* : D1.10-Networking and Communication ;
- *programmation* : D1.14-Programming Languages, D1.15-Software Development Fundamentals, D1.13-Parallel and Distributed Computing et D1.16-Software Engineering ;
- *gestion des données* : D1.8-Information Management ;
- *intelligence artificielle* : D1.9-Intelligent Systems ;
- *sécurité* : D1.7-Information Assurance and Security ;
- *interactions homme-machine* : D1.6-Human-Computer Interaction.

Un troisième ensemble de domaines peut être qualifié d'*informatique appliquée*. En effet, l'informatique y est utilisée comme un outil permettant de produire divers artefacts (image, graphique, site Web, application mobile, etc.) : D1.3-Computational Science, D1.5-Graphics and Visualization et D1.12-Platform-Based Development.

Enfin, le dernier ensemble, que nous nommons *enjeux sociétaux* est en rapport avec les usages informatiques dans la société et les enjeux soulevés : D1.18-Social Issues and Professional Practice.

1.3.2 Programme de l'agrégation d'informatique

Dans un second temps, étudions le référentiel que nous avons choisi pour la France. Le Tableau 1-2 liste les domaines de connaissance présents dans le PAI23.

N°	Domaine de connaissance	Détails
D2.1	Architecture	Machine de von Neumann, appel de fonction et piles, hiérarchie mémoire, topologie des machines parallèles, représentation des nombres flottants.
D2.2	Base de données	Langage de requête SQL, algèbre relationnelle, théorème de Codd.
D2.3	Calculabilité et complexité	Modèle de calcul, machine de Turing, calculabilité, complexité, fonction récursives, Lambda-calcul, thèse de Church-Turing.
D2.4	Chaîne de compilation	Analyse lexicale, syntaxique et sémantique, génération de code.
D2.5	Fondements de la programmation	Preuve de programme, sémantique des langages.
D2.6	Génie logiciel	Cycle de vie, cycle de développement, agilité.
D2.7	Informatique et société	RGPD, DIG, impacts environnementaux, éthique et valeurs sous-jacentes.
D2.8	Intelligence artificielle	Apprentissage machine, données d'entrainement et de test, choix des descripteurs, biais et transparence.
D2.9	Logique	Logique du premier ordre.
D2.10	Programmation	Programmation objet et fonctionnelle.
D2.11	Programmation Web	Javascript : accès DOM, évènements, requêtes http, programmation asynchrone.
D2.12	Réseaux	Caractéristiques et performances, modélisation des couches, transmissions, programmation réseau.
D2.13	Systèmes d'exploitation	Séquence de démarrage, abstraction système, lien système-application, processus, concurrence, émulation et virtualisation.

Tableau 1-2 – Nom et détails des domaines de connaissance distingués dans le PAI23

De la même manière que pour le CSC13, nous sommes en mesure de classer les 13 domaines du PAI23 en différents ensembles.

D'abord, le premier ensemble théorique des mathématiques pour l'informatique et ses deux sous-ensembles :

- calculabilité et complexité : D2.3-Calculabilité et complexité ;
- mathématiques discrètes : D2.5-Fondements de la programmation, D2.9-Logique.

Ensuite, le deuxième ensemble qui constitue le cœur de la science informatique regroupant des domaines mêlant théorie et technique que nous pouvons sous-catégoriser :

- architectures des machines : D2.1-Architecture ;
- systèmes d'exploitation : D2.13-Systèmes d'exploitation;

– réseaux : D2.12-Réseaux ;
– programmation : D2.4-Chaîne de compilation, D2.10-Programmation, D2.6-Génie logiciel ;
– gestion des données : D2.2-Bases de données ;
– intelligence artificielle : D2.8-Intelligence artificielle.

Puis, l'ensemble de l'informatique appliquée qui ne comporte dans ce référentiel qu'un élément lié à programmation des site Internet : D2.11-Programmation Web.

Et pour terminer, l'ensemble des enjeux sociétaux en lien avec les usages informatique et ses conséquences : D2.7-Informatique et société.

1.3.3 Périmètre de la science informatique

Afin d'arrêter le périmètre de la science informatique, nous nous appuyons sur les quatre ensembles définis ci-dessus. Dans le but de maintenir la cohérence avec la définition de la science informatique établie dans la section précédente, nous ne conservons pas les ensembles de domaines en lien avec les outils ou les usages que sont l'informatique appliquée et les enjeux sociétaux. Par conséquent, dans le cadre de cette thèse, nous considérons que la science informatique est composée des ensembles mathématiques pour l'informatique et cœur de la science informatique. Pour résumer, nous définissons donc le périmètre de la science informatique comme l'agrégation des domaines de connaissance suivants : calculabilité et complexité, mathématiques discrètes, architectures des machines, systèmes d'exploitation, réseaux, programmation, gestion des données, intelligence artificielle, sécurité, interactions homme-machine.

1.3.4 Synthèse de la section

Q1.2 : Quel est le périmètre de la science informatique ? Quels sont les domaines de connaissance qui la constituent ?

Après avoir analysé des documents prescriptifs de référence, nous avons caractérisé la science informatique comme étant composée des dix domaines de connaissance suivants : **calculabilité et complexité**, **mathématiques discrètes**, **architectures des machines**, **systèmes d'exploitation**, **réseaux**, **programmation**, **gestion des données**, **intelligence artificielle**, **sécurité**, **interactions homme-machine**.

Ayant borné précisément le contenu de la science informatique, nous nous focalisons dans la section suivante sur les langages de programmation.

1.4 Les langages de programmation

Comme nous venons de le voir, la programmation informatique est un domaine de connaissance au cœur de la science informatique. Dowek et ses collègues (2011) avancent qu'il s'agit de sa clé de voûte :

> Un programme est un texte qui exprime un algorithme transformant de l'information et qui est écrit dans un langage de programmation afin d'être exécuté par une machine. Clé de voûte où les quatre arcs qui structurent l'informatique se rejoignent, la programmation a naturellement une place privilégiée [...]. (2011, p. 125)

Berry (2017) est du même avis lorsqu'il décline sa vision de la science informatique comme celle du traitement de l'information :

> L'informatique calcule sur l'information à l'aide d'algorithmes, de programmes et de machines [...]. L'information est codée dans des données numériques, l'algorithme est le mécanisme conceptuel de calcul systématique, le programme constitue l'écriture précise de l'algorithme dans les langages appropriés, et la machine est l'objet matériel capable de faire les calculs nécessaires pour transformer les programmes en actions (2017, p. 5)

Il parait sans équivoque, ici aussi, que la programmation informatique est la colonne vertébrale de la science informatique.

L'activité de programmation informatique s'effectue à travers l'utilisation de langages de programmation. Dans cette partie d'analyse épistémologique, nous nous intéressons donc plus à l'artefact qu'est le langage de programmation qu'à l'activité de programmation informatique. Nous allons donc dans les sous-sections suivantes en arrêter une définition, présenter les différents niveaux d'abstraction par rapport à la machine, lister les différents paradigmes de programmation, et en relater brièvement l'historique.

1.4.1 Définition d'un langage de programmation

Afin de caractériser précisément la notion de langage de programmation et dans le but d'en saisir les aspects fondamentaux, nous allons nous appuyer sur les définitions proposées par plusieurs auteurs dans des ouvrages universitaires généralistes en lien avec la programmation.

Différentes définitions

Selon Mitchell (2003), un langage de programmation est d'abord un moyen d'expression partagé par un groupe humain :

> Programming languages are the medium of expression in the art of computer programming. An ideal programming language will make it easy for programmers to write programs succinctly and clearly. Because programs are meant to be understood, modified, and maintained over their lifetime, a good programming language will help others read programs and understand how they work. (2003, p. 3)

Nous retenons qu'un langage de programmation est un *média*, au sens d'un procédé permettant la diffusion d'information, qui doit permettre aux programmeurs de se faire comprendre des autres dans la mesure ou les programmes informatiques peuvent avoir plusieurs auteurs différents durant leur cycle de vie.

Dowek (2008) ajoute qu'un langage de programmation, en plus d'être compréhensible par les hommes, doit être utilisable par des machines :

> Les textes écrits dans les langages de programmation présentent enfin la nouveauté d'être compréhensibles à la fois par les hommes, ce qui les rapproche des partitions utilisées par les organistes, et utilisables par des machines, ce qui s'approche des cartes perforées utilisées par les orgues de Barbarie. (2008, p. 4)

C'est, en effet, la raison d'être première d'un langage informatique que de permettre aux humains de contrôler des machines.

Les langages informatiques pilotent les machines par l'intermédiaire de *programmes* informatiques qui doivent avoir une certaine forme définie par la *syntaxe* du langage. Cela fait dire à Ghezzi et Jazayeri (2008) : « Any programming language specifies a set of rules for the form of valid programs in that language [...]. The syntax rules of the language state how to form expressions, statements, and programs that look right » (2008, p. 41). Cette syntaxe permet de décrire les constituants des programmes que sont les *instructions* :

> It is sufficient to know that the syntax of [a language] L allows us to use a given finite set of constructs, called instructions, to construct programs. A program in L (or program written in L) therefore is nothing more than a finite set of instruction of L. (Gabbrielli & Martini, 2010, p. 1)

Comme l'explique Sebesta (2012), au-delà de la syntaxe qui décrit les règles de formation des programmes, la *sémantique* d'un langage permet de lui donner une signification :

> The study of programming languages, like the study of natural languages, can be divided into examinations of syntax and semantics. The syntax of a programming language is the form of its expressions, statements, and program units. Its semantics is the meaning of those expressions, statements, and program units. (2012, p. 114)

Scott (2015) ajoute que la syntaxe et la sémantique des langages doivent être exprimées de façon très précise à l'aide d'une *notation formelle* :

> Unlike natural languages such as English or Chinese, computer languages must be precise. Both their form (syntax) and meaning (semantics) must be specified without ambiguity, so that both programmers and computers can tell what a program is supposed to do. To provide the needed degree of precision, language designers and implementors use formal syntactic and semantic notation. (2015, p. 41)

Cette notation formelle peut être une grammaire non-contextuelle (dite « Backus-Naur Form ») pour la syntaxe et une grammaire attribuée (« attribute grammar ») pour la sémantique.

Comme l'avance Knuth dans l'ouvrage de référence « The Art of Computer Programming » (1968), la sémantique des langages doit permettre de spécifier des algorithmes : « Formally defined programming languages or computer languages are designed for specifying algorithms, in which every statement has a very definite meaning » (1968, p. 5). Nous retenons comme définition d'un *algorithme*, celle élaborée par Modeste (2012) dans sa thèse : « Un algorithme est une procédure de résolution de problème, s'appliquant à une famille d'instances du problème et produisant, en un nombre fini d'étapes constructives, effectives, non-ambigües et organisées, la réponse au problème pour toute instance de cette famille. » (2012, p. 25).

Cependant, tous les langages de programmation ne sont pas capables de mettre en œuvre tous les algorithmes, c'est là qu'intervient la notion de *Turing-complétude* précisée par Perugini (2022) :

> A programming language is a system of data-manipulation rules for describing computation [...]. The notion of mechanical computation (or an algorithm) is formally defined by the abstract mathematical model of a computer called a Turing machine. A

> Turing machine is a universal computing model that establishes the notion of what is computable. A programming language is referred to as Turing-complete if it can describe any computational process that can be described by a Turing machine. […]. Support for sequential execution of both variable assignment and conditional-branching statements (e.g., if and while, and if and goto) is sufficient to describe computation that a Turing machine can perform. Thus, a programming language with those facilities is considered Turing-complete. (2022, p. 4-5)

La notion de Turing-complétude est un moyen d'établir la puissance d'un langage de programmation dans sa capacité à exprimer ou non tous les algorithmes (au sens d'une procédure systématique réalisable mécaniquement). Si un langage peut décrire tous les calculs qu'une machine de Turing peut effectuer, alors ce langage est Turing-complet, et il pourra exprimer tous les algorithmes. Cette notion s'appuie sur la thèse de Church-Turing (Church, 1936) qui affirme l'équivalence expressive de tous les modèles théoriques des algorithmes (machine de Turing, λ-calcul, fonctions récursives, etc.). Notons par exemple que les premières versions du langage de requête SQL, ou les langages de description tels que HTML ou celui des expressions régulières ne sont pas Turing-complets. Précisons également que certains problèmes (modélisables par des fonctions mathématiques) ne sont pas *calculables,* c'est-à-dire que l'on ne peut pas les décrire par un algorithme et donc pas les traiter mécaniquement. L'exemple fondateur est celui du problème d'arrêt (Turing, 1936).

Établissement d'une définition

Après avoir évoqué, en nous référant à plusieurs auteurs, les différents aspects fondamentaux des langages de programmation, nous sommes en mesure d'en proposer une définition robuste : un langage de programmation est un média qui permet de décrire des algorithmes calculables par des machines et compréhensibles par des humains par le biais de programmes constitués d'instructions suivant une certaine syntaxe et ayant une certaine sémantique définies de façon formelle.

Synthèse de la sous-section

Q1.3.1 : Quelle définition peut-on donner d'un langage de programmation afin d'en capturer les aspects fondamentaux ?

En nous appuyant sur des définitions proposées par divers auteurs dans des ouvrage généralistes portant sur les langages de programmation, nous avons établi une définition qui prend en compte les aspects fondamentaux des langages de programmation :

Un **langage de programmation** est un **média** qui permet de décrire des **algorithmes** calculables par des **machines** et **compréhensibles** par des **humains** par le biais de **programmes** constitués d'**instructions** suivant une certaine **syntaxe** et ayant une certaine **sémantique** définies de façon **formelle**.

Après avoir caractérisé de manière rigoureuse les langages de programmation, nous allons maintenant évoquer les différents niveaux d'abstraction de la machine qu'ils permettent.

1.4.2 Les différents niveaux d'abstraction des langages de programmation

Les langages de programmation ont été créés afin de permettre aux programmeurs de s'abstraire progressivement des contraintes imposées par les machines. Ceci avec deux principaux objectifs : pouvoir mettre en œuvre des algorithmes indépendamment des différentes architectures matérielles existantes, et avoir la possibilité de s'exprimer dans un langage se rapprochant des langages naturels[4]. Nous décrivons dans la suite ces différents niveaux d'abstraction.

Langages machine

Les *langages machine* sont les langages natifs des ordinateurs et la première génération de langages de programmation utilisés par les humains pour les contrôler[5]. Scott (2015) precise : « machine language is the sequence of bits that directly controls a processor, causing it to add, compare, move data from one place to another, and so forth at appropriate times » (2015, p. 5). Ces langages machine sont formés d'instructions en binaire qui permettent de contrôler directement les unités de calcul. Comme l'explique O'Regan (2021), ils ont des avantages et des inconvénients :

> The main advantage of these languages is execution speed as they may be directly executed on the computer [...]. However, writing a program in machine code is difficult and error prone, as it involves writing a stream of binary numbers. This made the programming language difficult to learn and difficult to correct should any errors occur. (2021, p. 177)

Les langages machine présentent donc la commodité d'être exécutés très rapidement, mais en contrepartie leur utilisation rend l'écriture et la correction des programmes très difficile. Pour illustrer nos propos, nous reproduisons dans la Figure 1-2 un programme en langage machine qui calcule le Plus Grand Diviseur Commun (PGCD) de deux entiers en implémentant l'algorithme d'Euclide. Ce programme est écrit en langage machine x86 et exprimé en nombres hexadécimaux (base 16).

```
55 89 e5 53  83 ec 04 83  e4 f0 e8 31  00 00 00 89  c3 e8 2a 00
00 00 39 c3  74 10 8d b6  00 00 00 00  39 c3 7e 13  29 c3 39 c3
75 f6 89 1c  24 e8 6e 00  00 00 8b 5d  fc c9 c3 29  d8 eb eb 90
```

Figure 1-2 – Programme PGCD en langage machine x86 (Scott, 2015, p. 5)

À la lecture, il est particulièrement fastidieux de comprendre ce que fait ce programme.

Langages d'assemblage

L'étape suivante dans le développement des langages de programmation est l'utilisation des *langages d'assemblage* (ou assembleurs). Scott (2015) décrit leur fonctionnement :

[4] En science informatique, le langage « naturel » désigne le langage utilisé par les humains pour communiquer entre eux. C'est la définition qu'en donne Kumar (2011) : « Natural languages are the languages which have naturally evolved and used by human beings for communication purposes, for example, Hindi, English, French, German are natural languages » (Kumar, 2011, p. 1).

[5] Même s'ils pourraient, d'un certain point de vue, correspondre à la définition que nous donnons d'un langage de programmation, nous ne considérons pas les premiers mécanismes de contrôle des machines que sont les cartes perforées (machine de Jacquard, de Babbage, ordinateur MARK1, etc.) comme des langages machine.

> As people began to write larger programs, it quickly became apparent that a less error-prone notation was required. Assembly languages were invented to allow operations to be expressed with mnemonic abbreviations. [...] Assembly languages were originally designed with a one-to-one correspondence between mnemonics and machine language instructions [...]. Translating from mnemonics to machine language became the job of systems program known as an assembler. (2015, p. 15-16)

Ces langages de bas niveau sont constitués de *mnémoniques*, c'est-à-dire de symboles représentant les séquences binaires les plus courantes. Un programme nommé *assembleur* est ensuite employé pour traduire le programme en langage machine qui pourra par la suite être exécuté. Les langages d'assemblage ont sensiblement les mêmes avantages et inconvénients que les langages machine :

> The assembly language is specific to a processor family and environment, and it is therefore not portable. They require considerably more programming effort than high-level programming languages, and are more difficult to use for larger applications. A program written in assembly language often needs to be rewritten for a different platform. However, since the assembly language is in the native language of the processor, it has significant speed advantages. (O'Regan, 2021, p. 178)

Les langages d'assemblage sont donc exécutés rapidement, mais malgré une syntaxe adaptée aux programmeurs, ils restent spécifiques au matériel et rendent difficile la programmation d'applications importantes. À titre d'exemple, nous représentons dans la Figure 1-3 le programme PGCD en langage d'assemblage x86.

```
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $4, %esp
        andl    $-16, %esp
        call    getint
        movl    %eax, %ebx
        call    getint
        cmpl    %eax, %ebx
        je      C
A:      cmpl    %eax, %ebx
        jle     D
        subl    %eax, %ebx
B:      cmpl    %eax, %ebx
        jne     A
C:      movl    %ebx, (%esp)
        call    putint
        movl    -4(%ebp), %ebx
        leave
        ret
D:      subl    %ebx, %eax
        jmp     B
```

Figure 1-3 – Programme PGCD en langage d'assemblage x86 (Scott, 2015, p. 5)

L'inconvénient majeur des langages d'assemblage et qu'il faut réécrire les programmes pour les adapter à chaque nouvelle machine.

Langages de haut niveau

C'est pour palier à cette difficulté que furent introduits les *langages de haut niveau,* à la fois indépendants de la machine et exprimés dans un langage plus proche du langage naturel. Scott (2015) en expose les principes :

> Translating from a high-level language to assembly or machine language is the job of a systems program known as a compiler. Compilers are substantially more complicated than assemblers because the one-to-one correspondence between source and target operations no longer exist when the source is a high-level language. (2015, p. 6)

Les programmes écrits en langage de haut niveau doivent donc d'abord être compilés par un programme nommé *compilateur* qui les traduit dans le langage machine du matériel qui doit les exécuter. Notons que le compilateur présente l'avantage d'être écrit, une fois pour toutes, pour chaque type de machine ciblé pour l'exécution. Comme l'indiquent Gabbrielli & Martini (2010), les avantages des langages de haut niveau sont nombreux : « these were designed as abstract languages which would ignore the physical characteristics of the computer and were instead suited to express algorithms in a way that was relatively easy for the human user » (2010, p. 415). L'édition des programmes est donc indépendante des machines (même s'ils doivent être compilés spécifiquement pour chaque type de matériel) et la syntaxe est plus facilement compréhensible par les humains. Pour comparaison, nous donnons à voir dans la Figure 1-4 le programme PGCD écrit dans le langage de haut niveau C.

```
int gcd(int a, int b) {                                    // C
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```

Figure 1-4 – Programme PGCD en langage de haut niveau C (Scott, 2015, p. 13)

Certains langages de haut-niveau comme Pascal ou plus récemment Java et C# ont la particularité de proposer une *machine virtuelle* pour chaque architecture matérielle. Cette fonctionnalité permet d'écrire *et* de compiler les programmes une fois pour toutes. Ils pourront être ensuite exécutés sur toutes les plateformes (qui disposent d'une machine virtuelle) indépendamment des aspects matériels.

Enfin, depuis quelques années, les langages de haut niveau *interprétés* comme Python ou Javascript rencontrent un succès grandissant. Scott (2015) en donne la description suivante :

> An alternative of implementation for high-level languages is known as interpretation. Unlike a compiler, an interpreter stays around for the execution of the application. […]. The interpreter reads statements in that language more or less one at a time, executing them as it goes along. In general, interpretation leads to greater flexibility and better diagnostics (error messages) than does compilation. […]. Compilation, by contrast, generally leads to better performance. (2015, p. 17)

Les langages interprétés sont donc directement exécutés, sans compilation préalable, par un *interpréteur* qui opère ligne par ligne en exécutant immédiatement les instructions machines correspondantes. L'avantage de ces langages est une grande souplesse et une correction d'erreurs facilitée. Le revers de la médaille est une exécution plus lente des programmes que s'ils avaient été préalablement compilés puis exécutés en langage machine.

Synthèse de la sous-section

Q1.3.2 : Quels sont les différents niveaux d'abstraction de la machine permis par les langages de programmation ?

En nous basant sur des ouvrages généralistes traitant de programmation, nous avons établi les différents **niveaux d'abstraction** des langages de programmation par rapport à la machine :

- les **langages machine** sont de très bas niveau, ils permettent de contrôler directement les machines à l'aide de séquences binaires, ils sont très performants mais très difficiles à mettre en œuvre ;
- les **langages d'assemblage** sont formés de mnémoniques traduites en langage machine par un assembleur, tout aussi performant, ils sont plus faciles à appréhender que les langages machine ;
- les **langages de haut niveau** ont une syntaxe plus riche et plus proche du langage naturel et sont indépendants de la machine, ils sont traduits en langage machine par un compilateur avant d'être exécutés ou directement interprétés par un interpréteur.

Nous venons de présenter les différents niveaux de programmation, nous exposons dans la section suivante les différents paradigmes mis en œuvre par les langages de programmation de haut-niveau.

1.4.3 Les principaux paradigmes de programmation

La notion de paradigme de programmation fut introduite pour la première fois en 1978 par Floyd : « I want to talk about the paradigms of programming, how they affect our success as designers of computer programs, how they should be taught, and how they should be embodied in our programming languages » (1978, p. 455). En s'appuyant sur le modèle des paradigmes scientifiques décrits par Kuhn (1970), il proposa à travers ce terme de catégoriser les différents courants de programmation ayant cours dans l'objectif de développer leur enseignement, leurs implémentations dans des langages, ainsi que leur diffusion au sein de la communauté des programmeurs. De nos jours, cette classification paradigmatique est toujours utilisée. Ainsi, Krishnamurthi et Fisler (2019) nous en donnent une définition contemporaine : « Paradigms are supposedly groups that differentiate one class of similar languages from others in some high-level way, usually focused on features that exhibit common behaviors » (2019, p. 5). Ce sont donc les fonctionnalités offertes par un langage qui définissent de quel paradigme il relève. Les auteurs poursuivent en listant les paradigmes majeurs que l'on rencontre actuellement dans la littérature scientifique : « Authors conventionally list a few major paradigms: imperative, object-oriented, functional, and logic » (Krishnamurthi & Fisler, 2019, p. 5). Décrivons en quelques mots ces quatre paradigmes.

Quatre paradigmes de programmation

Dans un cours de programmation, Leavens (1997) présente chacun d'entre eux à travers une courte définition. Il décrit d'abord la *programmation impérative* : « The

imperative programming paradigm assumes that the computer can maintain through environments of variables any changes in a computation process. Computations are performed through a guided sequence of steps, in which these variables are referred to or changed » (Leavens, 1997). Ce que Nørmark (2013) résume, quelques années plus tard dans un autre cours de programmation, par la phrase : « First do this and next do that » (Nørmark, 2013).

Leavens poursuit en décrivant la *programmation logique* : « The logical paradigm takes a declarative approach to problem-solving. Various logical assertions about a situation are made, establishing all known facts. Then queries are made. The role of the computer becomes maintaining data and logical deduction » (Leavens, 1997). Ce qui est résumé par Nørmark dans la formule : « Answer a question via search for a solution » (Nørmark, 2013).

La *programmation fonctionnelle* est également dépeinte par Leavens : « The functional programming paradigm views all subprograms as functions in the mathematical sense, they take in arguments and return a single solution. The solution returned is based entirely on the input » (Leavens, 1997). Nørmark résume cela dans l'assertion : « Evaluate an expression and use the resulting value for something ».

Enfin, la *programmation orientée objet* est décrite par Leavens : « Object Oriented Programming is a paradigm in which real-world objects are each viewed as separate entities having their own state which is modified only by built in procedures, called methods » (Leavens, 1997). Ce qui tient dans la proposition : « Send messages between objects to simulate the temporal evolution of a set of real world phenomena » (Nørmark, 2013).

Impératif vs déclaratif

D'autres auteurs adoptent cette classification en quatre paradigmes en les divisant en deux méta-catégories : le groupe des paradigmes *impératifs* d'un côté et le groupe des paradigmes *déclaratifs* de l'autre. Ils les dissocient à l'aune de la sémantique de leurs instructions, ainsi selon Scott (2015) :

> The many existing languages can be classified into families based on their model of computation. […]. The top-level division distinguishes between the *declarative* languages, in which the focus is on *what* the computer is to do, and the *imperative* languages, in which the focus is on *how* the computer should do it [mis en valeur par l'auteur]. (2015, p. 10)

C'est également le choix retenu par les contributeurs de l'encyclopédie en ligne Wikipédia qui distingue une première catégorie de paradigmes qualifiés d'impératifs qui contient la programmation procédurale et la programmation orientée objet : « Imperative [programming languages] in which the programmer instructs the machine how to change its state : procedural which groups instructions into procedures ; object-oriented which groups instructions with the part of the state they operate on » (« Programming Paradigm », 2023) et une deuxième catégorie de paradigmes, désignés comme déclaratifs, qui englobe la programmation fonctionnelle et la programmation logique : « Declarative [programming languages] in which the programmer merely declares properties of the desired result, but not how to compute it : functional in which the desired result is declared as the value of a series of function

applications ; logic in which the desired result is declared as the answer to a question about a system of facts and rules » (« Programming Paradigm », 2023).

Pour les tenants de cette classification, l'adjectif « impératif » qualifiant un groupe de paradigmes, le paradigme du même nom est désigné différemment afin de lever toute ambiguïté. On rencontre par exemple le qualificatif « procedural programming » défini par Ghezzi et Jazayeri (2008) comme : « This is the conventional programming style, where programs are decomposed into computation steps that perform complex operations. Procedures and functions (collectively called routines) are used as modularization units to define such computation steps » (2008, p. 20). Scott (2015) utilise la locution « von Neumann languages » qu'il caractérise de la sorte : « [They are] the most familiar and successful [...] in which the basic means of computation is the modification of variables. von Neumann languages are based on statements that influence subsequent computation via the side effect of changing the value of memory » (2015, p. 11-12). Dans cette thèse, nous utilisons la dénomination *programmation procédurale* pour désigner la programmation impérative, et le terme « impératif » pour qualifier le groupe de paradigme contenant la programmation procédurale et la programmation orientée objet.

Compléments

Ajoutons trois remarques complémentaires. Premièrement, il existe des langages de programmation, non nécessairement Turing-complets, qui ne sont pas classables dans l'un de ces quatre paradigmes. Citons, par exemple, les langages de requêtes dans les bases de données (SQL, XQuery, MQL, etc.) ou les langages permettant de décrire des contenus[6] (HTML, XML, JSON, etc.). Deuxièmement, les frontières entre ces paradigmes sont poreuses et la plupart des langages de programmation offrent la possibilité de mettre en œuvre plusieurs paradigmes. Python permet par exemple de faire de la programmation objet, fonctionnelle ou procédurale. Troisièmement, certains auteurs considèrent la programmation parallèle, la programmation évènementielle ou la programmation structurée comme formant des paradigmes de programmation. Nous parlons plutôt dans cette thèse de *méthodes de programmation*, jugeant qu'elles s'appuient sur les autres paradigmes et qu'elles n'introduisent pas de spécificités justifiant d'en constituer un en tant que tel.

Synthèse de la sous-section

Q1.3.3 : Quels sont les principaux paradigmes de programmation ?

En partant de cours et d'ouvrages généralistes ayant pour objet la programmation informatique, nous avons distingué les principaux **paradigmes de la programmation** de haut niveau :

- La **programmation impérative** consiste à décrire *comment* la machine doit procéder pour implémenter un algorithme, elle est composée de la **programmation procédurale** dans laquelle des séquences d'instructions modifient l'état de variables en mémoire, et de

[6] On peut même se demander si ces langages de description sont des langages de programmation comme nous les avons définis dans la mesure où ils ne décrivent pas réellement d'algorithmes.

la **programmation orientée-objet** qui consiste à définir et à décrire les interactions de briques logicielles appelées objets ;
- La **programmation déclarative** consiste à décrire *ce que* la machine doit faire pour mettre en œuvre un algorithme, elle comprend la **programmation logique** qui s'appuie sur un ensemble de faits et de règles qui interagissent dans un moteur d'inférence, et de la **programmation fonctionnelle** basée sur l'unique évaluation de fonctions mathématiques.

Ayant précisé les différents paradigmes de programmation des langages de haut niveau, nous donnons maintenant un aperçu de leur filiation historique dans la section suivante.

1.4.4 Éléments historiques

Comme annoncé au début du chapitre, l'un des objectifs est de prendre de la distance avec les représentations épistémologiques contemporaines de la programmation informatique. Nous allons donc lui donner une historicité en présentant un aperçu des principaux langages de haut niveau et en mettant en lumière la filiation qui les relie. Nous nous basons principalement sur l'ouvrage d'O'Regan « A Brief History Of Computing » (2021), ainsi que sur les ouvrages généralistes cités précédemment dans ce chapitre (Gabbrielli & Martini, 2010; Ghezzi & Jazayeri, 2008; Mitchell, 2003; Scott, 2015; Sebesta, 2012)

Depuis 1957 et la création du premier langage de programmation de haut niveau, des milliers de langages ont été proposés et mis en œuvre. Seuls quelques dizaines d'entre eux se sont largement diffusés dans le monde académique et industriel. La plupart des nouveaux langages de programmation apparaissent en amendant et en améliorant un langage préexistant. Il est donc possible d'établir leur l'arbre généalogique. Nous reproduisons à titre d'exemple dans la Figure 1-5 un arbre proposé par Sebesta (2012) représentant la filiation des principaux langages de haut niveau.

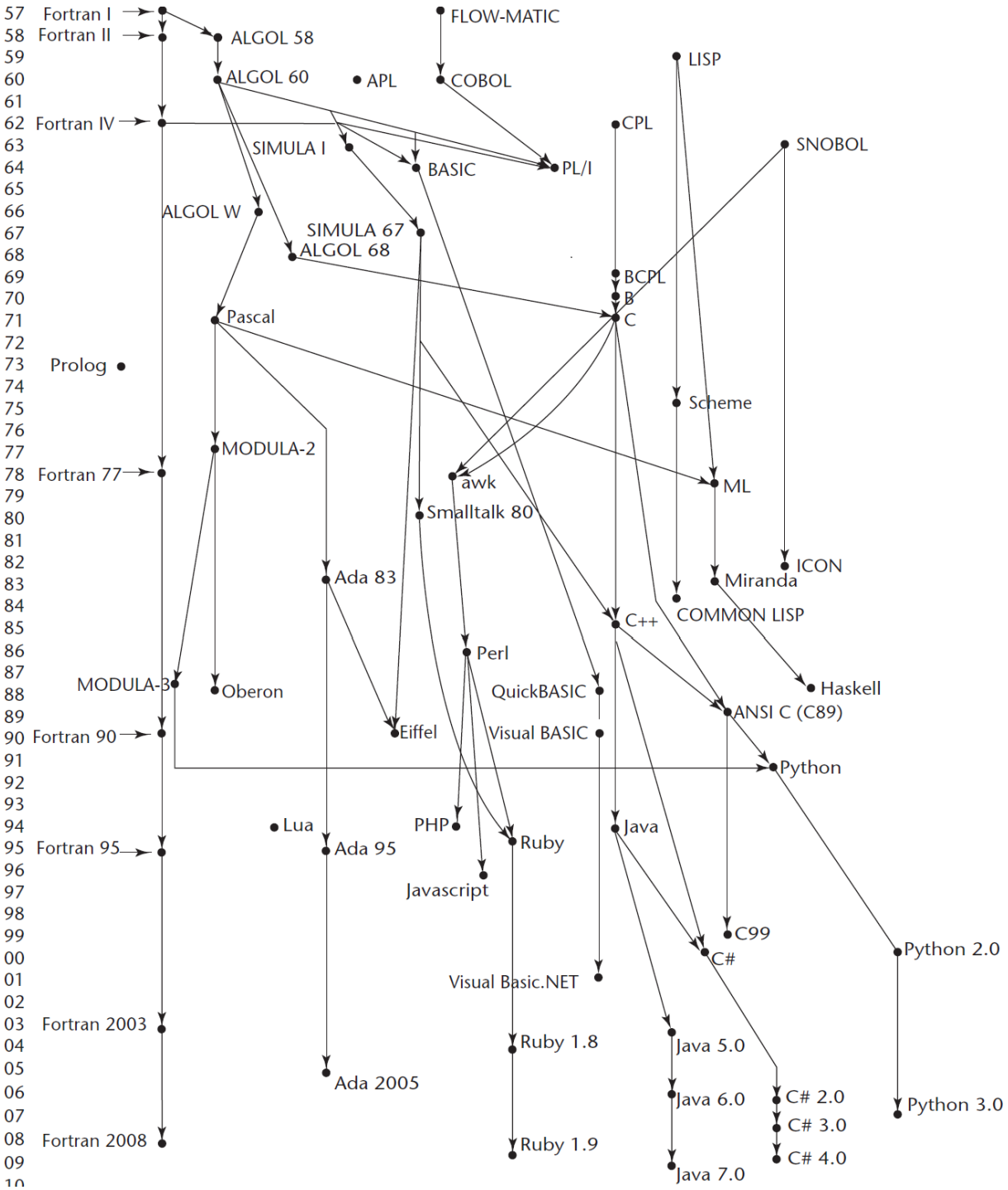


Figure 1-5 – Généalogie des principaux langages de programmation de haut niveau (Sebesta, 2012)

Il ne s'agit pas ici d'analyser cet arbre dans les moindres détails, mais plutôt d'y observer le fait que les langages modernes dérivent d'un même noyau de langages pionniers apparus dans les années 50 et 60. Nous allons, dans la suite, donner un rapide aperçu des langages qui ont fait l'histoire pour chaque paradigme de programmation.

Langages procéduraux

Commençons par les langages procéduraux. *Fortran* (FORmula TRANslator) a été développé au milieu des années 50 et est le premier langage de programmation de

haut niveau opérationnel. Sa principale innovation a été de rendre possible l'utilisation de la notation mathématique dans les expressions. Il est toujours utilisé de nos jours pour le calcul numérique de haute performance.

Algol (ALGOrithmic Language) a ensuite été conçu à la fin des années 50 comme un langage de programmation universel pour les applications scientifiques. Peu utilisé dans la sphère industrielle, il fut cependant prédominant dans le monde universitaire dans les années 60 et 70 et a eu un impact important sur la conception des langages de programmation qui lui succédèrent.

Cobol (COmmon Business Oriented Language) a été créé dans les mêmes années comme un langage de programmation destiné aux entreprises et aux applications commerciales. Sa syntaxe se voulait aussi proche que possible du langage naturel qu'est l'anglais. Il est encore marginalement en usage, en particulier dans le secteur bancaire et assurantiel.

Au cours des années 60, George Kemeny et Thomas Kurtz ont conçu le langage de programmation *Basic* (Beginner's All Purpose Symbolic Instruction Code). Étant facile à apprendre et ayant un syntaxe compacte pouvant être mise en œuvre sur des ordinateurs dotés d'une petite mémoire, il fut très populaire pour programmer les « micro-ordinateurs » à la fin des années 70 et au début des années 80.

En 1970, Niklaus Wirth a développé le langage *Pascal* nommé en l'honneur du mathématicien Blaise Pascal et de ses travaux sur le calcul mécanisé. Ce langage a eu un fort impact en devenant, à partir du milieu des années 70 et jusqu'au milieu des années 90, le langage de référence pour l'enseignement de la programmation. Pascal fut également le premier langage à proposer une machine virtuelle lui conférant une grande portabilité.

Un autre langage de programmation important fut développé au début des années 70 par Denis Ritchie et Ken Thompson au Bell Labs, il s'agit du langage *C*. Conçu à l'origine comme un langage système pour Unix, il est rapidement devenu un langage à usage général qui s'est considérablement développé dans l'industrie durant les années 80. Cela s'explique par une certaine flexibilité, en particulier concernant le typage, et par la grande disponibilité d'ordinateurs fonctionnant sous le système d'exploitation UNIX.

Langages orientés objet

Les langages orientés objet trouvent leur origine à la fin des années 60 dans la création du langage *Simula* par Kristen Nygaard et Ole-Johan Dahl au Norwegian Computing Center d'Oslo. Développé comme une extension d'Algol, Simula est le premier langage à introduire les notions de classe, d'objet et d'héritage. En avance sur son temps, il a eu une forte influence sur ses successeurs.

Ainsi, *Smalltalk*, développé dans les années 1970 par Alan Kay au Xerox PARC, est le premier langage tout-objet (du nombre entier à l'entité la plus complexe). Tous les calculs y sont effectués par la même technique consistant à envoyer un message à un objet pour invoquer l'une de ses méthodes. C'est, de plus, le premier langage orienté objet à rencontrer un succès commercial.

En 1985, Bjorne Stroustrup élabora le langage *C++* au Bell Labs. Son objectif était d'ajouter les notions de classes et d'héritage au langage C sans porter préjudice à son

l'efficacité et sans compromettre la compatibilité des programmes. D'abord conçu sans intention commerciale, l'intérêt pour la programmation orientée objet s'étant accru au cours des années 80, C++ s'est rapidement diffusé jusqu'à devenir le langage orienté objet le plus utilisé dans les années 90.

Enfin, en 1995, le langage *Java* fut initié par un groupe dirigé par Jim Gosling chez Sun Microsystems. Ce groupe avait pour objectif initial de définir une nouvelle implémentation de C++ destinée aux appareils électroniques grand public (fours à micro-ondes, télévisions interactives, etc.), avec des objectifs de conception alliant sécurité et portabilité. Java s'est d'abord imposé dans le développement de petites applications (applet) destinées à être exécutées dans les navigateurs Internet avant de devenir un des langages de programmation généraliste le plus utilisé actuellement.

Langages interprétés

Les langages interprétés[7] furent d'abord utilisés comme « langages de script » pour effectuer des tâches utilitaires telles que le lancement et la coordination de l'exécution de programmes (Bash) ou l'analyse de texte à l'aide d'expressions régulières (Perl). De par leur syntaxe simple et leur souplesse d'utilisation, certains d'entre eux sont devenu des langages polyvalents parmi les plus populaires.

Ainsi, en 1991, Guido van Rossum développa au CWI d'Amsterdam le langage *Python*. D'abord conçu comme un langage de script pour les systèmes distribués, Python se développa jusqu'à devenir actuellement l'un des langages les plus utilisés dans le monde. Il s'est particulièrement développé dans les domaines de la science des données, de l'intelligence artificielle, des services Web et de l'enseignement. Python a pour particularité le recours à l'indentation afin de différencier les regroupements syntaxiques dans les programmes.

Au milieu des années 90, alors qu'Internet et le Web se développent, le besoin d'un langage permettant de dynamiser les pages HTML statiques se fait ressentir. C'est pour répondre à cette attente que Brendan Eich, alors en poste chez Netscape Corp., conçu le langage *Javascript*. Eich avait initialement appelé sa création LiveScript, mais Netscape a choisi de la renommer dans le cadre d'un accord de marketing avec Sun Microsystems (Javascript n'a cependant aucun lien avec Java au-delà de quelques similitudes syntaxiques). Le navigateur Internet de Netscape devint leader du marché à partir de 1995, Javascript fut du même coup adopté comme un standard d'Internet. De nos jours, Javascript est parmi les langages les plus utilisés dans l'industrie du Web.

Langages fonctionnels

La conception des langages de programmation fonctionnels a été largement influencée par les travaux pionnier d'Alonzo Church sur le λ-calcul dans les années 30. Ainsi, le langage *Lisp* (LISt Processor) a été conçu en 1960 par un groupe dirigé par John Mc-Carthy au MIT. Ce premier langage de programmation fonctionnel a été inventé afin de permettre le traitement de données symboliques dans des listes chaînées. Ce besoin est né des premières applications dans le domaine de l'intelligence

[7] Nous distinguons ici les langages interprétés bien qu'ils ne constituent pas un paradigme de programmation (ils peuvent être par ailleurs procéduraux ou orientés objet). Néanmoins, il est commode de les regrouper d'un point de vue historique, du fait de leur destiné commune.

artificielle : traitement du langage naturel, modélisation du cerveau humain, démonstration automatisée de théorèmes, etc.

Le langage *Scheme* est né au MIT en 1975 suite aux travaux de Gerald Sussman et Guy Steele. En vertu de sa syntaxe extrêmement simple, composé d'un nombre très limité de mots-clés, Scheme a surtout eu des applications éducatives, telles que les cours de programmation fonctionnelle et les introductions générales à la programmation. Suivirent dans les années 80 et 90, plusieurs langages fonctionnels tels que *Haskell*, ou *OCaml* qui restent de nos jours surtout orientés vers l'enseignement et le monde académique.

Langages logiques

Enfin, le langage de programmation logique le plus connu est *Prolog* (PROgramming LOGic). Il a été développé au début des années 1970 par Alain Colmerauer et Robert Kowalski. Dans les années 80, un petit groupe d'informaticiens pensait que la programmation logique offrirait un espoir pour supplanter les langages impératifs dans la production d'un grand nombre de logiciels de manière fiable. Jusqu'à présent, ces espérances ne sont pas devenues réalité. Deux raisons majeures expliquent cela, tout d'abord, comme pour d'autres approches non impératives, les programmes écrits dans des langages logiques se sont avérés moins efficaces que les programmes impératifs équivalents. Deuxièmement, il a été établi que cette approche n'était vraiment efficace que pour quelques domaines d'application relativement restreints : certains types de systèmes de gestion de bases de données et certains domaines de l'intelligence artificielle.

Synthèse de la sous-section

Q1.3.4 : Quelle filiation historique pour les langages de programmation contemporains ?

En nous appuyant sur des ouvrages retraçant l'histoire des langages de programmation, nous avons établi que les langages de programmation contemporains dérivent d'un ensemble de **langages pionniers** conçus dès les années 50. De plus, nous avons exhibé les filiations des principaux langages en fonctions des paradigmes de programmation comme suit :

- les langages **procéduraux** : **Fortran**, **Algol**, **Cobol**, **Basic**, **Pascal** puis **C** ;
- les langages **orientés objet** : **Simula**, **Smalltalk**, **C++** puis **Java** ;
- les langages **fonctionnels** : **Lisp**, **Scheme**, **Haskell** puis **OCamel** ;
- les langages **logiques** : **Prolog**.

Les langages **interprétés** comme **Python** et **Javascript** avaient initialement un usage utilitaire avant de devenir des langages polyvalents largement utilisés que nous connaissons aujourd'hui.

Suite à la présentation de ces éléments historiques, nous cherchons dans la section suivante à déterminer les concepts fondamentaux qui constituent la programmation impérative.

1.5 Les concepts fondamentaux de la programmation impérative

L'activité de programmation informatique met en jeu un certain nombre de concepts. Nous nous attachons dans cette section à établir le corpus des concepts fondamentaux mobilisables dans une tâche de programmation à l'aide d'un langage de programmation impérative. De la même manière que nous l'avons fait dans la section 1.3, nous analysons ici des référentiels à la fois internationaux , avec le CS13, et français à travers un ouvrage rédigé en 2011 par Dowek et ses collèges nommé « Une Introduction à la Science Informatique pour les enseignants de la discipline au lycée » (ISI11) (Dowek et al., 2011).

1.5.1 Computer Science Curricula

Pour la partie internationale, nous nous basons donc sur le « Computer Science Curricula 2013 » (CSC13) (Task force ACM & IEEE-CS, 2013) que nous avons déjà présenté, et, en particulier, sur le domaine de connaissance D1.15 - Software Development Fundamentals. Aussi, nous présentons ci-dessous la synthèse des concepts listés dans les deux sous-domaines « Fundamental Programming Concepts » et « Fundamental Data Structures » :

- (C1.1) basic syntax and semantics of a higher-level language ;
- (C1.2) variables and primitive data types (e.g., numbers, characters, Booleans) ;
- (C1.3) expressions and assignments ;
- (C1.4) simple Input/Output including file I/O ;
- (C1.5) conditional and iterative control structures ;
- (C1.6) functions and parameter passing ;
- (C1.7) the concept of recursion ;
- (C1.8) fundamental data structures : arrays, records, string, abstract type (stacks, queues, sets, maps, linked lists).

1.5.2 Introduction à la Science Informatique

Pour la France, nous prenons comme référence l'ISI11 (Dowek et al., 2011). Co-écrit par des chercheurs en informatique, cet ouvrage est paru juste avant la mise en place de la spécialité Informatique et Science du Numérique (ISN) en terminale scientifique à la rentrée 2012. Il est conçu comme une ressource de référence pour les futurs enseignants d'ISN et aborde de façon synthétique les différents domaines de la science informatique. Afin d'établir notre corpus fondamental, nous listons ci-après les concepts présentés dans le chapitre « Langages et programmation » :

- (C2.1) affectation, déclaration et portée des variables ;

- (C2.2) type scalaires : entier, flottants, booléen, caractère ;
- (C2.3) séquence, test et boucles for et while ;
- (C2.4) les entrées/sorties ;
- (C2.5) la notion de fonction d'argument et de retour ;
- (C2.6) la récursivité ;
- (C2.7) types composites : chaînes de caractères, enregistrements, tableaux, type dynamique (listes, arbres).

1.5.3 Concepts fondamentaux

En parcourant les deux listes que nous venons d'établir, nous pouvons d'abord constater que les ensembles de notions sont très proches. Dans le détail, nous retrouvons le concept de variable (C1.2, C1.3 et C2.1) avec la déclaration, l'affectation et, spécifiquement dans l'ISI11, la portée. Les types qualifiés de primitif (ou scalaire) (C1.2 et C2.2) sont présents dans les deux référentiels et sont composés de façon identique : nombres, caractères et booléens. Les structures de contrôle énumérées sont la conditionnelle et les deux types de boucles (C1.5 et C2.3). Le concept de fonction est présent (C1.6 et C2.5) associé à celui d'argument (ou de paramètre) et, pour l'ISI11, au concept de retour. La récursivité est également dans les deux listes (C1.7 et C2.6), en lien avec le concept de fonction[8]. Les entrées-sorties d'un programme (C1.4 et C2.4) figurent aussi dans les deux référentiels. Enfin, le concept de types composites (ou structure de données) apparait aussi doublement (C1.8 et C2.7). Il comporte un premier ensemble de types composites que nous qualifions de « simple », qui comprend : les chaînes de caractères, les tableaux (ensemble d'éléments ordonnés et indicés du même type et de taille fixe) et les enregistrements (ensemble de champs nommés de type différents). Suivent des types composites plus complexes, regroupés sous le nom de « types dynamiques » dans l'ISI11 et formalisés dans la dénomination « types abstraits » dans le CSC13. Notons que le CSC13 va plus loin que l'ISI11 en énumérant un nombre plus important de types composites complexes tels que les files, les piles ou les ensembles. Nous retenons ici le seul type en commun : la liste (ensemble d'éléments liés de type différents et de taille variable).

À l'aune de cette analyse, nous établissons le corpus conceptuel fondamental de la programmation impérative comme étant composé des éléments suivants : variable, types primitifs : nombres, caractères et booléens, structure conditionnelle, structures de boucle : for et while, entrées-sorties des programmes, fonction et paramètres, récursivité, types composites simples : chaines de caractères, tableaux et enregistrements et, types composites complexes : listes.

1.5.4 Synthèse de la section

Q1.4 : Quel est le corpus des concepts fondamentaux mis en jeu dans l'activité de programmation avec un langage impératif ?

[8] Bien que fortement en lien avec le paradigme déclaratif de la programmation fonctionnelle, la notion de récursivité peut être mise en œuvre dans tous les langages impératifs modernes.

Après avoir analysé deux référentiels, nous avons établi l'ensemble des **concepts fondamentaux de la programmation impérative**. Ce corpus comprend les concepts suivants : **variable**, **types primitifs** : nombres, caractères et booléens, **structure conditionnelle**, **structures de boucle** : for et while, **entrées-sorties** des programmes, **fonction** et paramètres, **récursivité**, **types composites simples** : chaines de caractères, tableaux et enregistrements et, **types composites complexes** : listes.

Dans ce chapitre, nous avons mené une analyse épistémologique relative aux concepts visés par l'enseignement de la programmation informatique. Notre objectif était à la fois de fixer le vocabulaire, de prendre un double recul spatial et temporel sur les savoirs en jeu, et d'expliciter et de délimiter le savoir savant afin de pouvoir en étudier la transposition didactique. Pour cela, nous avons caractérisé l'informatique selon quatre niveaux hiérarchiques, avant de circonscrire le niveau de la science informatique à travers dix domaines de connaissance. Par la suite, nous avons défini rigoureusement les langages de programmation informatique puis exposé les aspects liés au niveaux d'abstraction de la machine, aux paradigmes de programmation, ainsi qu'à la genèse des principaux langages. Enfin, nous avons établi le corpus des concepts fondamentaux de la programmation informatique.

Dans le chapitre suivant, nous analysons les textes officiels qui régissent l'enseignement de la programmation informatique en France en troisième et en seconde.

Chapitre 2

Analyses curriculaires en France à la transition collège-lycée

Dans ce chapitre nous étudions la place de la programmation informatique dans les programmes scolaires français à la fin du collège et au début du lycée.

En France, depuis 2014, et au moment où nous écrivons ces lignes, la scolarité de l'école maternelle à la fin de collège est organisée en quatre cycles pédagogiques successifs (MEN, 2013) :

- le cycle 1, des apprentissages premiers, correspond aux trois niveaux de l'école maternelle (preschool et kindergarten) ;
- le cycle 2, des apprentissages fondamentaux, couvre les trois premières années de l'école élémentaire : CP, CE1 et CE2 (grade 1, 2 et 3) ;
- le cycle 3, de consolidation, correspond aux deux dernières années de l'école élémentaire et à la première année du collège : CM1, CM2 et 6ème (grade 4, 5 et 6) ;
- le cycle 4, des approfondissements, regroupe les trois dernières années du collège : 5ème, 4ème et 3ème (grade 7,8 et 9).

Ensuite, les trois dernières années de l'enseignement secondaire correspondent au lycée qui est organisé en trois filières : générale, technologique et professionnelle et en trois niveaux : seconde[9], première et terminale (grade 10, 11 et 12).

Comme nous l'avons présenté en introduction, nous nous plaçons dans cette thèse à la transition entre la fin de collège, c'est à dire la classe de troisième, et le début du lycée général et technologique, à savoir la classe de seconde.

Commençons par présenter succinctement l'historique d'introduction de l'informatique dans l'enseignement secondaire français. Cette introduction a fait l'objet d'une histoire longue d'un demi-siècle. Pour en donner un aperçu, nous nous appuyons sur la Figure 2-1 qui présente une frise chronologique allant des années 1970 jusqu'à nos jours. Nous nous focalisons particulièrement sur les niveaux qui nous intéressent dans cette thèse : la troisième et la seconde.

[9] Le niveau seconde est commun aux filières générales et technologiques, la filière professionnelle propose des secondes différenciées pour chaque « famille de métiers ».

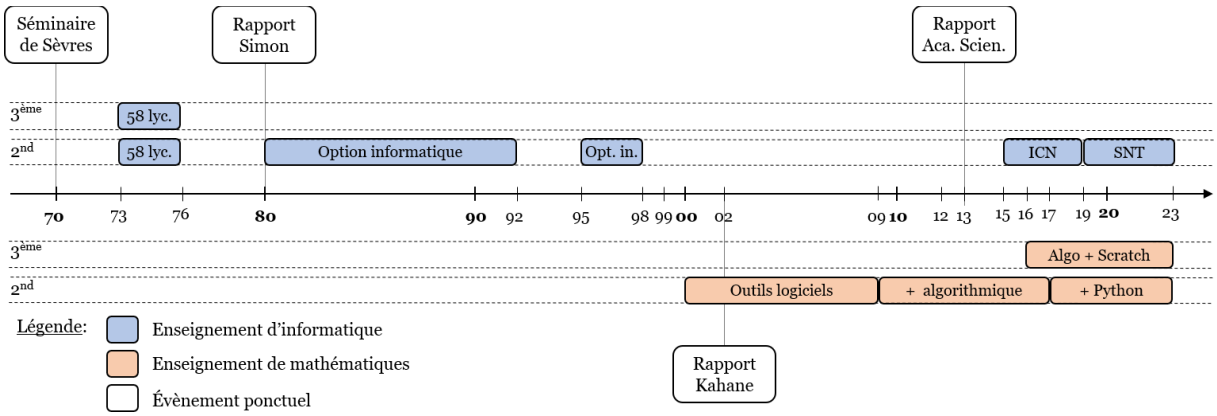


Figure 2-1 – Frise chronologique résumant les différents enseignements de l'informatique en troisième et en seconde, ainsi que les évènements ponctuels ayant influencé leur avènement

En 1970, le séminaire de Sèvres marque le début de l'introduction de l'informatique dans l'enseignement secondaire français. Les débats contradictoires autour de deux positions antagonistes ne permettront pas de trancher une question qui reviendra régulièrement durant les cinquante années suivantes, à savoir si l'introduction de l'informatique doit être : « envisagé[e] comme enseignement ayant son propre statut, ou intégré dans une autre discipline d'enseignement » (Baudé, 2017, p. 122). Nous retrouvons cette distinction chez Douady (1984) qui avance qu'un ensemble de savoirs peut avoir le statut d'*outil* lorsqu'il a une disponibilité fonctionnelle permettant de résoudre des problèmes ou d'interpréter de nouvelles questions, et le statut d'*objet* lorsqu'il est décontextualisé et fait partie d'un corpus scientifiquement et socialement reconnu.

Suite à ce séminaire, le ministère de l'Éducation nationale lança en 1973 un premier essai qui restera comme « l'expérimentation des 58 lycées ». Il instaure à cette occasion une formation ambitieuse à l'informatique pour plus de 500 enseignants et équipe en « mini-ordinateurs » 58 collèges et lycées dans l'objectif de les utiliser comme « outils pédagogiques » dans différentes disciplines (on parle à l'époque d'enseignement assisté par ordinateur). L'expérimentation prit fin en 1976 suite à des coupes budgétaires liées à la crise économique (Baudé, 2014a).

Plus tard, en 1980, le rapport Simon est suivi du colloque « Le mariage du siècle : éducation et informatique » qui se prononce pour un enseignement de l'informatique en tant qu'objet (Baudé, 2019). Il est suivi par la création d'une option informatique autonome dans 75 lycées qui fut généralisée en 1985, et qui fit d'objet d'une épreuve au baccalauréat à partir de 1988. Cette option connait ensuite une série de suppressions-réapparitions au gré des différentes alternances politiques, jusqu'à sa disparition totale en 1998 (Baudé, 2014b).

Le début des années 2000 marque le retour de l'informatique outil avec l'avènement du Brevet Informatique et Internet (B2I) axé uniquement sur les usages. C'est le début du « désert explicatif » où l'école « prétendait donner aux élèves la culture informatique correspondant aux besoins de l'époque par le biais de l'utilisation de l'informatique dans les autres disciplines » (Baudé, 2021, p. 7). La suite de la décennie connait une importante activité de lobbying d'un groupe composé d'enseignants du secondaire (Jean-Pierre Archambault et Jacques Baudé) et

d'universitaires (Maurice Nivat, Gérard Berry, Gilles Dowek et Serge Abiteboul) ayant pour objectif de faire advenir dans le secondaire une discipline informatique indépendante et un corps d'enseignants spécialisés (Donnard, 2021).

En 2012, ce travail d'influence commence à porter ses fruits avec la création de plusieurs options dont la spécialité de terminale scientifique Informatique et Sciences du Numérique (ISN) qui sera ensuite généralisée à toutes les filières en première et terminale. En 2015, apparait dans le même esprit en seconde l'enseignement d'exploration optionnel Informatique et Création Numérique (ICN) (Froidevaux, 2023).

L'année suivante, afin d'orienter les politiques éducatives vers la constitution d'une discipline informatique touchant le plus grand nombre, le rapport de l'académie des sciences sous-titré « il est urgent de ne plus attendre » appelle à « mettre en place un enseignement de science informatique depuis le primaire jusqu'au lycée, orienté vers la compréhension et la maîtrise de l'informatique, et dépassant donc largement les seuls usages des matériels et logiciels » (Berry et al., 2013, p. 5).

Enfin, 2019 marque l'aboutissement d'un long combat pour les informaticiens (Donnard, 2021) avec la création de la discipline obligatoire Sciences Numériques et Technologie (SNT) en seconde (1h30), et de la spécialité Numérique et Sciences Informatiques (NSI) dotée comme telle d'un important volume horaire en première (4h) puis en terminale (6h). L'instauration de ces disciplines informatiques est accompagnée de la création d'un Certificat d'Aptitude au Professorat de l'Enseignement du Second degré (CAPES) NSI en 2020, puis d'une agrégation d'informatique en 2022.

En parallèle de cette introduction progressive de l'informatique en tant que discipline autonome, les mathématiciens essayèrent de s'approprier l'informatique pour en faire un outil au service de leur matière (Donnard, 2021). Ainsi, en 2000, est d'abord ajouté dans le programme de seconde (MEN, 1999) l'utilisation de différents outils logiciels : logiciel de construction en géométrie, logiciel grapheur en analyse, et tableur en arithmétique et en probabilités-statistiques.

Suite aux travaux de la Commission de Réflexion sur l'Enseignement des Mathématiques (CREM), le « rapport Kahane » encourage à aller plus loin en introduisant des notions algorithmiques à travers des activités mathématiques de programmation dans le but d'éveiller l'intérêt des élèves à travers la manipulation concrète d'objets mathématiques :

> Le point du vue constructif et expérimental confère au objets mathématiques une existence concrète dès lors qu'on les manipule ou les anime sur ordinateurs. C'est l'occasion d'une découverte personnelle par l'élève de « phénomènes » mathématiques, propre à susciter l'intérêt de bon nombre d'entre eux. (CREM, 2003, p. 2)

En 2009, sous l'impulsion d'un groupe d'inspecteurs généraux menés par Robert Cabane (Donnard, 2021), le nouveau programme de mathématiques de seconde (MEN, 2009a) voit la liste des outils logiciels à utiliser s'agrandir : logiciel de calcul formel pour l'algèbre, et logiciel de visualisation 3D pour la géométrie dans l'espace. En complément, ce groupe introduit pour la première fois des activités algorithmiques nécessitant l'écriture de programmes dans le cadre d'exercices de mathématiques : tracé de courbes, recherche de racines, résolution d'équations, etc. Afin de formaliser

ces algorithmes mathématiques, des notions algorithmiques font également partie des nouveautés introduites (variable, boucles, etc.). Même si ces notions ont d'une certaine manière le statut d'objet en étant présentées de manière décontextualisée dans une section dédiée du programme, elles ont surtout le statut d'outils utilisés pour résoudre des problèmes mathématiques. Aucun langage d'implémentation n'est évoqué.

Dans le cadre de la réforme du collège menée en 2016, la programmation informatique fait également son apparition dans les programmes de mathématiques du collège (MEN, 2015). Les activités algorithmiques prescrites sont sans lien direct avec le reste du programme et s'appuient sur quelques notions algorithmiques. Elles doivent être implémentées dans un langage de blocs à l'aide du logiciel Scratch.

Enfin, en 2017 le programme de mathématiques de seconde (MEN, 2017a) est aménagé afin d'en améliorer l'articulation avec celui de troisième. À cette occasion, les notions algorithmiques prescrites évoluent et le langage de programmation Python est conseillé pour la mise en œuvre.

Comme cela transparait dans ce récit, l'introduction de l'informatique dans l'enseignement secondaire français a fait l'objet de luttes souterraines entre différents groupes de pression pour défendre ou obtenir une part du « gâteau éducatif » (Donnard, 2021). Nous retrouvons d'une part les représentants des matières scientifiques historiques souhaitant intégrer l'informatique en leur sein, et d'autre part les informaticiens promouvant l'instauration d'une discipline indépendante. Le balancier historique entre l'informatique outil et l'informatique objet a ainsi oscillé au cours de ces 50 années en fonction des alternances politiques (Baron et al., 2014, 2015).

L'objectif de ce chapitre est d'analyser finement la situation contemporaine qui est la résultante de ce processus historique décousu. Nous procédons pour cela à l'étude des différents textes officiels émanant du ministère de l'Éducation nationale qui régissent l'enseignement de l'informatique.

Nous commençons par présenter notre revue de littérature portant sur les aspects curriculaires et les transitions scolaires (section 2.1), puis le cadre théorique que nous avons articulé autour de la Théorie anthropologique du didactique (section 2.2). Suite à quoi, nous détaillons nos questions de recherche et la méthodologie mise en œuvre pour y répondre (section 2.3). Ensuite, nous exposons nos résultats au sujet des disciplines d'enseignement de l'informatique (section 2.4), des habitats de la programmation (section 2.5), des concepts fondamentaux en jeu (section 2.6), des niches de ces concepts (section 2.7), puis des modalités d'enseignement (section 2.8). Nous terminons par une analyse transversale des discontinuités entre le collège et le lycée et des difficultés qui peuvent en découler pour les élèves (section 2.9).

2.1 Revue de littérature

À notre connaissance, les aspects curriculaires de la programmation informatique à la transition collège-lycée n'ont pas été étudiés en tant que tels par la recherche. Nous trouvons cependant dans la littérature des travaux connexes. Nous présentons d'abord des études françaises et internationales portant sur l'introduction de l'informatique dans les curriculums, puis des recherches en didactique des mathématiques traitant du passage du lycée à l'université dans le monde, et enfin des

travaux issus des sciences sociales ou de la didactique des mathématiques centrés sur la transition collège-lycée en France.

2.1.1 Informatique dans les curriculums

Depuis une décennie, l'informatique (« computing ») a été intégrée dans les curriculums de l'enseignement primaire et secondaire (« K-12 ») dans de nombreux pays du monde (European Commission, 2022; Storte et al., 2019). De façon similaire à ce que l'on a connu en France, Hubwieser et ses collègues (2014) avancent que l'on a assisté à un changement de cap global en passant des applications informatiques à l'enseignement de la science informatique pour elle-même. Dans une autre étude, Webb et ses collègues (2017) affirment qu'il n'y a pas, au niveau mondial, de consensus sur les concepts informatiques à enseigner ni sur les méthodes pédagogiques à employer dans le secondaire, néanmoins dans la majorité des pays, la formation professionnelle des enseignants dans le domaine informatique est repérée comme un facteur limitant. Dans un travail plus récent, Oda et ses collègues (2021) montrent que les concepts sont insérés dans les programmes graduellement d'année en année avec un effet cumulatif, et que l'informatique est introduite selon trois approches : comme une matière indépendante, au sein de plusieurs matières, ou dans le cadre de compétences transversales.

En France, la recherche s'est également penchée sur l'introduction curriculaire de l'informatique. Au niveau de l'enseignement primaire, l'informatique est de nouveau enseignée depuis 2016, les objectifs affichés sont l'acquisition de la pensée informatique (Wing, 2006) et la transmission des éléments de culture scientifique et technique permettant de comprendre le monde numérique entourant les élèves (Baron & Drot-Delange, 2016; Vandevelde & Fluckiger, 2020). Concernant le secondaire, comme nous l'avons montré en introduction de ce chapitre, des articles décrivent le mouvement historique de balancier entre l'informatique outil d'enseignement au service des autres disciplines et l'informatique objet enseignée en tant que telle (Baron et al., 2014; Baron & Bruillard, 2011).

À notre connaissance, à l'exception des travaux de Drot-Delange et Tort (2022a) qui portent sur l'éducation aux données dans le programme de SNT, il n'existe pas d'études récentes centrées sur l'intégration de la programmation informatique dans les programmes de l'enseignement secondaire français depuis la création des disciplines SNT et NSI en 2019. Cela constitue donc l'un des objectifs de ce chapitre.

2.1.2 Transition secondaire-supérieur

De multiples recherches ont été menées en didactique des mathématiques, en France comme à l'international, au sujet de la transition secondaire-supérieur. Gueudet et Vandebrouck (2022) en proposent une synthèse. Les auteurs énumèrent les difficultés que peuvent rencontrer les étudiants. Elles peuvent être liées aux contenus mathématiques eux-mêmes : les notions deviennent plus abstraites et les attentes se réfèrent aux pratiques expertes des mathématiciens, les enseignants surestiment les connaissances antérieures des étudiants. Les « cultures institutionnelles » différentes entre le lycée et l'université peuvent également constituer des obstacles : organisation différente de l'espace (amphithéâtres), moindre accessibilité des enseignants, plus grande autonomie dans la prise de notes et le travail

personnel, rythme soutenu des enseignements, moindre utilisation des outils numériques, ressources différentes (polycopié de cours), etc.

Nous retenons ici qu'outre les spécificités liées aux contenus enseignés, il est important de prendre en compte tous les aspects de l'enseignement afin d'étudier la transition d'une institution d'enseignement à une autre.

2.1.3 Transition collège-lycée

De manière générale, la transition du collège au lycée fait l'objet d'un nombre moins important de publications. En France, cela peut s'expliquer par des institutions moins distantes, régies par le même ministère, avec des enseignants ayant la même formation et des cultures institutionnelles assez proches. Cela est sans doute en mesure d'engendrer moins d'obstacles et de difficultés.

Néanmoins, quelques études abordent ce sujet. Loisel Decque (2002), dans sa thèse en sciences de l'éducation portant sur les liens entre la socialisation et la réussite scolaire au collège et au lycée, affirme que le passage de l'un à l'autre provoque des changements à plusieurs niveaux qui nécessitent des adaptations : « l'entrée dans un nouveau cycle de la scolarité, en l'occurrence ici, le passage du collège au lycée, place l'élève devant un ensemble de situations nouvelles, sur le plan spatial, institutionnel, relationnel et scolaire, qui l'amènent à se réorganiser » (2002, p. 10). Une étude menée en 2020 par des chercheurs en sciences sociales au sujet des enseignements en REP+ (Réseau d'Éducation Prioritaire) au collège et au lycée (Richard-Bossez et al., 2020) comporte une analyse des enjeux de la transition troisième-seconde. Les auteurs mettent en avant trois enjeux principaux : assurer la liaison entre deux institutions à l'histoire et l'évolution différentes (tronc commun s'adressant à l'ensemble des élèves d'une classe d'âge d'un côté, volonté première de former la future « élite de la nation » qui s'atténue pour tendre vers la « massification » de l'autre) ; un enjeu de justice sociale pour des catégories d'élèves qui accèdent moins au lycée, notamment pour les élèves de milieux populaires, ceux scolarisés en éducation prioritaire et pour les garçons ; un enjeu pour la poursuite d'études dans le supérieur : les répercussions des décisions d'orientation au moment du passage au lycée marquent fortement le devenir des élèves. Gardons en tête, ici encore, qu'au-delà des enjeux purement scolaires, le passage de la troisième à la seconde revêt de nombreux aspects, en particulier sociaux, qu'il est important de considérer pour comprendre ce qui se joue. Certains de ces aspects dépassent le cadre de cette thèse, nous ne pourrons par conséquent pas en tenir compte.

Finissons par présenter quelques travaux de recherche en didactique des mathématiques à la transition collège-lycée. Majaj (2011) étudie l'enseignement de l'arithmétique en France lors du passage troisième-seconde. Elle établit que les notions d'arithmétiques étudiées au collège ne sont pas réinvesties au lycée et que les notions d'arithmétiques ont une présence « instable » dans les programmes au cours de l'histoire. Alves et ses collègues (2019) s'intéressent à l'articulation de l'enseignement de la notion de fonction au Brésil entre le collège et le lycée. Ils avancent que le « point de vue » change sur cette notion, passant de « relation entre grandeurs » au collège à « relation entre deux ensembles » au lycée. Enfin, une étude centrée sur l'algèbre à la transition collège-lycée en Tunisie (Ben Nejma, 2019) montre qu'il s'agit de passer d'une conception des équations « en termes de quantités connues et de quantités inconnues » à une conception « en termes de variables dépendantes et

indépendantes » et que l'approche de l'algèbre est plus centrée sur la modélisation et la résolution de problème au lycée. Comme nous venons de le voir, la transition du collège au lycée peut également être analysée avec une perspective didactique et épistémique, ce qui est en mesure de révéler de potentiels obstacles pour les élèves.

À la lumière de ces résultats, ayant constaté une absence de travaux de recherche traitant ce sujet, nous allons dans ce chapitre analyser les textes officiels qui régissent l'enseignement de la programmation informatique au collège et en seconde avec une perspective à la fois épistémologique, didactique et pédagogique. Cette large focale doit nous permettre de constater les différences entre ces deux institutions et de faire des hypothèses sur les difficultés qu'elles peuvent engendrer chez les élèves.

La section qui suit présente notre cadre théorique et les outils conceptuels que nous allons utiliser dans la suite.

2.2 Cadre théorique

Nous mobilisons dans ce chapitre différents outils théoriques afin de mener notre analyse des textes officiels. Nous présentons d'abord un premier ensemble de concepts autour de la Théorie anthropologique du didactique (Chevallard, 1992), à savoir : l'approche écologique (Artaud, 1997), l'échelle des niveaux de codétermination didactique (Chevallard, 2002), le modèle des praxéologies (Chevallard, 1999) et ses extensions issues du cadre théorique T4TEL (Chaachoua, 2018). Ensuite, en complément, nous détaillons un des axes d'analyse des disciplines proposé par Reuters (2004) : le fonctionnement pédagogico-didactique.

2.2.1 Autour de la Théorie anthropologique du didactique

La majorité des outils théoriques que nous allons utiliser s'inscrivent dans la Théorie Anthropologique du Didactique (TAD) initiée par Yves Chevallard (1992). Nous allons donc commencer par en présenter les notions fondamentales, en particulier celles de rapport personnel et de rapport institutionnel aux objets de savoir.

Théorie anthropologique du didactique

Chevallard (2003) définit la notion d'*objet* comme une entité, matérielle ou immatérielle, qui existe pour au moins un individu. Il considère donc que tout est objet, y compris les personnes. Il décrit le *rapport personnel* d'un individu donné à un objet comme les interactions que cette individu entretien avec cet objet :

> La deuxième notion fondamentale est celle de *rapport personnel* d'un individu x à un objet o, expression par laquelle on désigne le système, noté $R(x, o)$, de toutes les interactions que x peut avoir avec l'objet o – que x le manipule, l'utilise, en parle, en rêve, etc. [mis en valeur par l'auteur]. (2003, p. 82)

Il détermine la notion d'*institution* de façon générale comme un dispositif social et la *position* d'un individu dans une institution comme la place qu'il y occupe :

> Une *institution I* est un dispositif social « total », qui peut certes n'avoir qu'une extension très réduite dans l'espace social (il existe des « micro-institutions »), mais qui permet – et impose – à ses sujets, c'est-à-dire aux personnes x qui viennent y occuper les différentes *positions p* offertes dans *I*, la mise en jeu de *manière de faire et de penser propres*. Ainsi la *classe* est-elle une institution (dont les deux positions essentielles sont celles de *professeur* et d'*élève*), de même que l'*établissement* (où

> d'autres positions apparaissent : celles de CPE, d'infirmière conseillère de santé, etc.) [mis en valeur par l'auteur]. (Chevallard, 2003, p. 83)

Chevallard parle d'*assujettissement* pour évoquer l'influence des institutions sur les individus, cette influence pouvant prendre la forme d'une soumission ou d'un soutien :

> Le rapport personnel de x à un objet o change – ou se crée, s'il n'existait pas encore – par l'entrée de x dans certaines œuvres O dont l'objet o est constitutif, et qui vivent en certaines institutions I où x vient occuper une certaine position p. Dès sa naissance, tout individu est ainsi *assujetti* à – c'est-à-dire à la fois *soumis* à et *soutenu* par – de multiples institutions, telle sa *famille*, dont il devient le *sujet*. [mis en valeur par l'auteur]. (Chevallard, 2003, p. 83)

Pour influencer les individus, l'institution établit ce que devrait être le rapport personnel d'un individu idéal et imaginaire à un certain savoir, c'est le *rapport institutionnel* :

> Étant donné un objet o, une institution I, et une position p dans I, on appelle *rapport institutionnel* à o en position p, et on note $R_I(p, o)$, le rapport à l'objet o qui devrait être, idéalement, celui des sujets de I en position p. Dire que x est un *bon sujet* de I en position p, c'est dire que l'on a $R(x, o) \cong R_I(p, o)$, où le symbole $\cong$ désigne la *conformité* du rapport personnel de x au rapport institutionnel en position p [mis en valeur par l'auteur]. (Chevallard, 2003, p. 83)

Ainsi, le rapport personnel d'un individu à un objet est la somme des influences des institutions auxquelles il a été assujetti au cours de sa vie :

> Si o existe pour les sujets de I en position p, le rapport personnel de x à o, $R(x, o)$, tendra à ressembler au rapport institutionnel $R_I(p, o)$, à moins que x ne se révèle être, à cet égard, un *mauvais* sujet de I. D'une manière générale, nos rapports « personnels » sont ainsi le fruit de l'histoire de nos assujettissements institutionnels passés et présents. [mis en valeur par l'auteur]. (Chevallard, 2003, p. 84)

Après avoir présenté ces concepts fondamentaux de la TAD, nous en exposons les outils conceptuels que nous allons mettre en œuvre dans ce chapitre, en commençant par ceux en lien avec l'écologie des savoirs.

Approche écologique du didactique

Dans un cours nommé « Introduction à l'approche écologique du didactique », Artaud (1997) présente l'approche écologique comme un questionnement sur les conditions d'existence des objets de savoir dans les différents textes prescriptifs :

> La problématique écologique se présente d'emblée, comme *un moyen de questionner le réel*. Qu'est-ce qui *existe* et *pourquoi* ? Mais aussi, qu'est-ce qui n'*existe pas*, et *pourquoi* ? Et qu'est ce qui *pourrait* exister ? Sous quelles conditions ? Inversement, étant donné un ensemble de *conditions*, quels objets sont-ils poussés à vivre, ou au contraire sont-ils empêchés de vivre dans ces conditions? [Mis en valeur par l'auteure]. (1997, p. 101)

Ce questionnement doit permettre au chercheur de se détacher d'une illusion de transparence « qui conduit à penser que les choses sont comme elles sont parce qu'elles se conforment à quelque ordre naturel » (Artaud, 1997, p. 100), ainsi que d'être attentif aux dépendances entre les objets qu'il étudie.

L'écologie didactique s'inspire directement de l'écologie biologique et de la notion d'écosystème qui modélise les interactions entre les êtres vivants et leur milieu naturel. Artaud emprunte en particulier les concepts de *niche* et d'*habitat* à la biologie :

> La niche, c'est le métier, l'art de vivre d'un animal ou d'une plante. La niche d'une tarentule, c'est tout ce qu'elle doit faire pour se nourrir et élever ses petits. Pour y parvenir, elle doit s'adapter correctement à l'endroit où elle vit, et aux autres habitants de cet endroit. Le lieu physique où l'on vit est appelé, dans le jargon écologiste, l'habitat. L'habitat est l'adresse, l'emplacement précis ou les individus d'une espèce donnée vivent. Le sol forestier ou chassent les tarentules est leur habitat [...]. (Colinvaux, 1993, cité dans Artaud, 1997, p. 111)

En filant la métaphore biologique, Artaud (1997) adapte ces deux concepts et les définit en prenant pour exemple la notion mathématique de fonction génératrice :

> Il convient donc d'examiner les différents lieux où l'on trouve des fonctions génératrices et les objets avec lesquelles elles entrent en association, ce qu'on appellera les *habitats*. Puis de regarder, en chacun de leurs habitats, *la niche écologique* qu'elles occupent, c'est-à-dire, en quelque sorte, la fonction qui est la leur [Mis en valeur par l'auteure]. (1997, p. 111)

Nous retenons ici que l'habitat d'une notion est son lieu d'implantation, c'est-à-dire l'endroit où elle se trouve et l'ensemble des objets avec lesquels elle rentre en interaction, et que sa niche est sa place fonctionnelle, le rôle qu'elle joue (à quoi sert-elle ? quel problème permet-elle de résoudre ?). Nous allons utiliser ces deux concepts afin d'analyser la place de la programmation informatique dans les programmes scolaires.

Échelle des niveaux de codétermination didactique

Afin de structurer notre analyse de l'habitat de l'objet programmation informatique, nous utilisons l'*échelle des niveaux de codétermination didactique* (Chevallard, 2002). Cette échelle, que nous reproduisons dans la Figure 2-2, contient cinq niveaux allant de la discipline au sujet[10].

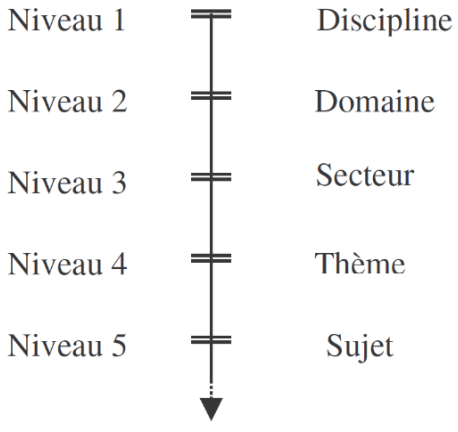


Figure 2-2 – Échelle des niveaux de codétermination didactique (Chevallard, 2002, p. 9)

Comme l'indique Chevallard (2002), chacun des échelons présente des conditions permettant la vie des objets, et des contraintes limitant leur existence : « chaque niveau

[10] Nous ne mobilisons pas les niveaux supérieurs de détermination que sont la pédagogie, l'école et la société.

concourt à déterminer l'écologie des organisations mathématiques et des organisations didactiques par les points d'appui qu'il offre et les contraintes qu'il impose » (2002, p. 10).

L'auteur propose un exemple d'utilisation de cette échelle en présentant une décomposition du programme de mathématiques de seconde de 2001 :

> Le programme de la classe de seconde, ainsi, apparaît scindé en trois domaines d'étude que les rédacteurs du programme ont appelés « chapitres » et intitulés respectivement « Statistique », « Calcul et fonctions », « Géométrie ». Le domaine de la « Statistique » est lui-même scindé en deux secteurs d'études, que l'on peut nommer « Résumé numérique d'une série statistique » et « Simulation et fluctuation d'échantillonnage ». Le premier de ces secteurs se divise à son tour en deux thèmes d'études, d'une part celui des « mesures de tendance centrale et de dispersion », d'autre part celui de la « distribution des fréquences d'une série statistique ». Le premier thème d'études, pléthorique, se laisse partager en sept sujets d'études : 1) calcul de la moyenne d'une série statistique ; 2) calcul de la médiane d'une série statistique ; [...]. (Chevallard, 2002, p. 2)

Nous allons, de manière similaire, utiliser cette échelle pour analyser la place de la programmation informatique dans les programmes d'enseignement.

Organisation praxéologique

Ensuite, nous allons utiliser le concept d'*organisation praxéologique* (ou praxéologie), introduit par Chevallard (1999), qui permet de modéliser les activités des individus dans une institution donnée : « *toute* activité humaine régulièrement accomplie peut être subsumée sous un modèle *unique*, que résume ici le mot de *praxéologie* [mis en valeur par l'auteur] » (Chevallard, 1999, p. 223). Une organisation praxéologique est décrite au moyen d'un quadruplet $[\mathrm{T}; \tau; \theta; \Theta]$ dans lequel T représente un type de tâches, τ une technique, θ une technologie et Θ une théorie. Précisons ces quatre notions.

Le *type de tâches* permet de décrire la tâche modélisée par une phrase de type « verbe d'action + complément », il répond à la question du *quoi :*

> À la racine de la notion de praxéologie se trouve les notions solidaires de *tâche*, t, et de *type de tâches*, T. Quand une tâche t relève d'un type de tâches T, on écrira quelquefois : $t \in T$. Dans la plupart des cas, une tâche (et le type de tâches *parent*) s'exprime par un *verbe* : *balayer* la pièce, *développer* l'expression littérale donnée, *diviser* un entier par un autre, *saluer* un voisin, *lire* un mode d'emploi, *monter* l'escalier, *intégrer* la fonction $x \longmapsto xlnx$ entre $x = 1$ et $x = 2$, etc. [mis en valeur par l'auteur]. (Chevallard, 1999, p. 224)

La formulation du type de tâches est un choix du chercheur (le modélisateur) et implique des décisions, en particulier à propos de la granularité à considérer (voir la suite avec la notion d'ingrédient de techniques) qui dépend de l'enjeu de la recherche.

Une *technique* est attachée à un type de tâches, elle permet de modéliser la manière de la traiter, et donc de répondre à la question du *comment* :

> Soit donc T un type de tâches *donné*. Une praxéologie relative à T précise (en principe) *une manière d'accomplir*, de *réaliser* les tâches $t \in T$: à une telle *manière de faire*, τ, on donne ici le nom de *technique* (du grec tekhnê, savoir-faire) [mis en valeur par l'auteur]. (Chevallard, 1999, p. 225)

En partant d'un des exemples de type de tâches proposé par Chevallard, T_S« Saluer un voisin », nous pouvons imaginer au moins deux techniques : τ_{S1} « Faire un grand signe de la main » et τ_{S2} « Serrer la main et bavarder quelques minutes ». Remarquons que les techniques sont légitimées localement par chaque institution : « en une institution I donnée, à propos d'un type de tâches T donné, il existe en général une seule technique, ou du moins un petit nombre de techniques institutionnellement reconnues » (Chevallard, 1999, p. 225). Dans l'exemple que nous développons, nous pouvons dire que les techniques de salutation sont propres à chaque institution « pays ». On peut penser aux différences culturelles allant de la « bise » à la française au « hug » américain en passant par l'inclinaison japonaise pour s'en convaincre.

La *technologie* permet de justifier l'existence des techniques et donc de répondre à la question du *pourquoi* :

> On entend par *technologie*, et on note généralement θ un *discours rationnel* – le *logos* – sur la technique – la *tekhnê* – τ, discours ayant pour objet premier de *justifier* « rationnellement » la technique τ, en nous assurant qu'elle permet bien d'accomplir les tâches du type T, c'est-à dire de réaliser ce qui est prétendu. Le style de rationalité mis en jeu varie bien entendu dans l'espace institutionnel, et, en une institution donnée, au fil de l'histoire de cette institution [mis en valeur par l'auteur]. (Chevallard, 1999, p. 226)

En poursuivant notre exemple sur la salutation d'un voisin, il est possible d'envisager les technologies qui justifient les techniques explicitées : θ_{S1} « Les restrictions sanitaires m'empêchent d'approcher mon voisin » et θ_{S2} « Mon voisin est une personne qui m'est familière ».

Enfin, la *théorie* permet, à un plus haut niveau, de légitimer la technologie, en explicitant les théories sous-jacentes :

> À son tour, le discours technologique contient des assertions, plus ou moins explicites, dont on peut demander raison. On passe alors à un niveau supérieur de justification-explication-production, celui de la *théorie*, Θ, laquelle reprend, par rapport à la technologie, le rôle que cette dernière tient par rapport à la technique [mis en valeur par l'auteur]. (Chevallard, 1999, p. 227)

Nous pouvons essayer d'exhiber les théories légitimant les technologies θ_{S1} et θ_{S2} en invoquant la théorie médicale Θ_{S1} « Les agents infectieux se transmettent par contact étroit avec un sujet infecté », ou la théorie du « savoir vivre » Θ_{S2} « La politesse veut que l'on serre la main à ses connaissances pour les saluer ».

Une organisation praxéologique est donc composée de deux blocs, l'un traitant du *savoir-faire* (type de tâches et technique) et l'autre du *savoir* (technologie et théorie) :

> Une telle praxéologie – ou *organisation praxéologique* – est donc constituée d'un bloc *pratico-technique*, $[T/\tau]$, et d'un bloc *technologico-théorique*, $[\theta/\Theta]$. Le bloc $[\theta/\Theta]$ est ordinairement identifié comme un *savoir*, alors que le bloc $[T/\tau]$ constitue un *savoir-faire*. Par métonymie, on désigne couramment comme étant un savoir la praxéologie $[T/\tau/\theta/\Theta]$ *toute entière* [mis en valeur par l'auteur]. (Chevallard, 1999, p. 228)

Les praxéologies ainsi définies permettent de modéliser à la fois les activités mathématiques proposées aux élèves, mais également la manière d'orchestrer ces activités en classe :

> Les types d'objets *o* envisagés seront eux-mêmes de deux sortes. Étant donné un thème d'étude mathématique, on considérera successivement: a) *la réalité mathématique* qui peut se construire dans une classe de mathématiques où l'on étudie le thème ; b) *la manière* dont peut se construire cette réalité mathématique, c'est-à-dire la manière dont peut s'y réaliser l'étude du thème. Le premier objet – « la réalité mathématique qui… » – n'est rien d'autre qu'une *praxéologie mathématique* ou *organisation mathématique* […]. Le second objet – « la manière dont… » – est ce qu'on nommera une *organisation didactique* [mis en valeur par l'auteur]. (Chevallard, 1999, p. 232)

Dans cette thèse, nous adaptons cette terminologie à notre sujet d'étude. Ainsi, nous qualifions d'*organisations informatiques* les praxéologies modélisant les tâches de programmation informatique proposées aux élèves, par exemple « Programmer le jeu Pong en Scratch » ou « Écrire une fonction approchant la racine carrée de deux en Python ». D'autre part, nous nommons *organisations didactiques* les praxéologies modélisant les aspects didactiques et pédagogiques de l'implémentation de l'enseignement dans les classes, par exemple « Présenter aux élèves la variable informatique à l'aide de la métaphore d'une boîte » ou « Faire travailler les élèves en groupe ».

T4TEL

Enfin, dans le but de pouvoir décrire en détails les techniques de nos organisations informatiques, nous allons recourir à des concepts du cadre théorique T4TEL développée par Chaachoua (2018) qui « s'inscrit dans la Théorie Anthropologique du Didactique et plus spécifiquement dans l'approche praxéologique [et] représente une formalisation et une extension du modèle praxéologique. » (2018, p. 5).

Selon Chaachoua (2018), les techniques sont divisibles en tâches intermédiaires nommés *ingrédients de techniques* qui sont elles-mêmes des types de tâches : « Une technique est décrite par un ensemble de types de tâches appelés ingrédients de la technique. » (2018, p. 13). En reprenant la technique τ_{S1} « Faire un grand signe de la main », il ainsi est possible de la diviser en ingrédients de techniques : $T_{S1.1}$ « Sortir sa main de sa poche », $T_{S1.2}$ « Lever la main bien haut » et $T_{S1.3}$ « Dire bonjour ». Chacun de ces ingrédients est, à son tour, un type de tâches dont on peut détailler les techniques. En réitérant cette décomposition, un *type de tâches père* peut être représenté par un *arbre de types de tâches*. Pour que le processus de décomposition puisse se terminer, Chaachoua (2018) définit le *type de tâche élémentaire* en précisant : « un type de tâches est élémentaire si l'institution ou le chercheur du domaine considère qu'il n'est pas nécessaire d'expliciter la ou les techniques pour ce type de tâches » (2018, p. 14). Ainsi, les éléments atomiques que constituent les types de tâches élémentaires servent de feuilles aux arbres de types de tâches. Nous qualifions les ingrédients de techniques qui ne sont pas élémentaires d'*ingrédients intermédiaires*.

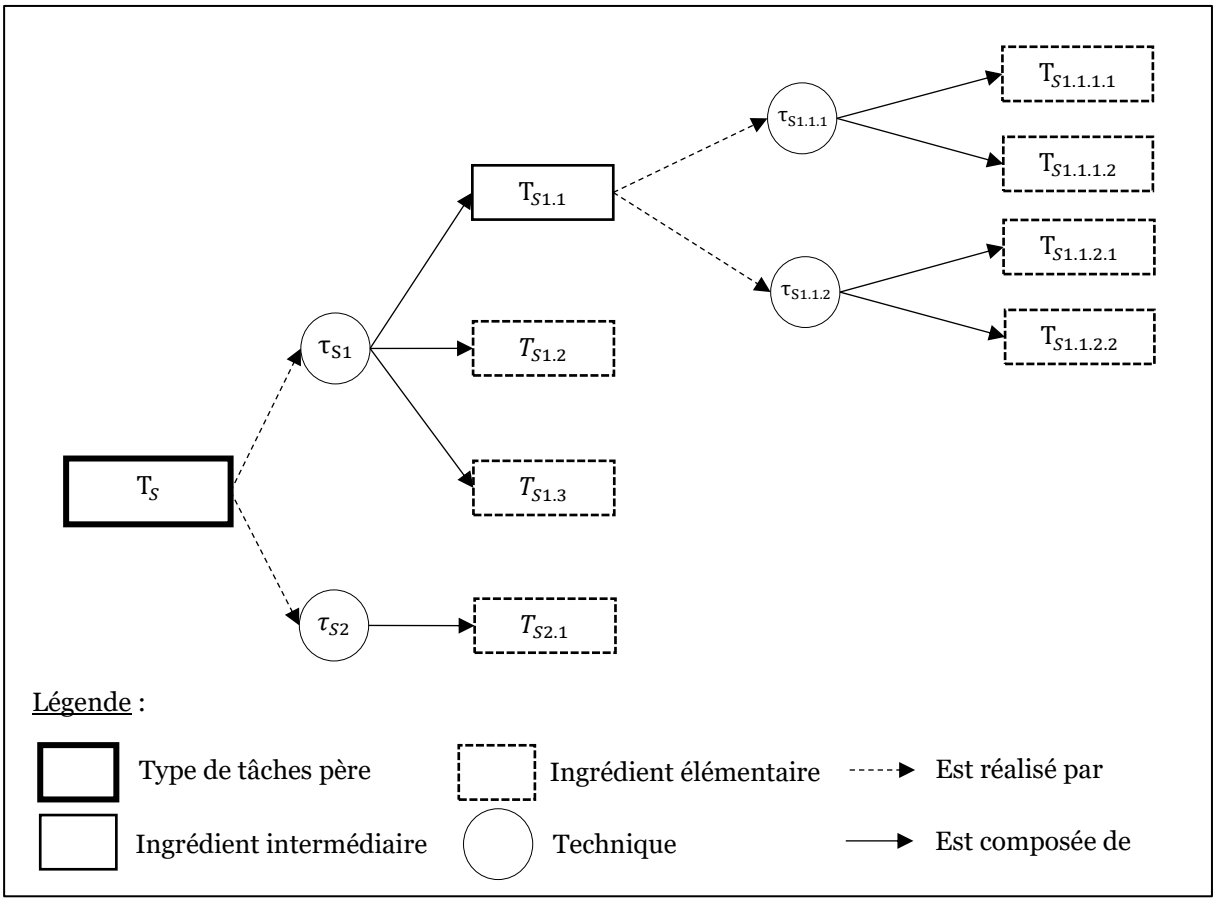


Figure 2-3 – Exemple d'arbre de type de tâches correspondant au type de tâches père de la salutation d'un voisin

En guise d'illustration, nous avons établi dans la Figure 2-3 l'arbre des types de tâches correspondant au type de tâche père T_S « Saluer son voisin ». Précisons que, dans les lignes qui précèdent, nous n'avons pas explicité toutes les parties de l'arbre. Il est cependant possible d'imaginer, par exemple, qu'il existe deux techniques pour sortir la main de sa poche déclinés chacune en deux ingrédients.

Les outils fournis par la TAD vont nous permettre d'étudier la place de la programmation informatique dans les programmes d'enseignement en modélisant les tâches de programmation recommandées et les prescriptions touchant à la mise en œuvre didactique et pédagogique des enseignements. Cependant, nous avons besoin d'un cadre théorique nous permettant de catégoriser ces prescriptions. Par conséquent, nous présentons dans la sous-section suivante les indicateurs proposés par Reuter pour analyser le fonctionnement pédagogico-didactique des disciplines scolaires.

2.2.2 Le fonctionnement pédagogico-didactique des disciplines

Nous utilisons dans notre analyse les différents *indicateurs* déclinés par Reuter (2004) pour analyser le fonctionnement pédagogico-didactique des disciplines afin de caractériser les organisations didactiques prescrites dans les textes officiels. Reuter (2004) distingue les contenus, les dispositifs et activités d'enseignement, et les aspects matériels : « sous l'expression [...] de fonctionnement pédagogico-didactique, je regroupe des indicateurs portant sur le fonctionnement "interne" (autonome) de la discipline : contenus, dispositifs et activités d'enseignement, d'apprentissages et

d'évaluation, mise en œuvre matérielle » (2004, p. 4). Nous donnons ci-dessous davantage de détails sur la manière dont Reuter conçoit ces indicateurs et la façon dont nous les avons adaptés.

L'indicateur des contenus fait référence au repérage et à l'organisation des savoirs dans les programmes :

> Concernant les contenus, je m'arrêterai d'abord sur leur *étendue* - vaste en l'occurrence - et sur leurs *contours*, flous et très instables [...]. Du point de vue de leur *mode d'organisation*, [...] le français se caractérise ainsi par une *organisation en sous-domaines* [...] cette organisation en sous domaines s'articule à une *hiérarchisation* instable [...] elle s'articule aussi à la redoutable question de la *progression* [mis en valeur par l'auteur]. (Reuter, 2004, p. 4-5)

Comme nous allons mobiliser les concepts de la TAD présentés précédemment pour analyser l'organisation, la hiérarchisation et la progression des savoirs dans les programmes, nous n'utiliserons pas l'indicateur des contenus avec cette signification. Nous l'adaptons de la manière suivante : l'indicateur *contenu* nous permettra, de manière plus générale, de qualifier les organisations didactiques ayant trait aux concepts de la programmation informatique.

Reuter (2004) propose un deuxième indicateur, celui des dispositifs et des activités, qu'il définit de la sorte :

> Concernant les *dispositifs et activités d'enseignement, d'apprentissage, et d'évaluation*, [...] il existe *a priori* une diversité potentielle des dispositifs d'enseignement et des activités (des plus « ouvertes » aux plus « fermées ») ; mais cette diversité *potentielle* se réduit souvent, dans les pratiques courantes, au profit de genres « emblématiques » (exercices de grammaire, récitations, questions de lecture, rédactions, dissertations, commentaires de textes.), et tend à exclure certains exercices caractéristiques d'autres disciplines (expérimentations, résolution de problèmes circonscrits, recherches.) [mis en valeur par l'auteur]. (2004, p. 5-6)

En dépit du nom « dispositifs et activités » utilisé par Reuter, cet indicateur est surtout en lien avec la forme des tâches proposées aux élèves : démarche, type d'exercices, etc. Nous proposons donc de le séparer en deux entités et de les adapter à notre sujet d'étude qui est la programmation informatique. Ainsi, nous considérons l'indicateur *activité*[11] comme étant en rapport avec la *démarche* (projet, démarche d'investigation, exposé, etc.), le *type d'activité* (écrire, modifier, compléter, interpréter un programme) et le *mode de programmation*, c'est-à-dire la manière dont l'élève doit procéder pour construire son programme (boucles d'essais-erreurs dans un langage de programmation, organigramme puis traduction dans un langage de programmation, etc.). Nous établissons donc un second indicateur que nous nommons *dispositif*, en rapport avec les *lieux d'enseignement* (salle de classe, salle informatique, laboratoire, etc.), l'*organisation de l'espace* (classe entière, îlots, etc.) et les *interactions entre les individus* (coopération, autonomie, rôle du professeur, etc.).

Troisièmement, Reuter (2004) propose l'indicateur de la mise en œuvre matérielle : « je m'intéresse aux dimensions *matérielles* de la mise en œuvre de la discipline : instruments pour inscrire ou effacer, supports (classeurs, cahiers.),

[11] Pour cette indicateur, nous entendons le terme « activité » comme « ce que font effectivement des apprenants en réponse à une tâche donnée » (Tchounikine, 2009, p. 30), là où pour Reuters il désignait plutôt un énoncé ou un problème proposé aux élèves.

manuels, photocopies [...][mis en valeur par l'auteur] » (2004, p. 6). Nous reprenons cette idée en l'élargissant à la notion d'artefact, défini comme un objet technique ou symbolique ayant subi une transformation d'origine humaine et destiné à une activité finalisée (Rabardel, 1995). Nous renommons donc l'indicateur en *artefact*. Il permet de catégoriser les indications concernant les objets présents dans l'environnement. Nous discernons les *artefacts-supports* qui permettent de formaliser des algorithmes et de fabriquer des programmes (ordinateur, tablette, éditeur de programme, langages de programmation, formalismes d'expression d'algorithmes, etc.) et les *artefacts-cibles* qui désignent les artefacts contrôlés par les programmes (robots, objets domotiques, etc.).

Au final, en nous appuyant sur l'axe d'analyse du fonctionnement pédagogico-didactique introduit par Reuter que nous avons adapté à notre sujet d'étude, nous obtenons quatre indicateurs (contenus, activités, dispositifs et artefacts) que nous avons, pour trois d'entre eux, déclinés en sous-indicateurs (démarche, type d'activité, mode de programmation, lieu, organisation de l'espace, interaction, artefact-support, artefact-cible). Ces catégories vont nous permettre de classer les organisations didactiques modélisant les prescriptions relatives à la mise en œuvre des enseignements.

2.2.3 Synthèse de la section

Cadre théorique

Nous avons présenté dans cette section les outils théoriques que nous allons utiliser dans ce chapitre pour analyser les textes officiels issus du ministère de l'Éducation nationale. Un premier ensemble de concepts s'inscrit dans la **Théorie anthropologique du didactique** (Chevallard, 1992) :

- l'**approche écologique** (Artaud, 1997) va nous permettre d'étudier la place de la programmation informatique dans les programmes d'enseignement au moyen du concept d'**habitat** qui désigne le lieu de vie et les interactions d'un objet de savoir, et du concept de **niche** qui correspond au rôle de cet objet ;
- l'**échelle des niveaux de codétermination didactique** (Chevallard, 2002), et ses cinq niveaux (**discipline**, **domaine**, **secteur**, **thème** et **sujet**) offrent un cadre d'analyse de l'habitat ;
- le modèle de l'**organisation praxéologique** (Chevallard, 1999) et ses quatre composantes (**type de tâches**, **technique**, **technologie** et **théorie**) étendu par les **ingrédients de techniques** et les **types de tâches élémentaires** de T4TEL (Chaachoua, 2018) nous fournit un moyen de modéliser les tâches de programmation recommandées sous la forme d'**organisations informatiques** et les prescriptions relatives à la mise en œuvre des enseignements en **organisations didactiques**.

Afin de catégoriser les organisations didactiques modélisées, nous avons adapté les **indicateurs** de Reuter (2004) concernant le **fonctionnement pédagogico-didactique** des disciplines, nous retrouvons :

- les **contenus** : relatif aux concepts enseignés ;

- l'**activité** : nous différencions la **démarche** (projet, démarche d'investigation, etc.), le **type d'activité** (écrire, modifier, compléter, etc.) et le **mode de programmation** (essais-erreurs, formalisation d'un algorithme puis traduction, etc.) ;
- le **dispositif** : en lien avec les **lieux** (salle de classe, salle informatique, etc.), l'**organisation de l'espace** (groupe classe, îlots, etc.) et les **rapports entre individus** (coopération, autonomie, etc.) ;
- les **artefacts** : concernent les artefacts matériels, logiciels et symboliques, nous discernons les **artefacts-supports** qui permettent de formaliser des algorithmes et de fabriquer des programmes (ordinateur, éditeur de programme, langage, etc.) des **artefacts-cibles** qui sont ceux contrôlés par les programmes (robots, etc.).

Nous venons d'exposer les différents outils théoriques que nous allons mobiliser, nous présentons dans la section suivante les questions de recherche et la méthodologie que nous allons adopter dans ce chapitre.

2.3 Questions de recherche et méthodologie

Nous détaillons dans cette section les différents questionnements qui dirigent les recherches présentées dans ce chapitre ainsi que la méthodologie retenue.

2.3.1 Questions de recherche

Nous étudions dans ce chapitre la question de recherche Q2 : Quelle est la place de la programmation informatique dans les programmes scolaires français à la transition collège-lycée ? Quelles sont les modalités d'enseignement prescrites ? Quelles sont les conséquences pour les élèves qui passent du collège au lycée ?

Cette question se décompose en plusieurs sous-questions que nous énonçons à la suite :

– Q2.1 : Dans quelles disciplines est enseignée la programmation informatique à la transition collège-lycée ? Quelles légitimations sont avancées dans les textes officiels ?

– Q2.2 : Quels sont les habitats de la programmation informatique dans les programmes du cycle 4 et de la seconde ? Autrement dit, quels sont les différents lieux où elle se trouve et avec quels savoirs est-elle en interaction ?

– Q2.3 : Quels sont les concepts fondamentaux de la programmation informatique mis en jeu dans les programmes de chaque discipline ? Quelle transposition des savoirs savants ?

– Q2.4 : Quelles sont les niches des concepts de la programmation dans les tâches recommandées par les ressources d'accompagnement ?

– Q2.5 : Quelles sont les indications des textes officiels concernant les modalités d'enseignement de la programmation informatique ? Qu'en ressort-il en termes de contenu, d'activité, de dispositif et d'artefact ?

– Q2.6 : Quelles discontinuités trouve-t-on dans les indications officielles entre le collège et le lycée ? Quelles difficultés sont-elles susceptibles d'engendrer pour les élèves ?

Nous exposons dans la sous-section suivante le détail de notre méthodologie pour chacune des six orientations de recherche présentées ci-dessus.

2.3.2 Méthodologie : analyses curriculaires

La méthodologie mise en œuvre dans ce chapitre repose sur l'étude des textes officiels émanant du ministère de l'Éducation nationale. Nous présentons d'abord les aspects communs à toutes les questions de recherche avant de préciser la méthodologie pour chaque question.

Aspects généraux

Nous prenons en considération deux types de textes émis par le ministère de l'Éducation nationale. D'abord, les *programmes d'enseignement* publiés dans les bulletins officiels qui sont des *prescriptions* nationales à l'attention des enseignants. Comme le précise le site du ministère : « [ils] définissent les connaissances essentielles et les méthodes qui doivent être acquises au cours du cycle par les élèves. Ils constituent le cadre national au sein duquel les enseignants organisent leurs enseignements » (MEN, 2023). En complément, nous portons notre attention sur un ensemble de ressources accompagnant les programmes. Ces ressources sont nommées différemment selon les disciplines : « guide d'accompagnement », « document ressource », « ressource », « guide pédagogique et didactique », ou « outil d'aide ». Dans cette thèse, nous suivons la terminologie proposée par Gueudet et Trouche (2008) qui désignent par « ressource » les éléments qui peuvent nourrir l'activité professionnelle du professeur. Nous parlerons donc de *ressources d'accompagnement*. Comme expliqué en préambule d'une de ces ressources, elles ont pour objectif d'aider les enseignants dans la mise en œuvre de leurs enseignements et ont le statut de *recommandations* :

> Cette ressource d'accompagnement ne revêt aucun caractère prescriptif. Elle a pour objectif d'aider les enseignants à concevoir et à mettre en œuvre l'enseignement de la technologie au cycle 4, en leur apportant un accompagnement scientifique, didactique et pédagogique. Ces apports se font dans le respect de la liberté pédagogique des enseignants. Il revient à chaque professeur, dans le cadre d'une progression pédagogique conçue en équipe, de s'approprier le programme, d'organiser le travail de ses élèves et de choisir les méthodes qui lui semblent les plus adaptées en fonction des acquis initiaux, des objectifs à atteindre et des progrès des élèves. (MEN, 2016a)

L'enseignant garde donc sa liberté pédagogique dans le « travail documentaire » (Gueudet & Trouche, 2010) l'amenant à combiner des ressources afin d'organiser ses séances d'enseignement.

Pour désigner de manière générale ces deux types de textes (programmes d'enseignement et ressources d'accompagnement), nous parlerons de *textes officiels*. Dans le même ordre d'idées, nous emploierons le terme *indications* pour qualifier de manière générique leurs contenus (les prescriptions et les recommandations).

Nos analyses vont donc porter sur un corpus de textes constitué des programmes d'enseignement et des ressources d'accompagnement relatifs aux quatre disciplines

contribuant à l'enseignement de la programmation informatique à la transition collège-lycée (voir section 2.4). Nous listons ces textes dans Tableau 2-1.

Disciplines	Codes	Textes officiels
Mathématiques cycle 4	PM4	Programme d'enseignement du cycle 4 - Mathématiques (MEN, 2020b, p. 126-136)
	R1M4	Ressources d'accompagnement - Algorithmique et programmation (MEN, 2016c)
Technologie cycle 4	PT4	Programme d'enseignement du cycle 4 - Technologie (MEN, 2020b, p. 116-125)
	R1T4	Ressources d'accompagnement - Guide pédagogique et didactique (MEN, 2016a)
	R2T4	Ressources d'accompagnement - Outil d'aide à l'élaboration de progression pédagogique (MEN, 2016b)
Mathématiques Seconde	PMS	Programme de mathématiques de seconde générale et technologique (MEN, 2019b)
	R1MS	Ressources d'accompagnement - Algorithmique et programmation (MEN, 2019a)
	R2MS	Ressources pour le lycée général : algorithmique et programmation (MEN, 2017b)
SNT seconde	PSS	Programme de sciences numériques et technologie de seconde générale et technologique (MEN, 2019e)
	R1SS	Ressources d'accompagnement par thématiques pour la mise en œuvre du programme (MEN, 2019h)

Tableau 2-1 – Ensemble des programmes d'enseignement et des ressources d'accompagnement analysés dans ce chapitre

Pour les mathématiques, nous disposons, au cycle 4 comme en seconde, du programme d'enseignement (PM4 et PMS) et d'une ressource d'accompagnement spécifique à l'algorithmique et la programmation (R1M4 et R1MS). En seconde, nous prenons également en considération la ressource d'accompagnement antérieure R2MS. En effet, R1MS précise que : « ce document peut être utilisé en liaison avec la ressource précédente sur l'algorithmique et la programmation au lycée, publiée en juin 2017, qui garde toute sa pertinence pour l'étude de Python » (MEN, 2019a, p. 1). Ces ressources d'accompagnement comprennent quelques lignes directrices didactiques et pédagogiques ainsi que des exemples d'activités. Concernant la technologie, un « guide pédagogique et didactique » (R1T4) ainsi qu'un « outil d'aide à l'élaboration de progression pédagogique » (R2T4) complètent le programme d'enseignement (PT4). La première ressource contient un ensemble de conseils didactiques et pédagogiques de mise en œuvre des enseignements et la deuxième des exemples d'activités. Enfin, en SNT, nous disposons, en plus du programme d'enseignement (PSS), d'une ressource présentant un large éventail d'activités balayant toutes les thématiques du programme (R1SS).

Bien que s'appuyant sur le même corpus de textes, notre méthodologie varie en fonction des questions de recherche. Nous présentons dans la Figure 2-4 un schéma résumant ces différences.

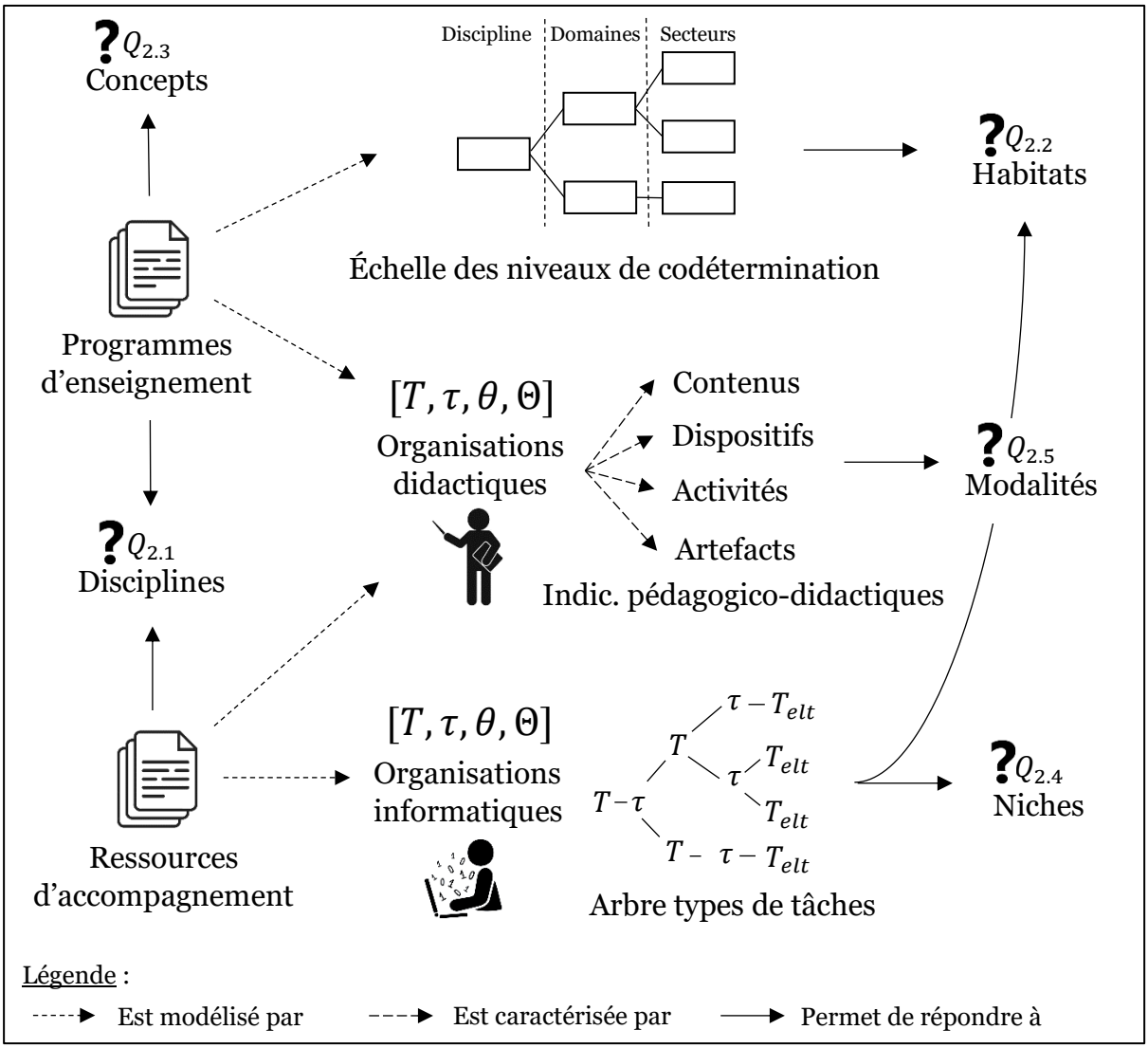


Figure 2-4 – Schéma récapitulatif de la méthodologie employée pour répondre aux différentes questions de recherche de ce chapitre

Nous détaillons ci-après les particularités méthodologiques propres à chaque question de recherche.

Disciplines (Q2.1)

Pour la question de recherche Q2.1 portant sur les disciplines hôtes de la programmation informatique, nous parcourons directement les programmes d'enseignement et les ressources d'accompagnement à la recherche d'éléments en lien avec la programmation informatique et de discours légitimant leur présence.

Habitats (Q2.2)

La méthodologie adoptée pour la question de recherche Q2.2 ayant trait à l'habitat de la programmation informatique s'appuie sur la structuration des programmes selon l'échelle des niveaux de codétermination didactique. Cela permet de déterminer les lieux d'habitat de la programmation informatique dans les différentes disciplines. Nous utilisons également la modélisation des activités recommandées sous la forme d'organisations informatiques afin d'évaluer les interactions de la programmation informatique avec les autres domaines du programme.

Concepts (Q2.3)

Concernant la question Q2.3 centrée sur les concepts fondamentaux prescrits, nous relevons, directement dans les programmes d'enseignement des différentes disciplines, les concepts listés dans les parties qui délimitent les concepts à enseigner (généralement sous les titres « contenus » ou « connaissances »).

Niches (Q2.4)

La question Q2.4 porte sur les niches des concepts de la programmation. Notre méthodologie consiste ici à disséquer finement les différentes activités proposées dans les ressources d'accompagnement (R1M4, R2T4, R1MS, R1SS). Plus précisément, nous comptons les occurrences et analysons l'utilité des différents concepts implémentés. Pour ce faire, nous suivons les étapes décrites ci-dessous.

D'abord, dans l'objectif d'avoir une vision globale et unifiée des activités, nous les modélisons sous la forme d'organisations informatiques (voir section 2.2.1). Pour ce faire, nous utilisons des tableaux à trois colonnes contenant les éléments relatifs aux types de tâches, aux techniques et aux technologies (voir Annexe A). Pour chaque ressource d'accompagnement, nous reportons d'abord, dans la première colonne, les extraits décrivant les activités. Nous caractérisons ensuite les différents passages à l'aide d'un code couleur. Les éléments surlignés en jaune correspondent aux types de tâches (quoi ?), nous les résumons en une phrase faisant office de titre pour la cellule. Les parties surlignées en vert et en bleu coïncident respectivement avec les techniques (comment ?) et les technologies (pourquoi ?), nous les synthétisons dans les deuxième et troisième colonnes. Nous reproduisons également, lorsqu'ils sont disponibles, les scripts Scratch ou les programmes Python qui explicitent les techniques associées aux types de tâches dans la deuxième colonne. Ne disposant pas d'éléments relatifs aux théories dans les ressources, nous n'avons pas prévu de colonne dédiée.

La deuxième étape consiste en un découpage récursif des techniques en ingrédients de techniques (étant eux-mêmes des types de tâches). Le résultat de cette subdivision est un arbre de types de tâches dont les éléments terminaux sont des ingrédients élémentaires. Ces ingrédients de base mettent directement en œuvre les concepts de la programmation. Nous décrivons à la suite ce découpage spécifique à chaque ressource d'accompagnement. Précisons que les arbres de types de tâches n'apparaissent pas sous leur représentation « arborée » mais sont intégrés dans les tableaux.

En mathématiques, au cycle 4, la ressource d'accompagnement R1M4 contient six propositions de projets ludiques en lien avec la programmation. Les activités sont décrites en détail à travers l'explicitation des différentes phases de la résolution. Nous ne disposons pas des supports pédagogiques formalisés, cependant, la plupart des projets sont accompagnés de « corrections » sous la forme de scripts Scratch.

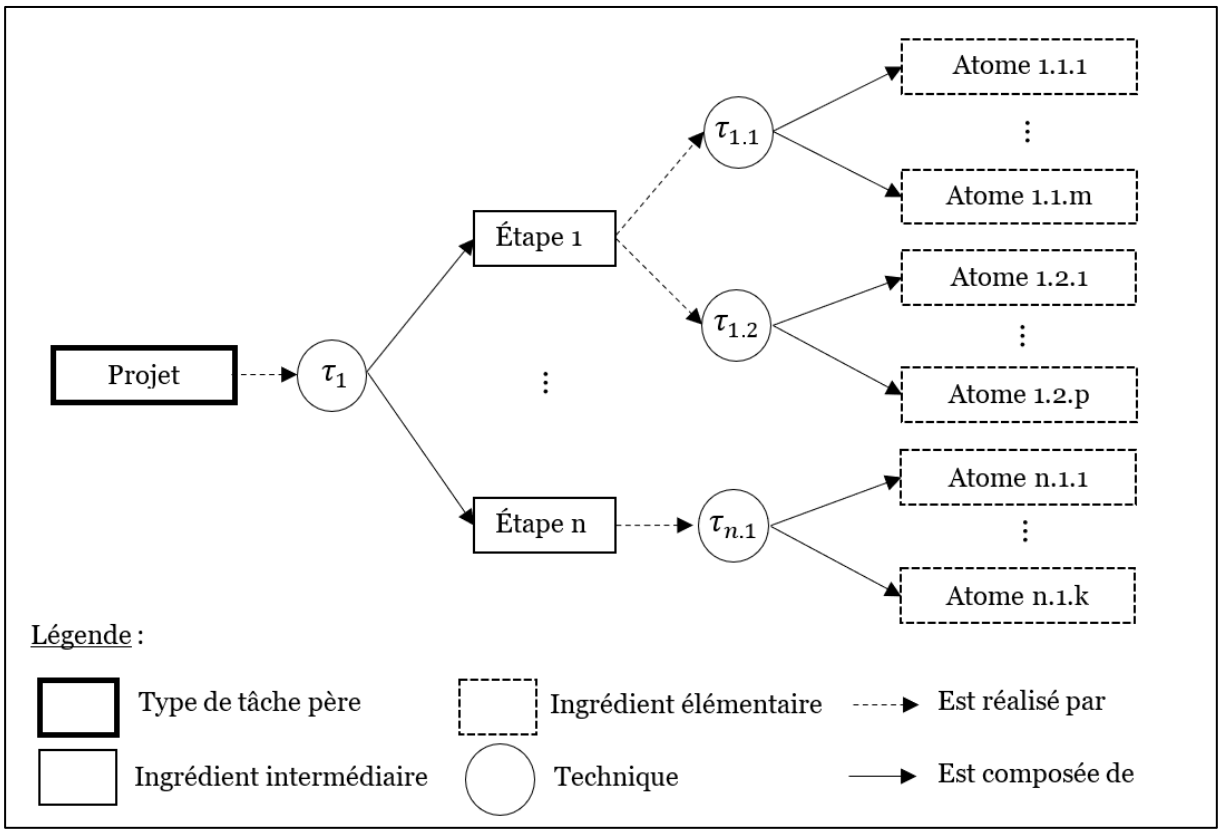


Figure 2-5– Découpage en ingrédients des types de tâches en mathématiques au cycle 4

Comme illustré dans la Figure 2-5, nous avons modélisé chaque projet en un découpage en étapes, qui sont des ingrédients intermédiaires. Ces étapes, qui peuvent être accomplies par une ou deux techniques, sont, à leur tour, subdivisées en ingrédients élémentaires. Ces ingrédients ont une granularité équivalente à celle des instructions Scratch. Nous les nommons *atomes,* ce sont des types de tâches atomiques, directement liées aux concepts de la programmation. Nous pouvons citer par exemple : « modifier la valeur d'une variable », « utiliser une structure conditionnelle à deux branches » ou « définir une fonction sans paramètre avec retour ». Précisons que le premier projet (« déplacer un lutin selon un parcours prédéfini »), plus rudimentaire, n'a pas été découpé en étapes. Nous avons, au final, pour cette ressource d'accompagnement, analysé un corpus de 28 scripts Scratch, issus de 6 organisations informatiques (voir Annexe A.1.2.2).

En technologie au cycle 4, la ressource d'accompagnement R2T4 propose une soixantaine de séquences d'enseignements dont dix mettent directement en jeu la programmation informatique. La ressource présente chaque séquence à travers : ses objectifs, ses questions directrices, ses activités et son bilan. Nous ne disposons, ni de support pédagogique précis, ni de corrigé. Cependant, la liste des notions algorithmiques travaillées est établie dans la partie bilan de chaque séquence.

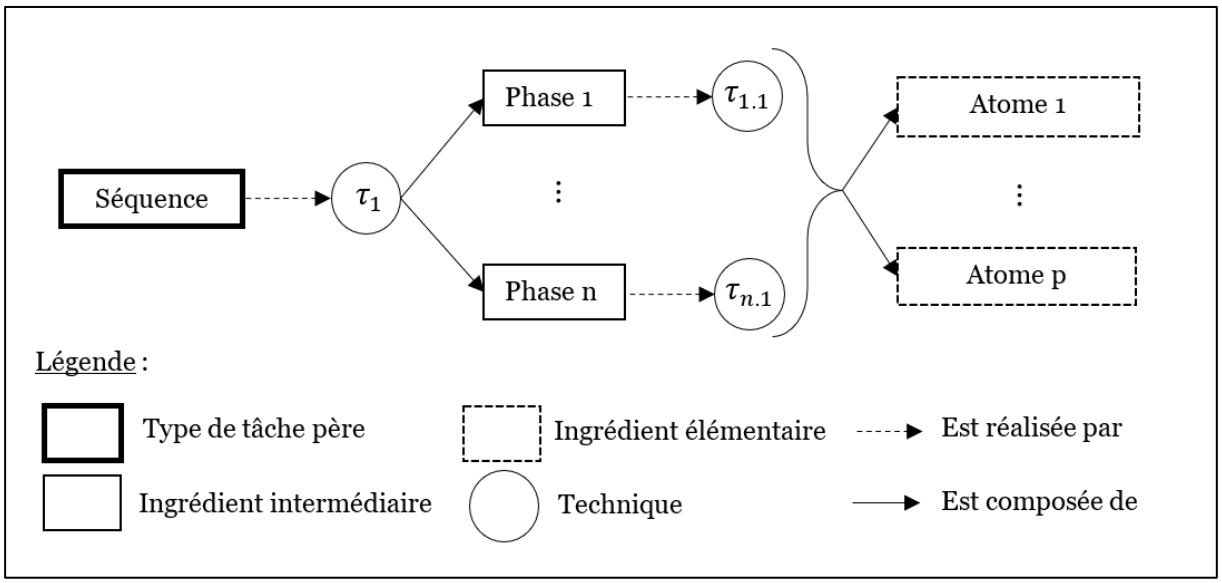


Figure 2-6 - Découpage en ingrédients des types de tâches en technologie au cycle 4

La Figure 2-6 précise le découpage que nous avons retenu. Chaque séquence est partagée en phases, qui sont des ingrédients intermédiaires. Nous ne disposons pas du détail des techniques associées à ces phases. Nous avons, néanmoins, à disposition une liste d'ingrédients élémentaires que sont les concepts de programmation mis en jeu globalement dans la séquence. Ces concepts n'étant pas contextualisés dans des techniques, ils sont, dès lors, moins précis et plus génériques. Donnons en quelques exemples : « utiliser une boucle », « utiliser une variable » ou « utiliser un connecteur logique ». Nous avons donc extrait de cette ressource dix organisations informatiques (Annexe A.2.3.1).

En mathématiques, au niveau seconde, la ressource d'accompagnement R1MS fournit treize exemples d'activités de programmation pour la classe, dont quatre sont destinées au niveau seconde. Ces activités se présentent sous la forme de fiches commentées. Ces fiches contiennent des suggestions de questions que les enseignants peuvent poser aux élèves. La plupart d'entre-elles sont corrigées par la mise à disposition de programmes exprimés en Python.

Comme l'illustre la Figure 2-7, nous avons décomposé les activités en deux niveaux d'ingrédients intermédiaires : les parties puis les questions. Ensuite, pour chaque question, nous disposons d'une ou plusieurs techniques que nous avons fractionnées en ingrédients élémentaires. Ces ingrédients ont la même granularité que pour les mathématiques au cycle 4. Ce sont des concepts élémentaires de programmation mis en œuvre dans des programmes Python. Nous avons donc, pour cette ressource, étudié quinze programmes Python émanant de quatre organisations informatiques (Annexe A.3.2.2).

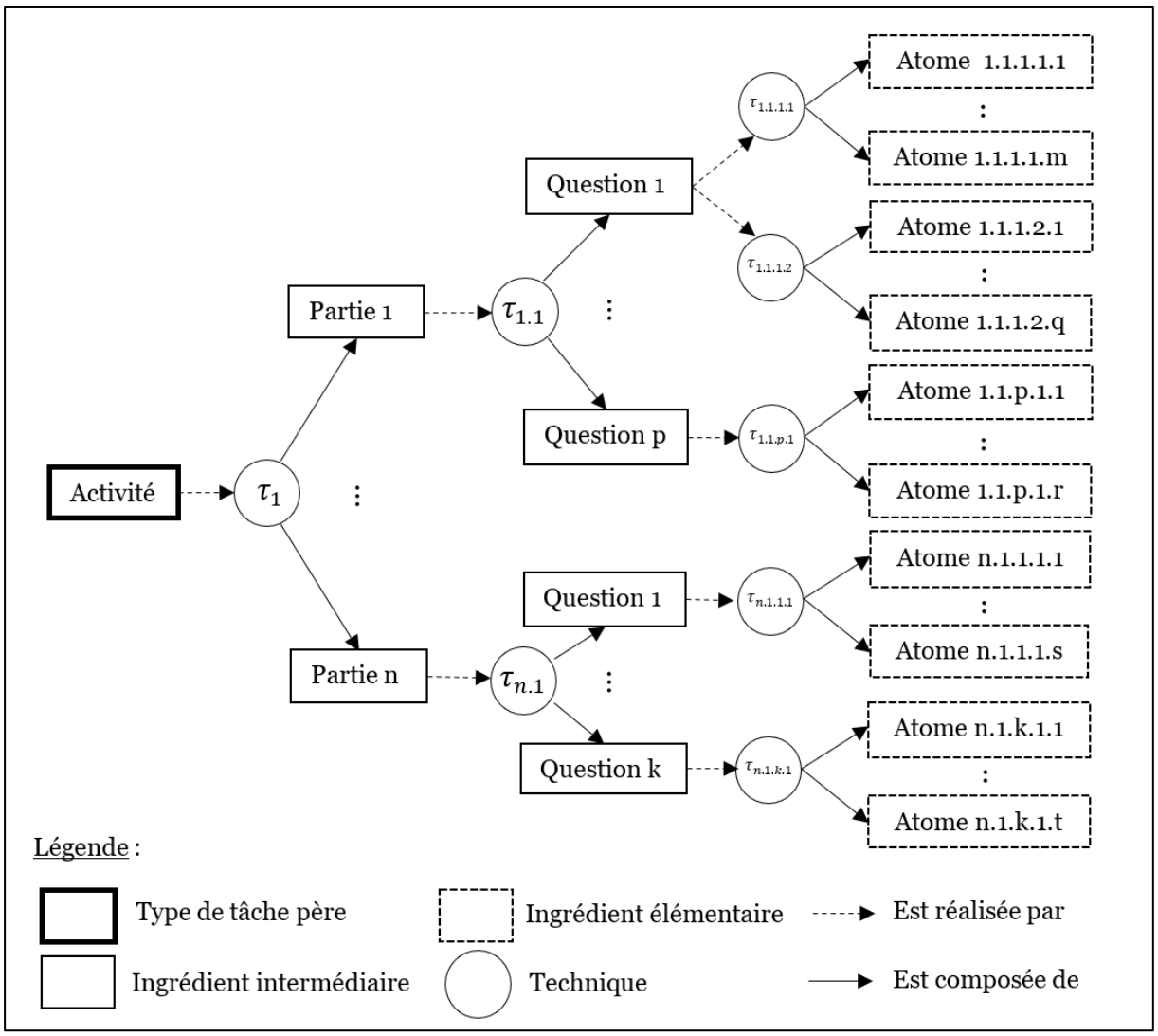


Figure 2-7 - Découpage en ingrédients des types de tâches en mathématiques et en SNT en seconde

Enfin, pour les SNT, le site Éduscol héberge un ensemble de 31 activités illustrant les sept domaines du programmes R1SS. Parmi elles, quatre mettent en œuvre la programmation informatique. Chaque activité est exposée dans une fiche explicative comprenant : des exercices, des commentaires et des éléments de correction prenant la forme de code Scratch ou Python. Nous utilisons le même découpage que pour les mathématiques (voir Figure 2-7) à savoir : activités, parties, questions puis atomes. Nous avons ainsi, à travers la modélisation de ces quatre organisations informatiques, considéré 31 programmes Python et 5 scripts Scratch (Annexe A.4.2.1).

Lors d'une troisième étape, nous classons les atomes issus des 79 scripts analysés dans six ensembles de concepts que nous avons établis. Ces différents ensembles sont issus du regroupement des atomes par thèmes, nous distinguons : variables et types de données, structures de contrôle, programmation parallèle et évènementielle, opérations logiques et numériques et, bibliothèques. Par la suite, nous dénombrons les occurrences de chaque atome pour chaque discipline puis calculons leurs proportions respectives. Ces éléments quantitatifs sont rassemblés dans des tableaux pour chaque ensemble de concepts dans la section 2.7. Précisons que ces ensembles de concepts vont au-delà des concepts fondamentaux de la programmation du savoir savant (voir

section 1.5.3) et, plus étonnamment, de ceux prescrits dans les programmes d'enseignement (voir section 2.6).

C'est pourquoi, lors d'une dernière étape, nous caractérisons ces atomes au regard des prescriptions concernant les différentes disciplines. Nous rencontrons ainsi trois cas de figure que nous détaillons ci-dessous.

- Au programme : l'atome met en œuvre un concept explicitement prescrit dans le programme d'enseignement de la discipline. C'est par exemple le cas de l'ingrédient « modifier une variable » pour les mathématiques en cycle 4. Cette situation est signalée par un fond de cellule de couleur verte dans les tableaux.
- Hors programme : l'atome implémente un concept qui n'est pas mentionné dans le programme d'enseignement de la discipline, cette notion faisant partie du programme d'une année supérieure. Prenons l'exemple de l'atome « accéder à un l'élément d'un n-uplet » en SNT. La notion de n-uplet n'est pas contenue dans le programme de cette discipline, elle fait partie des contenus du programme de NSI en première. Cette situation est signalée par un fond de cellule rouge.
- Implicite : l'atome emploie un concept qui n'apparait pas explicitement dans le programme d'enseignement de la discipline, ce concept étant, dans un autre contexte, au programme d'une année inférieure. Néanmoins, sa mise en œuvre dans un contexte de programmation ne va pas de soi et nécessite un apprentissage. Exposons l'exemple de l'ingrédient « tester une égalité sur des nombres » en seconde. Le concept d'égalité est en principe déjà acquis au cycle 4, cependant, sa mise en œuvre en Python peut entraîner des confusions avec l'affectation de variable (Delmas-Rigoutsos, 2020). Cette situation est indiquée par un fond de cellule jaune.

De plus, nous grisons les cases lorsque l'atome n'est pas techniquement réalisable dans le langage de programmation couramment utilisé dans une discipline, ou si nous n'avons pas accès à ce niveau de granularité comme c'est le cas pour la technologie.

Finalement, nos analyses s'appuient sur ces tableaux de comptage pour répondre à la partie quantitative de la question de recherche (quelle utilisation des concepts ?) et sur les contextes d'implémentation des atomes pour la partie qualitative (quel rôle des concepts ?).

Modalités (Q2.5)

Pour la question de recherche Q2.5 en lien avec les modalités d'enseignement, nous procédons en suivant les étapes que nous décrivons ci-dessous.

Dans un premier temps, nous relevons systématiquement les extraits traitant des modalités de mise en œuvre des enseignements de la programmation informatique dans les programmes d'enseignement et les ressources d'accompagnement pour chaque discipline. Nous regroupons à cette occasion les extraits traitant d'un même sujet.

Lors d'une deuxième étape, nous modélisons ces éléments sous la forme d'organisations didactiques (voir section 2.2.1). De la même manière que pour la question de recherche précédente sur les niches, ces praxéologies sont consignées, pour chaque texte officiel, dans un tableau à trois colonnes (voir Annexe A). Ces colonnes concernent respectivement : les types de tâches (quoi ?), les techniques

(comment ?) et les technologies (pourquoi ?). Nous ne renseignons pas les aspect liés à la théorie faute d'éléments de cette nature dans les textes. Plus précisément, nous citons dans la première colonne les extraits des différents textes. Un code couleur dans le surlignage permet de catégoriser le type de discours. Le type de tâche (soulignage jaune) est ensuite reformulé puis indiqué en gras dans cette même première colonne. Nous synthétisons puis classons ensuite les éléments surlignés relatifs aux techniques (surlignage vert) et aux technologies (surlignage bleu) respectivement dans les deuxième et troisième colonnes.

Dans un troisième temps, nous caractérisons chaque type de tâches à l'aide des indicateurs adaptés de Reuter concernant la fonctionnement pédagogico-didactique des disciplines (voir section 2.2.2). Pour rappel, nous distinguons quatre indicateurs et huit sous-indicateurs, selon que les types de tâches aient pour objet :

- le *contenu* : relatifs à l'enseignement des concepts ;
- l'*activité* : en rapport avec la forme des tâches proposées aux élèves, nous différencions la *démarche* (projet, démarche d'investigation, etc.), le *type d'activité* (écrire, modifier, compléter, etc.) et le *mode de programmation* (boucles d'essais-erreurs dans un langage de programmation, formalisation d'un algorithme puis traduction, etc.) ;
- le *dispositif* : en lien avec les *lieux* (salle de classe, salle informatique, etc.), l'*organisation de l'espace* (groupe classe, îlots, etc.) et les *rapports entre individus* (coopération, autonomie, etc.) ;
- les *artefacts* : concernent les artefacts matériels, logiciels et symboliques, nous discernons les *artefacts-supports* qui permettent de formaliser des algorithmes et de fabriquer des programmes (ordinateur, éditeur de programme, langages de programmation, etc.) des *artefacts-cibles* qui sont ceux contrôlés par les programmes (robots, etc.)

Nous indiquons donc, en rouge après le type de tâches, le ou les indicateurs qui le caractérisent. En effet, ces catégories ne sont pas forcément étanches entre elles et certains types de tâches concernent plusieurs indicateurs. Par exemple, le type de tâches « Faire travailler les élèves en mode débranché » (voir Annexe A.1.2.1), comporte, lorsqu'on explore les techniques proposées, à la fois des aspects relatifs à l'activité (dans la cour de récréation, travail en binôme, coopération) et aux artefacts (écrire un programme « à la main », « contrôler » un autre élève).

Dans une dernière étape, afin de faciliter nos analyses, nous résumons les organisations didactiques modélisées. Nous procédons en créant des *Synthèses d'Organisations Didactiques* (SOD) qui fusionnent les types de tâches similaires par indicateur. Ces SOD consistent en un type de tâche suivi, après deux points, de ses techniques séparées par des points-virgules (lorsqu'elles sont disponibles). À cette occasion, nous :

- abandonnons les technologies même si nous pouvons y faire référence dans les analyses rédigées ;
- négligeons les organisations didactiques relatives aux contenus en rapport avec les sujets déjà traités : habitats, concepts mis en jeu par la programmation et leurs niches ;

- fusionnons les types de tâches proches et ainsi regroupons les techniques afférentes ;
- scindons les organisations didactiques relevant de plusieurs indicateurs, par exemple le type de tâche « Faire travailler les élèves en mode débranché » se retrouve dans la SOD « Faire travailler les élèves en salle de classe, en salle informatique ou dans la cour » de type activité et la SOD « Proposer des activités débranchées : simuler physiquement les déplacements des lutins sur un quadrillage » de type artefact.

Nous classons enfin ces SOD dans des tableaux séparés par discipline et par indicateur (voir section 2.8). Dans chacun de ces tableaux, nous finissons par regrouper les SOD en fonction des sous-indicateurs que nous avons établi dans la section 2.2.2. Dans un souci de simplification, nous utilisons le terme « catégorie » pour désigner ces sous-indicateurs. Nos analyses s'appuient donc sur ces tableaux synthétiques, nous citons ensuite au fil des paragraphes des extraits des textes du corpus afin de soutenir nos propos.

Comparaisons collège-lycée (Q2.6)

Enfin, pour cette dernière question de recherche, nous repartons de l'ensemble des résultats établis dans ce chapitre pour les interpréter au prisme de la transition collège-lycée. Ceci dans l'objectif d'y repérer des différences et d'essayer d'inférer d'éventuels obstacles et difficultés pour les élèves.

Nous venons de détailler notre méthodologie pour ce chapitre, nous présentons les résultats qui en découlent dans les sections qui suivent.

2.4 Trois disciplines hôtes pour la programmation informatique

Dans cette section, nous répondons à la question de recherche Q2.1 en déterminant les disciplines scolaires dans lesquelles s'insère la programmation informatique au cycle 4 et au niveau seconde. Nous cherchons ensuite dans les textes officiels les arguments justifiant cette présence.

2.4.1 Disciplines hôtes

En parcourant les programmes scolaires du cycle 4 (MEN, 2020b) et du niveau seconde (MEN, 2018), nous avons identifié les disciplines suivantes :

- les mathématiques au cycle 4 (3h30/semaine) et au niveau seconde (4h/semaine) ;
- la technologie au cycle 4 (1h30/semaine) ;
- les Sciences Numériques et Technologie (SNT) au niveau seconde (1h30/semaine).

Il s'agit ensuite de se demander pourquoi la programmation informatique est enseignée dans chacun de ces quatre cours ? Quelles justifications trouve-t-on dans les différents textes officiels ? Pour répondre à ces questions, nous analysons notre corpus pour repérer les éléments discursifs pouvant légitimer cette présence.

2.4.2 Légitimation

Au cycle 4, nous ne trouvons aucun élément de justification dans les textes régissant l'enseignement des mathématiques. En revanche, pour la technologie, la ressource d'accompagnement R1T4 précise que « l'informatique est omniprésente dans les objets et systèmes technologiques qui assistent l'homme au quotidien ainsi que dans les services qui l'aident à la prise de décisions. Elle prend donc place naturellement dans le programme de technologie » (MEN, 2016a, p. 17). Il s'agit, à travers ces lignes, d'affirmer que l'informatique, et en corollaire la programmation, constituant désormais un aspect prépondérant des artefacts techniques qui sont habituellement les objets d'étude de la discipline « technologie », fait dorénavant partie du périmètre de cette discipline.

Au niveau seconde, en mathématiques, nous pouvons relever dans le programme d'enseignement PMS les passages suivants : «la démarche algorithmique est, depuis les origines, une composante essentielle de l'activité mathématique » (MEN, 2019b, p. 14) et « l'algorithmique a une place naturelle dans tous les champs des mathématiques » (MEN, 2019b, p. 15). L'algorithmique y est présentée comme faisant historiquement partie des mathématiques, il est donc légitime que la programmation informatique, en tant qu'outil permettant de les opérationnaliser, soit enseignée dans cette discipline. En SNT, l'un des objectifs principaux est, comme son nom l'indique[12], d'enseigner les « sciences numériques ». La programmation informatique y tient une place importante. En effet, dans son préambule, le programme d'enseignement PSS précise que « malgré leur grande variété, [les] avancées [technologiques] se fondent toutes sur l'universalité et la flexibilité d'un petit nombre de concepts en interaction : les données [...] ; les algorithmes [...] ; les langages [...] ; les machines [...] » (MEN, 2019e, p. 2). On retrouve ici les quatre concepts de l'informatique proposés par Dowek (2011) et l'idée selon laquelle la programmation en est l'élément central (voir section 1.4).

2.4.3 Synthèse de la section

Q2.1 : Dans quelles disciplines est enseignée la programmation informatique à la transition collège-lycée ? Quelles légitimations sont avancées dans les textes officiels ?

La programmation informatique est enseignée à la transition collège-lycée dans les disciples suivantes : **mathématiques** (cycle 4 et seconde), **technologie** (cycle 4) et **Sciences Numériques et Technologie** (seconde). Excepté en mathématiques au cycle 4, les textes officiels

[12] Remarquons plusieurs choses au sujet du nom de la discipline « Sciences numériques et technologie » au regard du savoir savant (voir section 1.2.3). D'abord, l'emploi du terme « numérique » en lieu et place du vocable « informatique » renvoie plutôt aux outils et aux usages qu'à la théorie et aux techniques informatiques. Ensuite, l'usage du pluriel laisse entendre qu'il y aurait plusieurs sciences qui traitent de l'informatique qui ne constituerait pas, en-elle-même, une science établie et unifiée. Enfin, le terme « technologie » au singulier semble être employé dans son sens étymologique (étude des techniques, comme pour la discipline « technologie ») plutôt que dans le sens, inspiré de l'anglais « information technology », de l'ensemble des outils et usages sociaux de l'informatique. Une dénomination disciplinaire cohérente avec le savoir savant aurait pu être : « science informatique et technologies ».

légitiment la présence de la programmation au sein des disciplines. Pour ce faire, ils utilisent des arguments différents :

- en technologie : les **artefacts techniques utilisés dans la société sont les objets d'étude de la technologie**, puisqu'ils sont désormais liés à l'informatique, la discipline intègre la programmation informatique ;
- en mathématiques en seconde : **les algorithmes sont historiquement liés aux mathématiques**, la programmation informatique prend donc naturellement sa place dans cette discipline ;
- en SNT : **cette discipline a pour objet l'étude des « sciences numériques »**, or la programmation informatique en est l'élément central.

Nous venons d'exposer les éléments légitimant la présence de la programmation informatique dans les programmes scolaires des différentes disciplines, attardons-nous désormais sur l'habitat de la programmation dans ces disciplines respectives.

2.5 Habitat de la programmation dans chaque discipline

Nous traitons dans cette section la question de recherche Q2.2 qui interroge la manière dont la programmation informatique s'insère dans les programmes d'enseignement du cycle 4 et de la seconde. Nous allons donc étudier son habitat au sens d'Artaud (voir section 2.2.1). Il s'agit donc de se demander : quels sont les différents « lieux » où elle se trouve ? Avec quels objets est-elle en interaction ?

Nous commençons donc par examiner la place de la programmation informatique dans le découpage hiérarchique des programmes en déterminant à quels niveaux elle intervient dans l'échelle des niveaux de codétermination didactique définie par Chevallard (voir section 2.2.1). Ensuite, nous analysons de quelle manière la programmation informatique s'articule avec les autres domaines du programme. Enfin, nous observons ses relations avec les autres disciplines dans un même niveau scolaire. Nous présentons ces trois aspects pour chaque discipline dans les sous-sections suivantes.

2.5.1 Mathématiques au cycle 4

Au cycle 4, le programme d'enseignement de mathématiques PM4 est structuré selon cinq domaines nommés « thèmes » :

– « nombres et calculs » ;

– « organisation et gestion de données, fonctions » ;

– « grandeurs et mesures » ;

– « espace et géométrie » ;

– « algorithmique et programmation » (MEN, 2020b, p. 126).

Nous pouvons tout d'abord constater que, pour cette discipline, la programmation informatique fait l'objet d'un domaine à part entière nommé « algorithmique et

programmation ». Ce domaine est lui-même décliné dans un unique secteur : « Écrire, mettre au point, exécuter un programme » (MEN, 2020b, p. 136).

Ce domaine lié à la programmation est quasiment autonome par rapport aux quatre autres. En effet, le programme d'enseignement ne comporte aucun lien évident avec les autres thèmes. Ainsi, les objectifs proposés concernant les activités informatiques ne sont pas directement en relation avec les mathématiques. Le programme indique que les élèves doivent « s'initier à la programmation, en développant [...] quelques programmes simples » (MEN, 2020b, p. 136) et « Écrire, mettre au point (tester, corriger) et exécuter un programme en réponse à un problème donné » (MEN, 2020b, p. 136). Des exemples de mises en œuvre sont proposés, il s'agit essentiellement de programmer des jeux : « jeux dans un labyrinthe, jeu de Pong, bataille navale, jeu de Nim, Tic-tac-toe, jeu du cadavre exquis » (MEN, 2020b, p. 136). La ressource d'accompagnement R1M4 propose, en complément, des exemples d'activités que nous avons modélisées sous forme d'organisations informatiques (voir Annexe A.1.2.2). Nous en reprenons ici uniquement les types de tâches pères pour s'en faire un idée :

- déplacer un personnage selon un parcours prédéfini ;
- faire produire à un personnage des tracés géométriques donnés ;
- programmer un jeu permettant de diriger un personnage en interaction avec d'autres éléments ;
- programmer le jeu Pong ;
- programmer un jeu basé sur le retournement de pièces ;
- programmer le jeu Tic-tac-toe.

Nous pouvons établir que, en accord avec le programme d'enseignement, ces activités ont un but ludique : concevoir des jeux ou effectuer des tracés géométriques, et ne visent pas principalement des concepts mathématiques.

Finissons par évoquer les interactions avec les autres disciplines et en particulier avec la technologie. Le programme indique que l'enseignement de l'informatique doit être effectué de façon concertée avec le cours de technologie : « [l']algorithmique et programmation [...] entre dans le cadre d'un enseignement de l'informatique dispensé conjointement en mathématiques et en technologie » (MEN, 2020b, p. 126) mais ne fournit néanmoins aucune précision quant à la manière de procéder.

Nous résumons l'habitat de la programmation informatique dans le programme de mathématiques au cycle 4 dans la Figure 2-8.

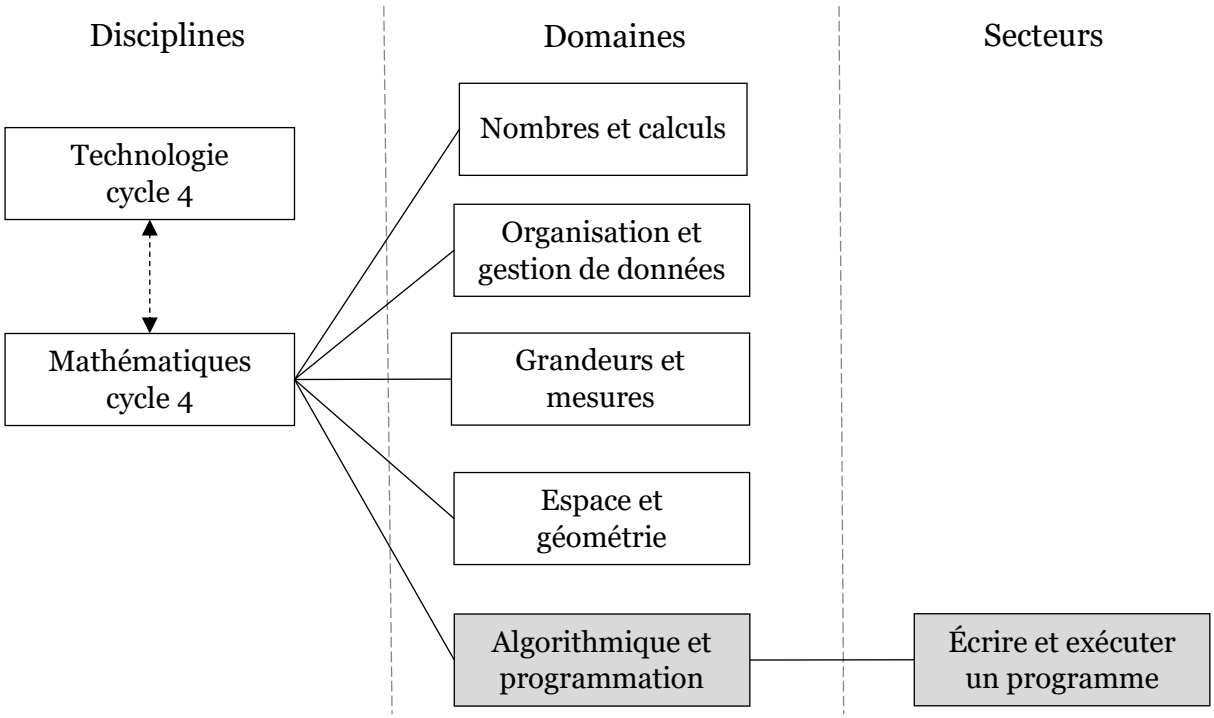


Figure 2-8 –Découpage hiérarchique du programme de mathématiques au cycle 4, place de la programmation informatique (cadres grisés) et interactions (flèches pointillées)

2.5.2 Technologie au cycle 4

Le programme d'enseignement de technologie PT4 est organisé autour de trois domaines appelés « grandes thématiques » :

- « le design, l'innovation, la créativité » ;
- « les objets techniques, les services et les changements induits dans la société » ;
- « la modélisation et la simulation des objets techniques » (MEN, 2020b, p. 116).

Ces trois domaines sont complétés par un quatrième, n'ayant pas exactement le même statut car présenté à part et dans un second temps : « En outre, un enseignement d'informatique, est dispensé » (MEN, 2020b, p. 116) et « ces trois thématiques s'articulent avec une quatrième qui est liée à l'enseignement d'informatique » (MEN, 2016a, p. 4). Ce quatrième domaine est nommé : « L'informatique et la programmation » (MEN, 2020b, p. 122). Il est divisé en deux secteurs : « Comprendre le fonctionnement d'un réseau informatique » et « Écrire, mettre au point et exécuter un programme » (MEN, 2020b, p. 123).

Les activités prescrites consistent principalement à programmer des systèmes technologiques en répondant à un cahier des charges précisant un besoin. Nous citons ci-après des extraits de textes officiels appuyant cette affirmation : « Dans le cadre des projets, les élèves […] conçoivent tout ou partie d'un programme, le compilent et l'exécutent pour répondre au besoin du système et des fonctions à réaliser » (MEN, 2020b, p. 123) ; « Les concepts, que devront acquérir les élèves, seront abordés […] par une approche concrète et active, et autant qu'il est possible de le faire à partir des objets

et systèmes présents dans le laboratoire » (MEN, 2016a, p. 4), et « Ces compétences [en algorithmique et programmation] ont […] vocation à être […] réinvesties dans une problématique plus complexe de pilotage de système en technologie » (MEN, 2016a, p. 18)

Des exemples d'activités sont proposés dans la ressource d'accompagnement R2T4. Nous les avons modélisés sous forme d'organisation informatique (voir Annexe A.2.3.1). Nous énumérons ci-dessous les types de tâches pères dans l'objectif d'en donner un aperçu :

- créer une carte interactive permettant de représenter sa ville ;
- programmer un éclairage urbain automatique ;
- piloter un panneau solaire pour optimiser sa position ;
- programmer un mini-robot pour éviter des obstacles et suivre une ligne ;
- programmer les effecteurs d'une serre en fonction de sa température intérieure ;
- réaliser une application simple pour smartphone ;
- programmer l'ouverture automatique d'un portail ;
- programmer une webcam pour un système de vidéosurveillance.

Au regard de cette liste, nous pouvons avancer que les activités de programmation sont inscrites dans le contexte de la discipline technologie dans le sens où elles s'appuient systématiquement sur des systèmes techniques, cependant elles ne s'articulent pas réellement avec les autres domaines du programme.

Nous présentons, dans le Tableau 2-2, les quelques croisements néanmoins relevés dans le programme d'enseignement (MEN, 2020b) :

Domaines	Références à la programmation informatique
Design, innovation et créativité	**Imaginer un programme informatique répondant à un besoin** : « Imaginer des solutions pour produire des objets et des éléments de programmes informatiques en réponse au besoin. » (MEN, 2020b, p. 119)
Objets techniques, services et changements induits dans la société	**Exprimer sa pensée à travers un algorithme** : « Exprimer sa pensée à l'aide d'outils de description adaptés : […] Notion d'algorithme. » (MEN, 2020b, p. 120)
Modélisation et simulation des objets techniques	∅

Tableau 2-2 - Références à la programmation informatique dans les autres domaines du programme de technologie au cycle 4

Nous observons un recoupement avec le premier domaine qui revêt une forte dominante de conception (analyse de besoins, formulation d'un cahier des charges, gestion de projet) et qui est davantage porté sur les réalisations matérielles (impression 3D, assemblages électroniques, etc.) que logicielles. Dans le deuxième domaine, l'algorithme est utilisé comme un outil de description parmi d'autres dans le but d'exprimer sa pensée avec une visée d'« étude des conditions d'utilisation des objets et des services ancrés dans leur réalité sociale » qui « permet […] de développer des

compétences associées à une compréhension critique des objets et systèmes techniques. » (MEN, 2020b, p. 119).

Qu'en-est-il des interactions de la programmation en technologie avec la discipline mathématique ? Comme le précisent les textes officiels PT4 et R1T4, la programmation informatique a vocation à être enseignée de façon concertée avec les mathématiques : « un enseignement d'informatique, est dispensé à la fois dans le cadre des mathématiques et de la technologie » (MEN, 2020b, p. 116) ; « Les notions d'algorithmique sont traitées conjointement en mathématiques et en technologie » (MEN, 2020b, p. 123) et « l'enseignement d'informatique dont la responsabilité incombe aux professeurs de technologie et de mathématiques » (MEN, 2016a, p. 4). À la différence du programme de mathématiques, les conditions de cette coopération sont davantage explicitées :

> Une partie des compétences de l'enseignement d'informatique en technologie est commune avec le thème « algorithmique et programmation » du programme de mathématiques. Ces compétences développées dans l'une ou l'autre discipline n'ont pas vocation à être vues deux fois de manière séparée, mais à être réinvesties dans une problématique plus complexe de pilotage de système en technologie. Il appartient aux professeurs de ces deux disciplines de mettre en cohérence l'enseignement de l'informatique par la définition de thématiques communes. (MEN, 2016a, p. 18)

Il s'agit donc d'une invitation aux enseignants des deux matières à coopérer dans le but d'éviter d'introduire concomitamment les mêmes notions. La responsabilité d'introduction de ces notions élémentaires n'est pas clairement tranchée mais il semble tout de même possible de lire entre les lignes qu'elle incombe plutôt au professeur de mathématiques dans la mesure où celui de technologie est amené à les « réinvestir ».

La Figure 2-9 présente une synthèse de l'habitat de la programmation informatique en technologie au cycle 4.

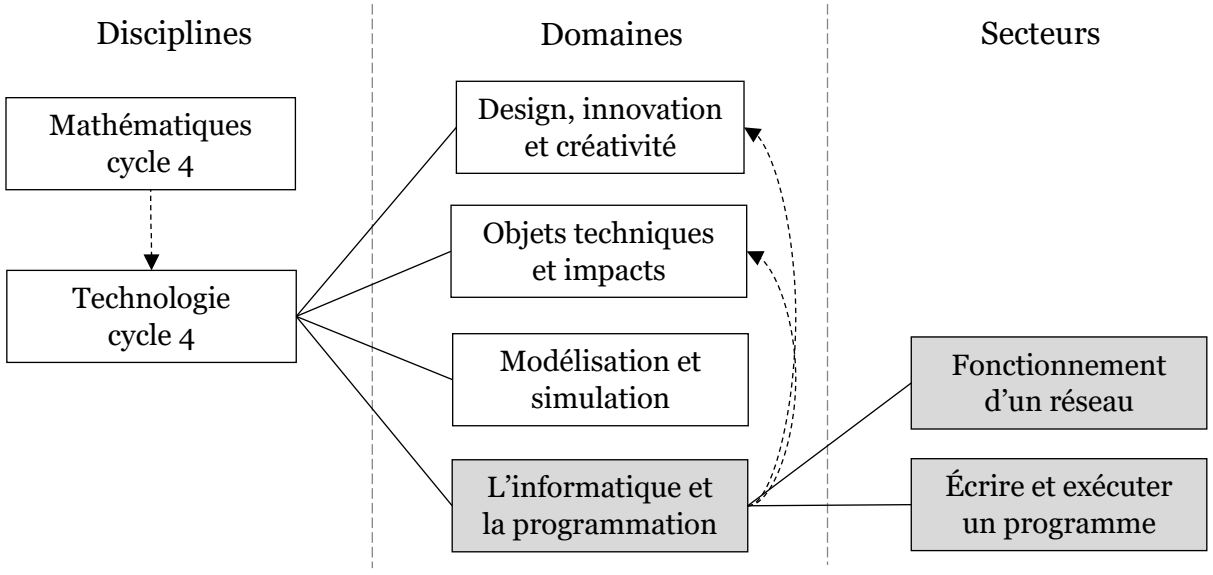


Figure 2-9 –Découpage hiérarchique du programme de technologie au cycle 4, place de la programmation informatique (cadres grisés) et interactions (flèches pointillées)

2.5.3 Mathématiques en seconde

En seconde, le programme d'enseignement de mathématiques (MEN, 2019b) est partagé en cinq domaines baptisés « grandes parties » que nous listons ci-dessous :

- « nombres et calculs » ;
- « géométrie » ;
- « fonctions » ;
- « statistiques et probabilités » ;
- « algorithmique et programmation » (MEN, 2019b, p. 5).

Pour cette discipline, la programmation informatique fait donc l'objet d'un domaine spécifique nommé « Algorithmique et programmation ». Ce domaine est décliné en deux secteurs : « Utiliser les variables et les instructions élémentaires » et « Notion de fonction » (MEN, 2019b, p. 15). De plus, les quatre autres domaines comportent des secteurs qui contiennent un thème « Exemples d'algorithme » (MEN, 2019b).

Contrairement au cycle 4, le programme stipule explicitement qu'« il est essentiel d'exploiter les possibilités d'interaction entre ces parties » (MEN, 2019b, p. 5), que « l'algorithmique a une place naturelle dans tous les champs des mathématiques et les problèmes ainsi traités doivent être en relation avec les autres parties du programme » (MEN, 2019b, p. 15), et encore que : « il s'agit donc d'adosser explicitement les activités de la partie algorithmique et programmation aux mathématiques » (MEN, 2017b, p. 1). Nous trouvons, par conséquent, dans le programme, de nombreux exemples d'algorithmes à mettre en œuvre dans tous les autres domaines (thème « Exemples d'algorithme »), nous les avons consignés dans le Tableau 2-3.

Domaines	Références à la programmation informatique
Nombres et calculs	**Déterminer un encadrement de $\sqrt{2}$** : « Exemple d'algorithme : Déterminer par balayage un encadrement de $\sqrt{2}$ d'amplitude inférieure ou égale à 10^{-n} » (MEN, 2019b, p. 6) **Déterminer la première puissance d'un nombre supérieure à une valeur** :« Exemple d'algorithme : Déterminer la première puissance d'un nombre positif donné supérieure ou inférieure à une valeur donnée. » (MEN, 2019b, p. 8)
Géométrie	**Étudier l'alignement de trois points du plan** : « Exemples d'algorithme : Étudier l'alignement de trois points dans le plan. » p. 10 **Déterminer une équation de droite passant par deux points** : « Exemples d'algorithme : […] Déterminer une équation de droite passant par deux points donnés. » (MEN, 2019b, p. 10)
Fonctions	**Programmer des fonctions mathématiques en Python** : « Les outils numériques sont mis à profit : […] Python, le tableur ou la calculatrice, pour mettre en évidence l'aspect de programme de calcul [des fonctions]. » (MEN, 2019b, p. 11) **Approximer un extremum de fonction** : « Exemples d'algorithme : Pour une fonction dont le tableau de variations est donné, algorithmes d'approximation numérique d'un extremum (balayage, dichotomie). » (MEN, 2019b, p. 12) **Calculer la longueur d'une portion de courbe représentative** : « Exemples d'algorithme : Algorithme de calcul approché de longueur d'une portion de courbe représentative de fonction. » (MEN, 2019b, p. 12)

Statistiques et probabilités	**Comprendre une fonction Python renvoyant les indicateurs d'une série statistique** : « Capacités attendues : Pour des données réelles ou issues d'une simulation, lire et comprendre une fonction écrite en Python renvoyant la moyenne m, l'écart type s, et la proportion d'éléments appartenant à $[m - 2s\,;\ m + 2s]$ » (MEN, 2019b, p. 13) **Comprendre une fonction Python renvoyant le nombre de succès dans un échantillon pour une expérience à deux issues** : « Capacités attendues : Lire et comprendre une fonction Python renvoyant le nombre ou la fréquence de succès dans un échantillon de taille n pour une expérience aléatoire à deux issues. » (MEN, 2019b, p. 14) **Observer la loi des grands nombres à l'aide d'une simulation** : « Capacités attendues : […] Observer la loi des grands nombres à l'aide d'une simulation sur Python ou tableur » (MEN, 2019b, p. 14) **Simuler plusieurs échantillons et calculer la proportion de cas où la proportion est dans l'intervalle de confiance** : « Capacités attendues : […] Simuler N échantillons de taille n d'une expérience aléatoire à deux issues. Si p est la probabilité d'une issue et f sa fréquence observée dans un échantillon, calculer la proportion des cas où l'écart entre p et f est inférieur ou égal à $\frac{1}{\sqrt{n}}$. » (MEN, 2019b, p. 14)

Tableau 2-3 - Références à la programmation informatique dans les autres domaines du programme de mathématiques en seconde

Nous listons, en complément, les exemples d'activités proposées dans la ressource d'accompagnement R1MS que nous avons, par ailleurs, modélisé sous forme d'organisations informatiques (voir Annexe A.3.2.2). Nous donnons l'intitulé des types de tâches pères et nous précisons le détail des ingrédients de technique afin de donner une idée plus précise des activités :

- étudier un tremplin de ski : vérifier l'alignement de trois points ; calculer l'équation réduite d'une droite ; calculer une image par une fonction affine ; calculer la longueur d'une portion de courbe ;
- encadrer $\sqrt{2}$ par balayage : représenter graphiquement une fonction ; trouver une valeur approché par défaut de $\sqrt{2}$ par balayage ; donner un encadrement de $\sqrt{2}$;
- donner le nombre de chiffres d'un nombre : déterminer l'exposant de la plus petite puissance de 10 strictement supérieure à un nombre ;
- programmer un jeu de dé : calculer si un triangle est constructible ; modéliser une expérience aléatoire à deux issues ; calculer la fréquence de succès dans un échantillon simulé ; observer la fluctuation d'échantillonnage et la loi des grands nombres ; calculer la proportion de cas où la proportion est dans l'intervalle de confiance.

Nous pouvons remarquer que ces activités répondent très précisément aux attendus du programme au niveau algorithmique et qu'elles en balayent quasiment tous les items.

Les éléments précédents nous permettent d'affirmer qu'en seconde la programmation informatique est un outil au service des mathématiques. En effet, elle permet : d'implémenter des algorithmes issus de textes historiques (valeur approchée de $\sqrt{2}$), de mettre en œuvre des méthodes de calcul par approximations (longueur d'une courbe, recherche des extremums d'une fonction), d'automatiser des calculs

fastidieux (alignements de points, calcul d'une équation de droite) et de faire découvrir de manière expérimentale les notions autour de l'échantillonnage.

Évaluons maintenant les interactions avec les SNT. En seconde, la programmation informatique est un enseignement partagé entre les mathématiques et les SNT, cependant, les textes officiels régissant les mathématiques n'y font aucune référence. Cela peut être interprété comme une injonction aux professeurs de mathématiques à introduire les concepts dans leur discipline, sans se préoccuper d'une éventuelle synchronisation avec les enseignants de SNT. Les seconds pourront, dans un deuxième temps, réinvestir ces notions dans leur discipline.

Nous fournissons dans la Figure 2-10 un récapitulatif de l'habitat de la programmation informatique dans les programmes de mathématiques en seconde.

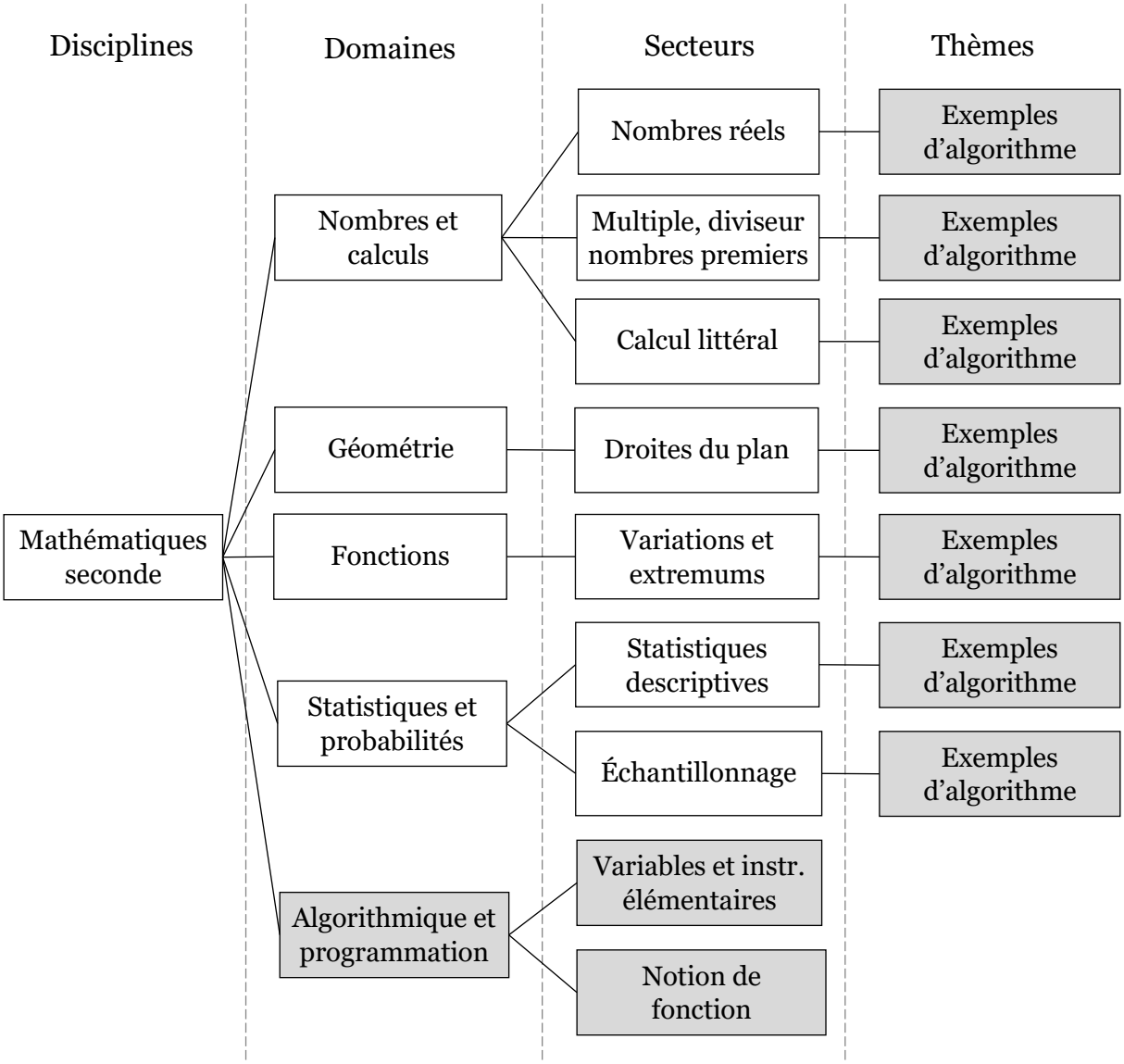


Figure 2-10 –Découpage hiérarchique du programme de mathématiques en seconde, place de la programmation informatique (cadres grisés)

2.5.4 SNT en seconde

Le programme de SNT (MEN, 2019e) est composé de sept domaines nommés « thématiques », nous les énumérons ci-dessous :

- « Internet » ;
- « le Web » ;
- « les réseaux sociaux » ;
- « les données structurées et leur traitement » ;
- « localisation, cartographie et mobilité » ;
- « informatique embarquée et objets connectés » ;
- « photographie numérique » (MEN, 2019e, p. 3).

La programmation informatique ne fait pas l'objet d'un domaine particulier mais est présentée séparément comme un ensemble de « notions transversales » (MEN, 2019e, p. 3). Elle forme donc un secteur propre dans chacun des domaines en constituant à chaque fois une facette de cette « thématique ». Comme nous l'avons indiqué dans la section 2.4, la programmation informatique est présentée en SNT comme l'élément central à l'intersection des quatre concepts qui fondent la science informatique. Ainsi, dans le programme d'enseignement, le déroulé de chaque domaine comporte un secteur « les algorithmes et les programmes » listant les usages industriels des algorithmes et programmes informatiques pour cette « thématique ».

Les activités de programmation sont nettement inscrites dans les différents domaines, nous pouvons nous en convaincre un peu plus à la lecture du programme d'enseignement : « On approfondit [...] la pratique de la programmation à travers les activités liées aux thèmes du programme » (MEN, 2019e, p. 3-4) et « Exemples d'activités : Illustrer ces notions [d'algorithmique] par des activités liées aux différents thèmes du programme. » (MEN, 2019e, p. 3). Le programme donne également des exemples d'algorithmes que les élèves peuvent implémenter. Ils découlent directement de la liste des usages industriels repérés pour chaque domaine en gardant ceux dont la complexité et les notions mises en jeu restent accessibles à des élèves de seconde. Nous retranscrivons dans le Tableau 2-4 ces exemples d'activités.

Domaines	Références à la programmation informatique
Internet	**Suivre des données sur le réseau** : « Suivre le chemin d'un courriel en utilisant une commande du protocole IP. » (MEN, 2019e, p. 6)
Le Web	**Programmer un algorithme de calcul de popularité d'une page Web** : « Calculer la popularité d'une page à l'aide d'un graphe simple puis programmer l'algorithme » (MEN, 2019e, p. 8)
Les réseaux sociaux	**Manipuler des graphes de relation** :« Construire ou utiliser une représentation du graphe des relations d'un utilisateur. S'appuyer sur la densité des liens pour identifier des groupes, des communautés. » (MEN, 2019e, p. 11)
Les données structurées et leur traitement	**Télécharger des données ouvertes et observer les traitements possibles avec Python** : « Télécharger des données ouvertes (sous forme d'un fichier au format CSV avec les métadonnées associées), observer les différences de traitements possibles selon le logiciel choisi pour lire le fichier : programme Python, [...] » (MEN, 2019e, p. 13)
Localisation, cartographie et mobilité	**Calculer un itinéraire** :« Calculer un itinéraire routier entre deux points à partir d'une carte numérique » (MEN, 2019e, p. 15) **Placer des points repérés sur une carte numérique** : « Situer sur une carte numérique la position récupérée » (MEN, 2019e, p. 15)

Informatique embarquée et objets connectées	**Identifier les algorithmes de contrôle des comportements physiques** : « Systèmes informatiques embarqués. Identifier des algorithmes de contrôle des comportements physiques à travers les données des capteurs, l'IHM et les actions des actionneurs dans des systèmes courants. » (MEN, 2019e, p. 17) **Réaliser l'IHM simple d'un objet connecté** : « Interface homme-machine (IHM). Réaliser une IHM simple d'un objet connecté. » (MEN, 2019e, p. 17) **Programmer l'acquisition de données ou la commande d'un actionneur** : « Partie Informatique embarquée et objets connectés : Commande d'un actionneur, acquisition des données d'un capteur. Écrire des programmes simples d'acquisition de données ou de commande d'un actionneur. » (MEN, 2019e, p. 17)
Photographie numérique	**Programmer un traitement de transformation d'image** : « Traitement d'image. Traiter par programme une image pour la transformer en agissant sur les trois composantes de ses pixels. » (MEN, 2019e, p. 19) **Expliciter les algorithmes liés à la prise de vue** : « Rôle des algorithmes dans les appareils photo numériques. Expliciter des algorithmes associés à la prise de vue. Identifier les étapes de la construction de l'image finale. » (MEN, 2019e, p. 19)

Tableau 2-4 - Références à la programmation informatique dans les autres domaines du programme de SNT seconde

Nous dressons également la liste des activités proposées dans la ressource d'accompagnement R1SS que nous avons modélisées sous forme praxéologique (voir Annexe A.4.2.1). Nous donnons la dénomination des types de tâches pères :

- traiter des données ouvertes à l'aide d'un programme ;
- programmer un robot Thymio ;
- construire une image numérique à l'aide d'un programme ;
- programmer le traitement d'une image numérique.

Ces propositions d'activités sont dans la droite lignée du programme d'enseignement en proposant des mises en œuvre d'algorithmes en lien fort avec les autres domaines.

Terminons en évoquant les liens avec la discipline mathématique. Un paragraphe du programme détaille succinctement comment doivent opérer ces interactions :

> Au collège (cycle 4), les élèves ont découvert et pratiqué les éléments fondamentaux d'algorithmique et de programmation. Le programme de seconde de mathématiques approfondit l'apprentissage de la programmation. Une coordination avec le cours de mathématiques est donc nécessaire pour déterminer à quel moment des éléments de programmation peuvent être utilisés en sciences numériques et technologie. (MEN, 2019e, p. 4).

Il s'agit donc pour le professeur de SNT de se coordonner avec ses collègues en charge des mathématiques, la dernière phrase précise que l'approfondissement des concepts est à la l'initiative des professeurs de mathématiques, les cours de SNT permettant, dans un second temps, de les mettre en œuvre.

Comme pour les autres disciplines, nous résumons dans la Figure 2-11 l'habitat de la programmation informatique en SNT.

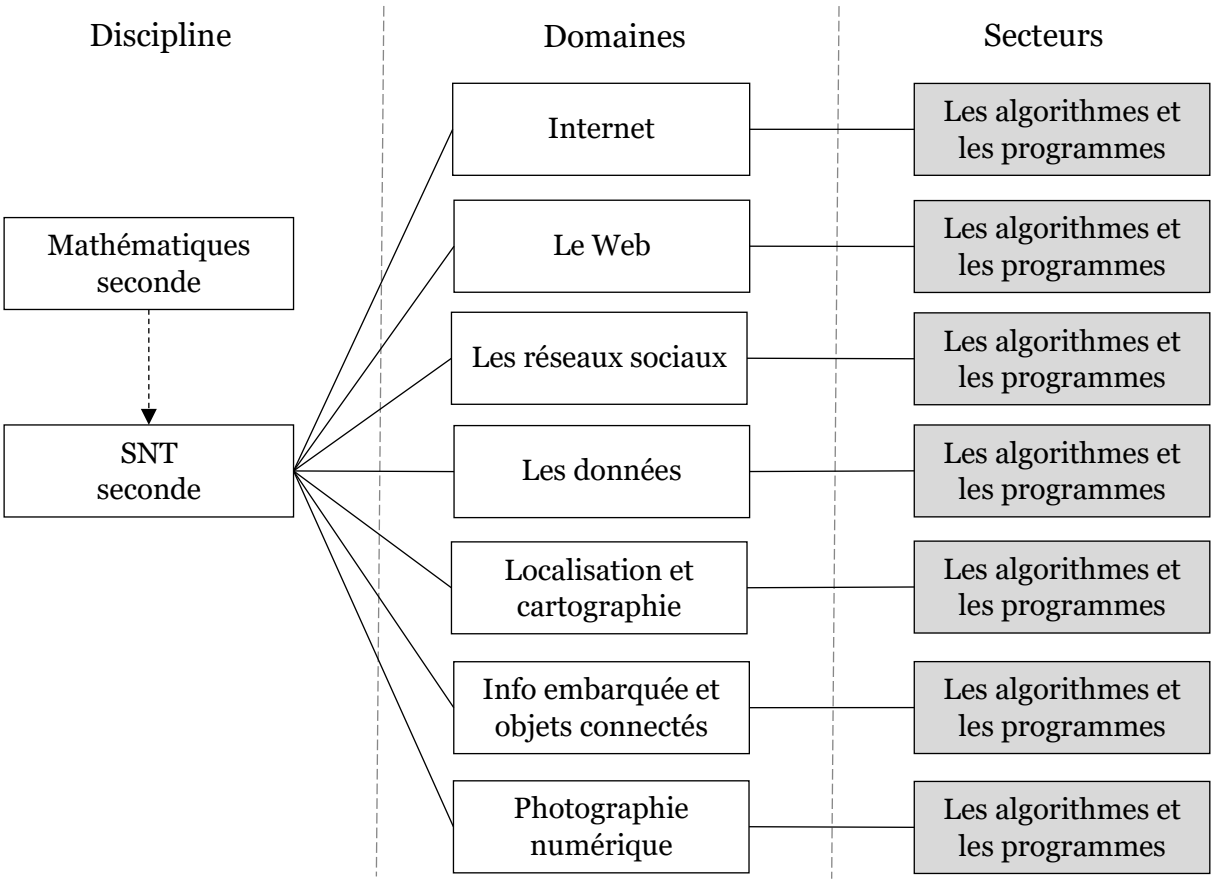


Figure 2-11 –Découpage hiérarchique du programme de SNT en seconde, place de la programmation informatique (cadres grisés) et interactions (flèches pointillées)

2.5.5 Bilan transversal aux disciplines

Nous venons de détailler pour chacune des quatre disciplines qui nous occupent : la place de la programmation informatique dans le découpage hiérarchique des programmes, son articulation avec les autres domaines, et ses interactions avec les autres disciplines. Nous allons, à présent, partir de ces éléments afin d'établir un bilan transdisciplinaire.

Évoquons d'abord les aspects d'agencement hiérarchique. La programmation fait l'objet d'un domaine spécifique en mathématiques (cycle 4 et seconde) ainsi qu'en technologie. En SNT, elle est présentée comme un ensemble de « notions transversales » aux sept autres domaines, elle constitue pour chacun d'entre eux un secteur à part. Au vu du peu d'interactions qu'a la programmation informatique avec les autres domaines au cycle 4, il semble logique qu'elle constitue un domaine à part entière. À l'inverse, en seconde, la programmation est en interaction forte avec les autres domaines du programme. Elle y est présente en tant que thèmes en mathématiques et en tant que secteurs en SNT. Nous pouvons par conséquent nous interroger sur l'existence d'un domaine dédié en mathématique seconde, tant les interactions sont nombreuses avec les autres domaines. Il s'agissait sans doute ici de donner une place importante aux concepts fondamentaux de la programmation qui constituent, en eux-mêmes, un objectif d'apprentissage.

Ensuite, intéressons-nous à l'articulation de la programmation informatique avec les autres domaines des programmes. On retrouve au cycle 4 des éléments concordants : en mathématiques, la programmation informatique est quasiment autarcique n'offrant que quelques rares opportunités de lien avec les autres domaines. L'objectif clairement affirmé est l'initiation à la programmation à travers des activités ludiques. En technologie, le domaine est également relativement indépendant des autres, malgré quelques rapprochements possibles avec le design ou l'expression de la pensée, la finalité première restant la programmation de systèmes techniques en réponse à un problème posé en termes de besoins. Au lycée, à l'inverse, la programmation informatique est fortement enchevêtrée dans les autres domaines du programme. En mathématiques, elle intervient dans tous les domaines, et se met au service des mathématiques. Nous trouvons des similitudes en SNT, avec une programmation informatique qui n'a pas réellement d'existence propre, mais qui se trouve intriquée dans les autres domaines. Il s'agit ici de s'inspirer des usages industriels ayant cours afin de proposer aux élèves des exemples d'utilisations de la programmation, restant à leur portée, dans les sept domaines du programme.

Enfin, penchons-nous sur les interactions entre disciplines dans les deux niveaux. Au cycle 4, les instructions officielles de mathématiques font simplement état du fait que l'enseignement de la programmation doit être dispensé conjointement avec la technologie. En technologie, les textes indiquent que les notions ne doivent pas être introduites deux fois séparément et il est suggéré que les professeurs de technologie « réinvestissent » les compétences de programmation, qui auraient été préalablement introduites en mathématiques. Ce partage des responsabilités plus ou moins implicite est assez naturel car il est en adéquation avec les objectifs fixés : introduire les notions algorithmiques en mathématiques et les utiliser pour programmer des artefacts en technologie. Pour ce qui est de la seconde, le découpage est, à priori, moins évident car l'apprentissage de la programmation ne constitue la visée principale d'aucune des deux matières. Le programme de mathématiques ne dit rien de cet enseignement conjoint. Celui de technologie laisse entendre qu'il incombe aux enseignants de mathématiques d'approfondir les connaissances en programmation initiée au cycle 4, après quoi, ceux de SNT pourront les mettre en œuvre. Nous pouvons sans doute expliquer ce choix par un argument historique : le programme de mathématiques contient des éléments algorithmiques depuis 2009, les enseignants ayant certainement pris l'habitude d'enseigner ces notions, il est compréhensible que, dix ans plus tard, la nouvelle discipline SNT ne bouleverse pas cette situation.

2.5.6 Synthèse de la section

Q2.2 : Quels sont les habitats de la programmation informatique dans les programmes du cycle 4 et de la seconde ? Autrement dit, quels sont les différents lieux où elle se trouve et avec quels savoirs est-elle en interaction ?

En nous appuyant sur les textes officiels émanant du ministère, nous avons pu analyser l'**habitat** de la programmation informatique, c'est-à-dire son **lieu d'implantation** dans l'échelle des niveaux de codétermination ainsi

que ses **interactions** avec les autres objets. Les résultats en fonction des disciplines sont les suivants :

- en **mathématique au cycle 4**, la programmation possède un **domaine dédié** ayant **aucune interaction** avec les autres domaines, l'objectif est l'initiation aux concepts fondamentaux à travers des activités ludiques ;
- en **technologie au cycle 4**, la programmation informatique est incluse dans un **domaine dédié** et opère **quelques interactions** avec les autres domaines, l'objectif est de programmer des systèmes techniques en réponse à un problème posé en termes de besoins ;
- en **mathématique en seconde**, la programmation fait l'objet d'un **domaine dédié** et connait d'**importantes interactions** avec tous les autres domaines via des **thèmes propres**, l'objectif est double, approfondir les concepts fondamentaux et fournir un outil au service des mathématiques ;
- en **SNT en seconde**, la programmation est diluée dans des **secteurs dans les autres domaines**, elle a donc d'**importantes interactions** avec eux, l'objectif est de proposer des activités en lien avec ces domaines.

Nous venons d'étudier les habitats de la programmation informatique dans les programmes de seconde et du cycle 4, attachons-nous dorénavant à déceler précisément les concepts mis en jeu dans chaque discipline.

2.6 Les concepts de programmation ciblés par les programmes d'enseignement

Nous répondons dans cette section à la question de recherche Q2.3 qui vise à connaitre les concepts fondamentaux ciblés par les programmes d'enseignement dans chacune des disciplines que nous étudions.

2.6.1 Concepts visés par les programmes

À cette fin, nous reproduisons dans le Tableau 2-5, pour chaque programme, les extraits en rapport avec la programmation informatique dans la partie nommée « contenus » ou « connaissances » qui explicite les contenus conceptuels qui doivent être enseignés.

Discipline	Contenus/connaissances
Mathématiques cycle 4	« Connaissances : - Notions d'**algorithme** et de **programme**. - Notion de **variable** informatique. - Déclenchement d'une action par un **événement**. - **Séquences** d'instructions, **boucles**, instructions **conditionnelles**. » (MEN, 2020b, p. 136)
Technologie cycle 4	« Connaissances […] : - Notions d'**algorithme** et de **programme**. - Notion de **variable** informatique. - Déclenchement d'une action par un **événement**, **séquences** d'instructions, **boucles**, instructions **conditionnelles**.

	- Systèmes embarqués. - Forme et transmission du signal. - Capteur, actionneur, interface. » (MEN, 2020b, p. 123)
Mathématiques seconde	« Une consolidation des acquis du cycle 4 est proposée autour de deux idées essentielles : - la notion de **fonction** ; - la programmation comme production d'un texte dans un langage informatique. » (MEN, 2019b, p. 14) « Contenus : - **Variables** informatiques de **type entier, booléen, flottant, chaîne de caractères**. - **Affectation** (notée ← en langage naturel). - **Séquence** d'instructions. - Instruction **conditionnelle**. - **Boucle bornée** (for), **boucle non bornée** (while). - **Fonctions** à un ou plusieurs arguments. - Fonction renvoyant un nombre aléatoire. [pour produire des séries statistiques]» (MEN, 2019b, p. 15)
SNT seconde	« Contenus : - **Affectations**, **variables** - **Séquences** - Instructions **conditionnelles** - **Boucles bornées** et **non bornées** - Définitions et appels de **fonctions** » (MEN, 2019e, p. 3)

Tableau 2-5 – Extraits des programmes d'enseignement relatifs aux contenus, en gras, les concepts directement en lien avec la programmation

Remarquons qu'au cycle 4, les notions visées sont strictement les mêmes en mathématiques et technologie : algorithme, programme, variable, séquence, conditionnelle, boucles et programmation évènementielle. Nous pouvons expliquer cette concordance en avançant que ces deux programmes d'enseignement ont été rédigés au même moment en réponse à la même lettre de saisine du ministère relative à « l'introduction de connaissances et de compétences en informatique et en sciences du numérique dans les programmes de la scolarité obligatoire » (MEN, 2014).

Concernant le lycée, notons d'abord que les notions abordées au collège sont en partie reprises : variable, séquences conditionnelles, boucles. On perd cependant la programmation événementielle, qui n'est pas un passage obligé en Python comme elle peut l'être en Scratch (nous développons ces aspects dans la section 3.3.1). Du côté des ajouts, on peut évoquer le typage des variables et les fonctions. Ajoutons qu'il n'est pas commun pour une discipline de reprendre dans son programme la quasi intégralité des contenus du niveau précédent. Nous avançons l'hypothèse que les changements dans les modalités de programmation (blocs vers texte) nécessitent de réinvestir ces notions. Remarquons également que, comme nous l'avons vu dans la revue de littérature (section 2.1.1), cet effet cumulatif des notions dans les curriculums est plus largement constaté au niveau mondial (Oda et al., 2021). Enfin, à l'instar du cycle 4, nous pouvons observer que les savoirs prescrits sont relativement proches en mathématiques et en SNT. L'introduction de la programmation informatique étant antérieure en mathématiques (2009) par rapport aux SNT (2016), il est probable que les concepteurs des programmes de SNT se soient conformés aux contenus prescrits en mathématiques.

2.6.2 Transposition du savoir savant

Finissons par étudier la transposition du savoir savant, c'est-à-dire le corpus des concepts fondamentaux de la programmation que nous avons établi dans la section 1.5.3, dans le savoir à enseigner, autrement dit les programmes d'enseignement. Le Tableau 2-6 rend compte de la présence de ces concepts fondamentaux dans les programmes des différentes disciplines.

Concepts fondamentaux	Maths C4	Tech. C4	Maths 2nd	SNT 2nd
Variables	✓	✓	✓	✓
Types primitifs : nombres, caractères, booléens			✓	
Structure conditionnelle	✓	✓	✓	✓
Boucle For	✓	✓	✓	✓
Boucle While	✓	✓	✓	✓
Entrées-sorties des programmes				
Fonctions et paramètres			✓	✓
Récursivité				
Type composites simples : chaînes de caractères, tableaux, enregistrements			✓	
Type composites complexes : listes				

Tableau 2-6 – Présence des concepts fondamentaux de la programmation (savoir savant) dans les différents programmes d'enseignement (savoir à enseigner)

Nous pouvons d'abord remarquer que les concepts de variable ainsi que les structures de contrôle classiques (conditionnelle, boucles for et while) font partie des quatre programmes. Ensuite, les programmes du lycée comprennent les notions de fonction et, pour les mathématiques, le typage primitif et composite (seulement les chaînes de caractères). Les notions d'entrées-sorties, de récursivité, ainsi que les types composites simples et complexes ne sont pas abordés au niveau seconde. Nous les retrouvons cependant plus tard au lycée dans les programmes de NSI (MEN, 2019f, 2019g), en première (entrées-sorties, types composites simples) et terminale (récursivité et types composites complexes). Notons qu'à contrario, la notion de programmation évènementielle est présente dans les programmes du cycle 4 mais ne fait pas partie du corpus des concepts fondamentaux. Cela est vraisemblablement dû à l'utilisation du logiciel Scratch, qui en raison du mode de déclenchement de ses programmes, nécessite d'appréhender la notion d'évènement.

2.6.3 Synthèse de la section

Q2.3 : Quels sont les concepts fondamentaux de la programmation informatique mis en jeu dans les programmes de chaque discipline ? Quelle transposition des savoirs savants ?

Après avoir analysé les programmes d'enseignement, nous avons établi les concepts fondamentaux à enseigner dans chacune des quatre disciplines :

- **au cycle 4**, en mathématiques et en technologie, les concepts visés sont les **variables**, les **structures conditionnelles**, les **boucles for et while** et la **programmation évènementielle** ;
- **en seconde** , s'ajoutent le **typage** des variables et les **fonctions** ;

Tous les concepts fondamentaux du savoir savant ne sont pas enseignés au niveau seconde, les entrées-sorties, les types composites et la récursivité apparaissent dans le programme de la spécialité NSI en première et terminale. À l'inverse, la programmation évènementielle est au programme du cycle 4 mais ne fait pas partie des concepts fondamentaux.

Nous venons de présenter les concepts visés par les programmes d'enseignement dans les différentes disciplines, nous analysons dans la section suivante les niches écologiques de ces notions.

2.7 Niches écologiques des concepts de programmation

Cette section traite la question de recherche Q2.4 en rapport avec les niches écologiques des concepts de la programmation. Nous allons d'abord traiter la question sous l'angle quantitatif en nous demandant dans quelle mesure chaque concept est mis en œuvre dans les différents exemples d'activités proposés dans les textes officiels puis, d'un point du vue qualitatif, en interrogant le rôle et l'utilité de ces concepts dans les activités.

Nous rappelons succinctement la méthodologie exposée dans la section 2.3.2. Nous avons modélisé chacune des activités sous la forme de praxéologies informatiques en passant par un découpage récursif en types de tâches. Le résultat de cette subdivision est un arbre de types de tâches dont les éléments terminaux sont des atomes mettant directement en œuvre les concepts de la programmation informatique. Nous avons ensuite classifié ces atomes en lien avec les notions qu'ils implémentent. Nous retrouvons les ensembles de concepts suivants : les variables et types de données, les structures de contrôle, les méthodes de programmation parallèle et évènementielle, les opérations logiques et numériques, et les bibliothèques. Nous avons terminé par un comptage des occurrences d'utilisation des notions et un calcul de proportion pour chaque discipline.

Les sous-sections qui suivent détaillent nos résultats pour chaque ensemble de concepts, nous finissons par un bilan transdisciplinaire.

2.7.1 Variables et types de données

Indiquons, en préambule, une particularité méthodologique concernant les variables. Nous avons, en supplément du comptage des atomes relatifs à cette notion, inventorié le type des variables utilisées dans les programmes. Nous avons donc listé l'usage des variables de type primitif : nombre entier, nombre flottant, booléen ; des variables de type composite : chaîne de caractères, liste et n-uplet ; des variables de

type objet. Précisons également que les activités recommandées dans les ressources d'accompagnement s'appuient, en cohérence avec les prescriptions officielles, sur la programmation par blocs en Scratch au cycle 4 (et ponctuellement en seconde), et sur la programmation textuelle en Python en seconde. Comme nous le verrons plus tard (section 3.3.1), la programmation en Scratch s'apparente à de la programmation orientée objet. Par conséquent, ne nous étonnons pas dans la suite de trouver des références aux concepts de la programmation objet au cycle 4.

Atomes	**Disciplines**			
	Cycle 4		Seconde	
	Maths	Techno	Maths	SNT
Variable : nombre d'occurrences et proportion dans la discipline				
Utiliser une variable (générique)		4 (20%)		
Déclarer une variable	10 (2,5%)			3 (0,5%)
Utiliser la valeur d'une variable	34 (8,5%)		70 (26%)	163 (28%)
Modifier la valeur d'une variable	28 (7%)		49 (18%)	108 (18%)
Total	72 (18%)	4 (20%)	119 (43%)	274 (47%)
Type des variables : nombre de variables utilisées et proportion dans la discipline				
Variable de type entier	19 (79%)		26 (65%)	15 (18%)
Variable de type flottant	1 (4%)		12 (30%)	6 (7%)
Variable de type chaine de caractères				11 (13,5%)
Variable de type booléen			2 (5%)	
Variable de type liste	4 (16,5%)			7 (8,5%)
Variable de type n-uplet				9 (11%)
Variable de type objet				35 (42%)
Total	24		40	83

Tableau 2-7 – Nombre et proportion d'atomes relatifs aux variables

En parcourant le Tableau 2-7, nous pouvons commencer par remarquer que, globalement, les tâches relatives à la manipulation des variables sont essentielles au lycée (43% de l'activité en mathématiques et 47% de l'activité en SNT). Cela est lié aux paradigmes de programmation mis en œuvre. En effet, la programmation impérative s'appuie principalement sur des espaces mémoires indexés par des variables. Au cycle 4, cette activité est moindre (18% en mathématiques et 20% en technologie). Cela s'explique par le fait que la programmation Scratch a pour support principal les lutins qui ne sont pas désignés explicitement par des variables mais qui sont les destinataires implicites de la majorité des instructions[13].

Rentrons maintenant dans le détail des types de tâches liés aux variables. L'atome « déclarer une variable » est utilisé uniquement lors de la programmation en Scratch (10 occurrences en mathématiques au cycle 4 et 3 occurrences en SNT). En Python, il

[13] Contrairement à la programmation orientée objet sous sa forme textuelle où la variable « this » ou « self » désigne l'instance de l'objet en cours.

n'y a pas de déclaration de variable explicite, cela est fait implicitement lors de la première affectation.

Ensuite, en portant notre attention sur le ratio entre les actions de modification et d'utilisation des variables nous remarquons un usage relatif très proche dans les deux disciplines du lycée (autour de 2/3 : 18%/26% en mathématiques et 18%/28% en SNT). Nous ne sommes pas en mesure de dire s'il s'agit d'une constante propre à toute activité de programmation, ou si cela est lié au paradigme de programmation ou au langage Python. Dans tous les cas, malgré la nature très diverse des activités dans ces deux disciplines (calculs, traitements d'image, traitements de données, etc.) ce rapport modification/utilisation des variables semble stable. Ce ratio est plus équilibré en cycle 4 (7%/8,5%), cependant, comme nous l'avons précisé ci-avant, les variables ne rendent pas compte de toute l'activité car les lutins sont adressés implicitement en Scratch.

Portons désormais notre intérêt sur le type des variables utilisées et sur leurs niches écologiques dans les différentes disciplines. Comme nous pouvons le voir dans le Tableau 2-7, au cycle 4, en mathématiques, les scripts sont majoritairement constitués de variables de type entier. Elles servent à la gestion des scores dans les jeux, et à la mémorisation des positions des lutins. Le type flottant est quasi inutilisé car, à priori, plutôt utile aux activités de calcul, qui, comme nous l'avons vu dans la section 2.5.1, ne sont pas privilégiées à ce niveau. Les chaînes de caractères ne sont à aucun moment assignées à des variables car elles ne font pas l'objet de traitements. Elles sont directement données en arguments aux méthodes d'expression des lutins : « dire … » et « penser à … ». Notons l'utilisation, hors programme, des listes, qui n'ont pas vocation à être abordées avant la terminale en NSI. Elles permettent de proposer des projets de jeux plus ambitieux en donnant notamment la possibilité de maintenir un état global des différents lutins dans une même structure de données.

En seconde, on note une différence marquée entre les mathématiques et les SNT. En effet, en mathématiques, les scripts manipulent principalement des variables de type entier et flottant. Ceci s'explique par des activités franchement orientées vers le calcul, et en particulier le calcul approché, ce qui peut éclairer la présence importante de variables flottantes relativement aux autres disciplines. En SNT, les types des variables employées sont beaucoup plus hétérogènes, ce qui semble refléter la grande diversité des activités proposées : traitement de données, traitement d'image, pilotage de robot, etc. Notons une utilisation majoritaire des types composites hors programme que sont les n-uplets, les listes et, des objets (42%+11%+8,5%=61,5%). C'est particulièrement le cas des objets, qui représentent 42% des variables utilisées. Nous avons montré dans la section 2.5.4 que les activités proposées en SNT se doivent d'illustrer les autres domaines du programme en s'inspirant des usages industriels réels. Ces visées rendent nécessaire la manipulation d'objets plus complexes : des fichiers texte, des cartes géographiques numériques, des fichiers image ou des robots. Les scripts Python peuvent interagir avec ces objets à travers l'utilisation de bibliothèques logicielles. La plupart de ces bibliothèques n'ont pas une visée pédagogique mais sont issues du monde industriel ou universitaire. C'est pourquoi leur utilisation met en jeu les concepts et notions de programmation avancés que sont les n-uplets, les listes et les objets.

Le Tableau 2-8 nous donne une vision plus fine de l'utilisation des atomes liés aux types composites plus complexes.

Atomes	Disciplines			
	Cycle 4		Seconde	
	Maths	Techno	Maths	SNT
Type composite liste : nombre d'occurrences et proportion dans la discipline				
Accéder à un élément d'une liste	40 (10%)			9 (1,5%)
Modifier la valeur d'un élément d'une liste	9 (2%)			
Total	49 (12%)			9 (1,5%)
Type composite n-uplet : nombre d'occurrences et proportion dans la discipline				
Accéder à un élément d'un n-uplet				10 (1,5%)
Type objet : nombre d'occurrences et proportion dans la discipline				
Utiliser une méthode sans paramètre sans retour	28 (7%)			1 (0,5%)
Utiliser une méthode avec paramètre sans retour	97 (24%)			66 (11%)
Utiliser une méthode sans paramètre avec retour				21 (3,5%)
Utiliser une méthode avec paramètre avec retour				30 (5%)
Total	125 (31 %)			118 (20%)

Tableau 2-8 – Nombre et proportion d'atomes relatifs aux types composites et objet

Nous retrouvons bien l'utilisation des listes en mathématiques au cycle 4 et en SNT. Les n-uplets connaissent une utilisation marginale en SNT. L'utilisation des méthodes en programmation objet constitue une part importante de l'activité en mathématiques en cycle 4 (31%) et en SNT (20%). Au cycle 4, cela s'explique par les multiples instructions à destinations des lutins : « avancer de … », « tourner de … », « aller a … » « basculer vers le costume … », « dire … », « jouer le son … », « relever stylo », etc. qui sont autant de méthodes de ces objets lutins. Nous pouvons remarquer qu'une grande majorité des méthodes utilisées prennent des arguments en entrée. En SNT, comme nous l'avons avancé précédemment, il s'agit d'utiliser les Interfaces de Programmation Applicative (API) offertes par différentes bibliothèques logicielles à travers un ensemble de classes et de méthodes. L'utilisation majoritaire des méthodes avec paramètres et sans retour illustre bien cette idée d'agir sur des éléments externes tels que des images, des cartes géographiques ou des robots. La sous-section suivante traite des atomes en rapport avec les structures de contrôle.

2.7.2 Structures de contrôle

Nous prenons en compte, dans le Tableau 2-9, les trois structures de contrôle que nous retrouvons dans les programmes : les conditionnelles, les boucles et les fonctions.

Atomes	Disciplines			
	Cycle 4		Seconde	
	Maths	Techno	Maths	SNT
Structure conditionnelle : nombre d'occurrences et proportion dans la discipline				
Utiliser une conditionnelle (générique)		3 (15%)		
Utiliser une conditionnelle à une branche	17 (4%)			2 (0,5%)
Utiliser une conditionnelle à deux branches	4 (1%)		4 (1,5%)	8 (1,5%)

Utiliser une conditionnelle à trois branches				1 (0,5%)
Total	21 (5%)	3 (15%)	4 (1,5%)	11 (2%)
Structure de boucle : nombre d'occurrences et proportion dans la discipline				
Utiliser une boucle (générique)		7 (35%)		
Utiliser une boucle bornée sans var. de boucle	2 (0,5%)		2 (0,5%)	3 (0,5%)
Utiliser une boucle bornée avec var. de boucle			2 (0,5%)	28 (5%)
Utiliser une boucle non-bornée	14 (3,5%)		3 (1%)	8 (1,5%)
Total	16 (4%)	7 (35%)	7 (2,5%)	39 (6,5 %)
Fonction : nombre d'occurrences et proportion dans la discipline				
Utiliser une fonction (générique)		1 (5%)		
Utiliser une fonction sans paramètre sans retour	5 (1%)		1 (0,5%)	
Utiliser une fonction avec paramètre sans retour	8 (2%)		22 (8%)	15 (2,5%)
Utiliser une fonction sans paramètre avec retour				
Utiliser une fonction avec paramètre avec retour	11 (3%)		33 (12%)	27 (4,5%)
Définir une fonction sans paramètre sans retour				
Définir une fonction avec paramètre sans retour	1 (0,5%)		1 (0,5%)	
Définir une fonction sans paramètre avec retour				
Définir une fonction avec paramètre avec retour			11 (4%)	
Total	25 (6%)	1 (5%)	68 (25%)	42 (7%)

Tableau 2-9 - Nombre et proportion d'atomes relatifs aux structures de contrôle

Commençons par mettre en avant le fait que l'utilisation des structures conditionnelles représente une faible partie de l'activité au cycle 4 (5% en mathématiques et 15% en technologie) et qu'elle est réduite à la portion congrue en seconde (1,5% en mathématiques et 2% en SNT). Au sujet des niches écologiques des conditionnelles, au cycle 4, en mathématiques, elles permettent : de détecter des fins de parties en testant des variables de score, de tester des positions particulières des lutins ou de vérifier ponctuellement la survenue d'évènements. Les conditionnelles utilisées comportent le plus souvent une unique branche. En seconde, la conditionnelle est utilisée avec parcimonie. En mathématiques, elle sert, dans les ressources analysées : à vérifier qu'un déterminant est nul, à tester la constructibilité d'un triangle, ou à détecter un succès lors d'une expérience aléatoire. En SNT, elle permet de : filtrer les données issues d'un fichier, détecter la collision d'un robot avec un objet, ou tester la position ou la couleur d'un pixel dans une image. Ici la forme à deux branches est privilégiée.

Les structures de boucles sont globalement plus utilisées que les conditionnelles, et ce d'un facteur deux à trois, excepté en mathématiques au cycle 4. Concernant leurs niches, commençons par la boucle non-bornée. Elle sert principalement à créer des boucles infinies d'attente d'évènements dans les jeux (mathématiques au cycle 4) ou pour des applications physiques (technologie et SNT). Elle sert également à mettre fin à des actions répétitives dont on ne connait pas, à priori, le nombre d'itérations : détecter la fin du parcours de toutes les lignes d'un fichier texte en SNT ou détecter la condition d'arrêt dans l'affinage d'un calcul approché en mathématiques en seconde.

La boucle bornée, lorsqu'elle est utilisée sans la variable de boucle, a pour fonction de : répéter un nombre prédéfini de fois un tracer ou un déplacement en vue de constituer un motif (mathématiques au cycle 4 et SNT), parcourir un certain nombre de lignes d'un fichier texte (SNT), réitérer un certain nombre de fois un tirage aléatoire lors d'un échantillonnage (mathématiques en seconde), réitérer un calcul sur une partie d'un intervalle défini par un certain pas dans une méthode de calcul approché (mathématiques en seconde). Lorsque la variable de boucle est exploitée, il s'agit à chaque fois de parcourir une collection d'éléments indexés : parcours des lignes, des diagonales ou des pixels d'une image numérique (SNT), parcours de listes dans une démarche de dénombrement en probabilité (mathématiques en seconde). Nous remarquons alors dans ces activités de parcours, une forte présence de boucles imbriquées permettant de se déplacer dans plusieurs dimensions.

Enfin, intéressons-nous aux fonctions. Dans les ressources que nous avons analysées, elles sont utilisées de manière modérée (autour de 5% de l'activité) au cycle 4 et en SNT, alors qu'elles représentent 25% de l'activité en mathématiques en seconde. Cela s'explique par le fait que dans cette discipline, la programmation orientée objet est absente, donc toute l'activité passe par les variables de type primitif et l'utilisation des fonctions. Penchons-nous sur les niches écologiques des fonctions selon les disciplines. Rappelons qu'en Scratch, il est possible de créer des fonctions à travers la constitution de blocs personnels. Nous distinguons les méthodes, qui sont les fonctions définies au sein des objets, et les fonctions, qui peuvent être utilisées indépendamment de tout objet. Au cycle 4, il s'agit surtout de scinder le code dans des « sous-programmes ». En mathématiques, un bloc personnel peut être créé par l'enseignant dans le but pédagogique de cacher temporairement une certaine complexité aux élèves qui ne feront qu'utiliser ce bloc sans en comprendre la mise en œuvre. Certains projets, plus complexes, proposent la création et l'utilisation par les élèves de blocs personnels dans un objectif de résolution modulaire de problèmes. Cette démarche de découpage en sous-programme permettant de diminuer la complexité est également mise en avant en technologie. Rappelons que la notion de fonction informatique est hors programme au cycle 4. Elle doit en effet être introduite en classe de seconde. Nous pouvons également constater une autre utilisation, plus rare, des fonctions au cycle 4 : elles permettent de faire appel aux fonctionnalités natives de Scratch non liées aux lutins. Citons, par exemple, le tirage d'un nombre entier aléatoire.

En seconde, nous retrouvons sensiblement les mêmes niches. En mathématiques, il s'agit soit, d'utiliser les fonctions natives de Python (« print(…) » ou « max(…) ») ou celles fournies par des bibliothèques externes (« sqrt(…) »), soit, de modulariser les résolutions de problèmes en créant des sous-programmes dans un découpage reprenant la structure des questions de l'activité. Concernant les SNT, les fonctions servent uniquement à appeler les fonctionnalités natives de Python (« print(…) », « max(…) », « int(…) » ou « float(…) »), ou certaines fonctions des bibliothèques logicielles. La sous-section suivante traite de la programmation parallèle et évènementielle.

2.7.3 Programmation parallèle et évènementielle

Certaines disciplines utilisent des méthodes de programmation non-linéaire. Nous avons regroupé, dans le Tableau 2-10, les données concernant les atomes mettant en œuvre ces notions.

Atomes	Disciplines			
	Cycle 4		Seconde	
	Maths	Techno	Maths	SNT
Programmation parallèle : nombre d'occurrences et proportion dans la discipline				
Envoyer un message	6 (1,5%)			
Attendre la réception d'un message	5 (1%)			
Total	11 (3%)			
Programmation événementielle : nombre d'occurrences et proportion dans la discipline				
Déclencher une action suite à un évènement (générique)		3 (15%)		
Attendre la survenue d'un évènement	26 (6,5%)			16 (2,5%)
Tester la survenue d'un évènement	5 (1%)			
Total	31 (8%)	3 (15%)		16 (2,5%)

Tableau 2-10 - Nombre et proportion d'atomes relatifs aux méthodes de programmation

La programmation parallèle (ou concurrente) est une méthode de programmation fréquemment mise en œuvre dans le logiciel Scratch. En effet, les projets Scratch s'appuient communément sur plusieurs lutins. Or, les scripts de ces lutins ne peuvent se lancer que suite à la survenue d'un évènement, et rien n'empêche le déclenchement de plusieurs scripts par un même évènement (typiquement « drapeau vert cliqué »). Par conséquent, ces scripts ont la possibilité de s'exécuter en même temps, c'est-à-dire, en parallèle. En informatique, la programmation parallèle pose classiquement des problèmes de synchronisation et d'accès concurrents aux ressources partagées. Afin d'organiser cette synchronisation, Scratch propose un système d'envoi et de réception de messages entre les scripts. Dans les exemples dont nous disposons, ces atomes ne sont utilisés qu'en mathématiques au cycle 4. Et ce uniquement dans les scripts des projets les plus complexes faisant intervenir un grand nombre de lutins. La notion de programmation parallèle est immanente à la programmation Scratch, elle n'est cependant mentionnée dans aucun programme du cycle 4. Néanmoins, la ressource d'accompagnement R1M4 en dit quelques mots :

> On introduit ici la notion de parallélisme, c'est-à-dire d'exécution simultanée de deux scripts différents. Le terme « parallélisme » n'est d'ailleurs pas une connaissance attendue des élèves, c'est l'utilisation de scripts parallèles qui figure seule parmi les objectifs de formation. Bien sûr, tout développement théorique sur le parallélisme est à proscrire. (MEN, 2016c, p. 8)

Il s'agit donc de faire prendre conscience aux élèves que plusieurs scripts peuvent se jouer en même temps, tout en évitant toute institutionnalisation de la notion de parallélisme. L'auteur poursuit : « il est sans doute préférable de n'évoquer le parallélisme que dans des situations où les différents scripts amenés à se dérouler en parallèle sont déclenchés par des événements différents ». Cette précaution doit permettre d'éviter les problèmes d'accès concurrents aux variables globales. On est donc bien loin de l'idée de synchroniser des scripts à l'aide de messages. La notion de programmation parallèle n'apparait pas non plus dans les programmes de NSI au lycée, elle ne sera abordée qu'en deuxième cycle universitaire. En cohérence avec les

programmes, les exemples tirés des ressources d'accompagnement de seconde ne font à aucun moment usage de ces méthodes. Et ce malgré le fait qu'il soit techniquement possible de mettre en œuvre la programmation concurrente dans une utilisation avancée du langage Python.

La méthode de programmation évènementielle est, quant à elle, inévitablement liée à l'utilisation de Scratch. Et ce pour une raison ontologique : les scripts ne peuvent se lancer que sur la survenue d'évènements. De fait, dès lors qu'un programme est implémenté en Scratch (au cycle 4 et en SNT), on retrouve, au minimum, la présence de l'atome « attendre la survenue d'un évènement ». Analysons plus précisément les conditions d'émission et de réception de ces évènements. D'abord, quelles sont les circonstances qui déclenchent un évènement ? Dans les exemples que nous avons recueillis dans les ressources d'accompagnement, il s'agit : d'un clic sur le drapeau vert, d'appuis sur les touches du clavier, de lutins qui rentrent en collision, ou des conséquences de l'actionnement des capteurs d'un robot. Ensuite, comment les programmes interagissent avec ces évènements ? On retrouve deux manières de procéder : en bloquant l'exécution d'un script dans l'attente d'un évènement particulier (atome « attendre la survenue d'un évènement ») ou en testant ponctuellement, au fil de l'exécution, la survenue d'un évènement précis (atome « tester la survenue d'un évènement »). Remarquons que, dans les exemples fournis par l'institution, les atomes « tester la survenue d'un évènement » sont systématiquement utilisés en combinaison avec une boucle non-bornée « infinie » et une structure conditionnelle. Comme illustré dans le deuxième programme de la Figure 2-12, cette configuration provoque un comportement strictement identique au premier programme, c'est à dire à l'utilisation seule de l'atome « attendre la survenue d'un évènement ». Dès lors, d'un point de vue fonctionnel, la distinction entre ces deux atomes ne fait pas sens ici.

Figure 2-12 – Exemples d'utilisation des atomes liés aux évènements

Précisons que la notion de programmation évènementielle fait partie des attendus du programme au cycle 4, aussi bien en mathématiques qu'en technologie et qu'en accord avec les programmes de seconde, elle n'y est pas employée (excepté en SNT, dans les quelques scripts écrits en Scratch). Passons à l'étude des atomes ayant trait aux calculs et à la logique.

2.7.4 Opérations logiques et numériques

L'emploi des structures conditionnelles et des boucles non-bornées nécessite l'utilisation d'expressions logiques afin d'expliciter les conditions d'entrée dans les différentes branches ou la condition d'arrêt des itérations. Ces conditions peuvent se

rapporter à la survenue d'évènements, comme nous l'avons montré dans la sous-section précédente, mais elles peuvent également porter sur des comparaisons de variables. Les expressions logiques ainsi constituées sont susceptibles d'être articulées entre-elles à l'aide des opérateurs de conjonction logique (et) et de disjonction logique (ou).

Atomes	**Disciplines**			
	Cycle 4		Seconde	
	Maths	Techno	Maths	SNT
Expression logique : nombre d'occurrences et proportion dans la discipline				
Tester une égalité sur des nombres	11 (3%)		3 (1%)	6 (1%)
Tester une égalité sur des chaines de caractères				6 (1%)
Tester une inégalité sur des nombres	10 (2,5%)		5 (2%)	4 (0,5%)
Utiliser un connecteur logique (générique)		2 (10%)		
Utiliser le connecteur logique « et »	2 (0,5%)			1 (0,5%)
Utiliser le connecteur logique « ou »	1 (0,5%)			
Total	24 (6 %)	2 (10%)	8 (3%)	17 (3%)
Calcul : nombre d'occurrences et proportion dans la discipline				
Additionner des nombres	16 (4%)		17 (6%)	7 (1%)
Soustraire des nombres	8 (2%)		23 (8,5%)	10 (1,5%)
Multiplier des nombres	2 (0,5%)		9 (3,5%)	7 (1%)
Diviser des nombres			6 (2%)	1 (0,5%)
Élever des nombres au carré			7 (2,5%)	
Élever des nombres à la puissance n			4 (1,5%)	
Total	26 (6,5%)		66 (24%)	25 (4%)

Tableau 2-11 - Nombre et proportion d'atomes relatifs aux opérations logiques et numériques

Dans les exemples que nous avons à disposition, dont l'analyse figure dans le Tableau 2-11, nous pouvons remarquer que les tests d'égalités et d'inégalités sont appliqués sur des nombres dans les disciplines mathématiques, et sur des nombres et des chaînes de caractères en SNT. Encore une fois, cette situation semble s'expliquer par la nature des activités dans chaque discipline. Les connecteurs logiques sont maniés, à de rares occasions, au cycle 4 ainsi qu'en SNT. Ils permettent d'exprimer des conditions plus complexes et de simplifier des structures conditionnelles à plusieurs branches. Les comparaisons (égalité et inégalité) ne sont signalées dans les programmes d'enseignement d'aucune des trois disciplines. Nous ne les considérons pas comme hors programme, car elles ne sont pas l'objet des programmes des années supérieures. Nous dirons que ce sont des implicites des programmes. En effet, ces notions d'égalité et d'inégalité sont connues des élèves en mathématiques dès le cycle 3, cependant, leur traduction dans un langage de programmation (évoquons par exemple l'instruction « == » en Python) et leur transposition sur le domaine des chaînes de caractères constituent un véritable enjeu pour les apprenants. Les connecteurs logiques sont, quant à eux, hors programme au cycle 4 comme en seconde. Ils font en effet partie des notions abordées en classe de première en NSI.

Les variables peuvent donc faire l'objet de comparaisons, celles de type nombre (entier et flottant) peuvent également constituer les opérandes d'opérations numériques, c'est à dire des opérations mathématiques élémentaires. Ainsi, dans le Tableau 2-11, nous retrouvons, sans surprise, une prépondérance de ces atomes de calcul numérique en mathématiques seconde où ils représentent 24% de l'activité globale. En effet, nous savons que dans cette discipline le travail est fortement lié à l'activité mathématique. Les autres disciplines utilisent, à l'occasion, les atomes liés aux quatre opérations mathématiques de base. Ces atomes de calcul numérique, ne sont pas évoqués dans les programmes d'enseignement. En effet, il est légitime de considérer les opérations élémentaires de calcul comme acquises, mais ici aussi, leur traduction dans les langages de programmation constitue un enjeu pour les élèves. Nous pensons par exemple à la multiplication notée « * » en Python, aux différentes façons de noter les divisions entières (« // ») ou décimales (« / »), ou à la notation « ** » pour désigner les puissances dans ce même langage. Nous avons donc classifié ces atomes comme implicites. La sous-section suivante traite de l'utilisation des bibliothèques.

2.7.5 Bibliothèques

Comme nous l'avons indiqué précédemment, certaines activités nécessitent de faire appel à des bibliothèques logicielles afin d'augmenter les possibilités offertes nativement par le langage Python. Avant de pouvoir utiliser les fonctions, instancier les classes et appliquer les méthodes proposées par ces bibliothèques, il est indispensable de procéder à leur importation. Nous avons regroupé dans le Tableau 2-12 les statistiques d'utilisation de l'atome « Importer une bibliothèque ».

Atomes	**Disciplines**			
	Cycle 4		Seconde	
	Maths	Techno	Maths	SNT
Bibliothèque : nombre d'occurrences et proportion dans la discipline				
Importer une bibliothèque			2 (0,5%)	23 (4%)

Tableau 2-12 - Nombre et proportion d'atomes relatifs aux bibliothèques

Nous avons déjà montré que les activités prescrites en SNT étaient grandement consommatrices des fonctionnalités de bibliothèques logicielles. Il est donc logique de retrouver un forte utilisation de l'atome d'importation. En mathématiques en seconde, la fonction de calcul racine carrée et les fonctions de tracé de courbes rendent nécessaire l'importation de deux bibliothèques. L'importation de bibliothèques est hors programme, l'utilisation des bibliothèques relevant des contenus de la partie « langages et programmation » du programme scolaire de NSI en première.

Résumons maintenant les principaux résultats de cette partie en établissant son bilan.

2.7.6 Bilan transdisciplinaire par concept

Nous venons de présenter les niches des concepts de programmation dans les exemples d'activités recommandées par les ressources d'accompagnement. Nous avons résumé ces éléments quantitatifs et qualitatifs dans le Tableau 2-13. Dans ce tableau, nous représentons les aspects quantitatifs à l'aide du signe « + » dans les

différentes cases. Le nombre de « + » peut aller de un à trois. Il est proportionnel à l'usage qui est fait du concept dans les activités proposées dans les documents d'accompagnement de la discipline. Concernant les aspects qualitatifs, nous avons synthétisé en quelques phrases les niches des concepts pour chaque discipline. En raison du peu d'éléments significatifs dont nous disposons pour la discipline technologie, nous avons choisi de ne pas la faire apparaitre.

	Concepts	Mathématiques cycle 4	Mathématiques seconde	SNT seconde
Variables	Types primitifs	+++ Mémoriser le score des jeux, la position d'un lutin dans une grille	+++ Calculs dans tous les domaines du programme : géométrie, fonctions, stats, etc..	++ Mémoriser : numéro ligne, état robot, couleur pixel, coordonnées sur une carte
	Types composites	+ Jeux plus complexes : mémoriser la position plusieurs lutins dans une grille	∅	+ Utilisation de bibliothèques logicielles : fichiers, carte, images, objets connectés
	Type objet	+++ Manipulation des lutins : se déplacer, changer d'apparence, jouer un son, etc.	∅	+++ Utilisation de bibliothèques logicielles : fichiers, carte, images, objets connectés
Structures de contrôle	Conditionnelle	+ Tester la fin de partie dans un jeu Tester la position d'un lutin	+ Tester si un déterminant est nul Tester la constructibilité triangle Tester le succès d'une expérience aléatoire	+ Filtrer des données issues d'un fichier Détecter des collisions robot-objet Tester la position/couleur d'un pixel
	Boucle for	+ Répéter un parcours ou un tracé	+ Réitérer un tirage aléatoire ou un calcul Parcourir un collection indexée (listes)	++ Parcourir un collection indexée (lignes d'un fichier, pixels d'une image)
	Boucle while	++ Boucle d'attente d'un évènement dans un jeu	+ Détecter la condition d'arrêt dans le calcul d'une valeur approchée	+ Détecter la fin du parcours des lignes d'un fichier
	Fonction	+Appeler fonctions natives (tirage aléatoire) Scinder les scripts ou sous-scripts Cacher la complexité	++ Appeler les fonctions natives (print, max, etc.) et celles des bibliothèques (sqrt) Découpage en sous-programmes	+Appeler les fonctions natives (print, max, etc.) Appeler les fonctions des bibliothèques
Mét.	Prog. parallèle	+ Projet complexes avec beaucoup de lutins	∅	∅
	Prog. événement.	+++ Lancement des scripts Scratch	∅	+ Lancement des scripts Scratch
Opérations	Tests	++ Expliciter : les conditions d'entrée dans les branches conditionnelles et les sorties de boucle while	+ Expliciter : les conditions d'entrée dans les branches conditionnelles et les sorties de boucle while	+ Expliciter : les conditions d'entrée dans les branches conditionnelles et les sorties de boucle while
	Connec. logiques	+ Exprimer des conditions complexes	∅	+ Simplifier des conditionnelles
	Calculs	+ Calculer position et orientation d'un lutin Calculer des scores de jeu	+++ Activités mathématiques diverses	+ Calculer la couleur d'un pixel Calculer la trajectoire d'un robot
Bibliothèques		∅	+ Importation de fonctions de calcul et de tracé de courbes	+++ Importation de nombreuses fonctions en rapport avec les : fichiers, cartes, images, etc.

Tableau 2-13 – Tableau récapitulatif des niches des concepts de la programmation par discipline

Concernant les variables et les types de données, nous pouvons d'abord affirmer qu'une grande partie de l'activité tourne autour de ces concepts. Cela est dû au paradigme de programmation impératif utilisé dans toutes les disciplines. Nous résumons ensuite les grandes lignes des niches écologiques des variables selon leur type. Les types primitifs servent de manière utilitaire aux activités prescrites dans chaque discipline : jeux au cycle 4, et au service des autres domaines du programme au lycée. Concernant les types composites, ils sont ponctuellement utilisés au cycle 4 dans la mise en œuvre de projets plus ambitieux, et en SNT pour l'utilisation de nombreuses bibliothèques logicielles issues de la sphère extra-scolaire. Enfin, les variables de type objet permettent d'accéder implicitement aux lutins manipulables dans Scratch au cycle 4 et d'accéder aux fonctionnalités des bibliothèques en SNT.

Les structures de contrôle ne représentent, en terme de volume, qu'une faible partie des atomes utilisés dans les activités, en particulier au lycée. Au niveau des niches écologiques, les conditionnelles sont utilisées au cycle 4, majoritairement à une branche, dans les jeux pour tester les fins de partie, ou les positions particulières des lutins. En seconde, la forme à deux branches est privilégiée et sert de façon transversale dans tout type d'activités. Au sujet des boucles, les boucles for ont essentiellement pour rôle de réitérer des traitements en mathématiques et de parcourir des collections indexées au lycée. Les boucles while servent principalement au cycle 4 à attendre indéfiniment un évènement (boucle « infinie ») et au lycée à détecter la fin d'un processus itératif. Enfin, les fonctions sont utilisées à deux fins. D'abord, dans toutes les disciplines, pour accéder aux fonctionnalités natives des langages, et en seconde, aux services fournis par des bibliothèques externes. Ensuite, en mathématiques et particulièrement en seconde, pour résoudre de façon modulaire les problèmes posés.

Au sujet des méthodes de programmation parallèle et évènementielle, la programmation parallèle fait intrinsèquement partie de la programmation Scratch dans la mesure où les scripts ont la possibilité d'être exécutés concomitamment. Elle sert sporadiquement dans des activités complexes nécessitant plusieurs lutins. La méthode de programmation évènementielle est, quant-à-elle, un passage obligé de tout script Scratch. Elle permet le lancement des scripts et l'interfaçage avec des éléments externes : souris, clavier, capteur, autre lutins. Nous la retrouvons donc beaucoup au cycle 4 et de façon marginale en SNT.

Ensuite, concernant les opérations logiques et numériques, l'emploi des structures de contrôle, en particulier des conditionnelles et des boucles while, nécessite l'explicitation d'expressions logiques. L'utilisation des opérations de test est donc liée à celle de ces structures de contrôle. Les connecteurs logiques sont utilisés, à de rares occasions, pour exprimer des conditions complexes ou simplifier des conditionnelles à plusieurs branches. Enfin, les opérations de calcul constituent l'une des activités principales de la discipline mathématiques en seconde. Elles sont également utilisées, ponctuellement, de façon utilitaire dans les autres disciplines.

Pour terminer, comme nous l'avons déjà indiqué, l'importation et l'utilisation des bibliothèques sont fondamentales en SNT pour pouvoir proposer des activités dans tous les domaines du programme. Elles servent également marginalement en mathématiques en seconde afin d'accéder à des modules de calcul ou de tracé.

2.7.7 Synthèse de la section

Q2.4 : Quelles sont les niches des concepts de la programmation dans les tâches recommandées par les ressources d'accompagnement ?

En analysant les exemples d'activités recommandés dans les ressources d'accompagnement émises par le ministère de l'Éducation nationale nous avons pu établir les niches écologiques des concepts de la programmation informatique, nous pouvons résumer les résultats comme suit :

- concernant les **variables**, les types primitifs servent de façon utilitaire dans les activités prescrites alors que les types composites et les objets permettent d'accéder aux fonctionnalités des bibliothèques externes ;
- concernant les **structures de contrôle**, les conditionnelles permettent d'effectuer des traitements contingents dans tous types d'activités, les boucles for ont pour rôle de réitérer des traitements et de parcourir des collections indexées, les boucles while permettent de détecter la fin d'un traitement itératif ou de faire une « boucle infinie » d'attente d'un évènement ;
- concernant les **méthodes des programmation parallèle et événementielle**, elles sont intimement liées à l'utilisation de la programmation en Scratch ;
- concernant les **opérations logiques et numériques**, les tests et les connecteurs logiques permettent d'expliciter les conditions dans les structures de contrôle, et les opérations numériques sont surtout au service des activités mathématiques ;
- concernant les **bibliothèques,** elles offrent des services supplémentaires exploités au lycée en mathématiques et dans tous les domaines des SNT.

Dans cette section, nous avons détaillé les niches des concepts de la programmation informatique, nous présentons dans la suivante les modalités d'enseignement de ces même notions.

2.8 Indications concernant les modalités d'enseignement

Cette partie a vocation à répondre à la question de recherche Q2.5 qui porte sur les modalités d'enseignement décrites dans les programmes d'enseignement et les ressources d'accompagnement. Les questions auxquelles nous allons répondre sont les suivantes : comment les concepts fondamentaux de la programmation doivent-ils être enseignés ? Quels types d'activités les enseignants sont-ils susceptibles de proposer aux élèves ? Quels dispositifs pédagogiques peuvent-ils mettre en œuvre ? Quelles sont les indications concernant les artefacts à mettre à disposition des élèves ?

En suivant la méthodologie décrite dans la section 2.3.2, nous avons analysé notre corpus de textes officiels à travers une modélisation praxéologique sous forme d'organisations didactiques. Ces praxéologies ont ensuite été compilées en Synthèses

d'Organisations Didactiques (SOD) puis classifiées selon les quatre indicateurs (contenu, activité, dispositif ou artefact), puis huit sous-indicateurs (démarche, type d'activité, mode de programmation, lieu, organisation de l'espace, interaction, artefact-support, artefact-cible). Nous exposons à la suite les résultats de ces analyses par discipline et par indicateur. Nous finissons par un bilan transdisciplinaire.

2.8.1 Mathématiques au cycle 4

Nous présentons dans les sous-sections suivantes les SOD issues des textes officiels pour les mathématiques au cycle 4. Elles sont classées dans quatre tableaux en fonction de l'indicateur qui les caractérise.

Contenus pour les mathématiques au cycle 4

Commençons par nous intéresser aux contenus, le Tableau 2-14 regroupe les SOD en lien avec cet indicateur.

N°	Synthèses d'organisations didactiques (SOD)
1	Présenter les notions d'algorithme et de programme : ne pas distinguer ces deux notions
2	Ne pas viser une connaissance experte et exhaustive dans un langage particulier
3	Fixer clairement et explicitement les objectifs de formation et les nouveaux concepts ciblés dans les activités : chaque séance doit viser des objectifs de formation clairs et explicites ; le professeur fixe clairement les objectifs de formation et les concepts nouveaux qui devront être installés dans une activité ; définir par avance les objectifs de formation
4	Mettre « naturellement » en jeu les notions dans les activités : introduire les variables pour la gestion des scores dans les jeux, introduire les échanges de messages dans des jeux à plusieurs lutins
5	Introduire des notions lorsque les élèves en ressentent le besoin : le professeur propose les blocs de boucle aux seuls élèves qui en font la demande
6	Présenter les variables de façon métaphorique : une variable est présentée comme une étiquette collée sur une boite qui contient des valeurs différentes au cours de l'exécution du programme ; l'exécution séquentielle d'un programme est présentée comme une suite temporelle de configurations des boites et de leurs contenus

Tableau 2-14 – SOD en rapport avec l'indicateur contenu pour la discipline mathématiques au cycle 4

Les SOD 1 et 2 portent sur les notions d'algorithme et de programme. Précisons d'abord que le programme d'enseignement PM4 cible ces deux notions dans la partie relative aux connaissances : « Notions d'algorithme et de programme » (MEN, 2020b, p. 136). Il s'agit cependant de ne pas les distinguer formellement, la ressource d'accompagnement R1M4 est catégorique sur ce point : « il n'est pas question de formaliser d'aucune manière la distinction algorithme/programme » (MEN, 2016c, p. 3). Rappelons que, dans le savoir savant, un algorithme désigne une procédure systématique de résolution de problème, et qu'un programme qualifie une suite d'instructions à destination des machines ayant pour but de décrire un algorithme (voir section 1.4.1). Dans l'institution que constituent les mathématiques au cycle 4, ces deux notions ont donc une signification équivalente qui est celle d'un programme. De plus, la prégnance du terme « algorithme » est corroborée par des éléments quantitatifs. Ainsi, dans les textes que nous avons analysés, le mot « algorithme » comporte 6 occurrences contre 47 pour le mot « programme ». Nous pouvons en conclure que le terme « algorithme » est largement sous-utilisé (6/53=11%). Notons également que

nous rencontrons régulièrement le terme « script » dans la ressource d'accompagnement R1M4 (57 occurrences) en tant que désignation alternative d'un programme.

Ajoutons que, comme l'indique la SOD 2, il n'est pas attendu des élèves qu'ils acquièrent des connaissances expertes dans un langage de programmation : « Au cycle 4, les élèves s'initient à la programmation [...] sans viser une connaissance experte et exhaustive d'un langage ou d'un logiciel particulier » (MEN, 2020b, p. 136).

Les SOD 3, 4 et 5 concernent la manière de mettre en jeu les savoirs dans les activités. D'abord, ces activités doivent avoir des objectifs de formation clairs, notamment en ce qui concerne les connaissances : « le professeur propose une activité, dont il a fixé clairement les objectifs de formation et les concepts nouveaux qui devront être installés » (MEN, 2016c, p. 4). Ensuite, ces activités sont tenues de mettre « naturellement » en jeu ces notions. Ainsi, on peut lire dans la ressource d'accompagnement R1M4 : « on rencontre [dans une activité de développement de jeu] également naturellement la notion de variable [pour la gestion du score] » (MEN, 2016c, p. 15), ou « un clic sur une pièce doit entraîner une action sur sa voisine : c'est une occasion naturelle d'introduire la notion de message entre lutins » (MEN, 2016c, p. 17). Enfin, les élèves peuvent utiliser une notion lorsqu'ils la jugent utile à la résolution de la situation d'enseignement. C'est l'objet de la SOD 4 : « L'objectif n'est pas que les élèves utilisent nécessairement un bloc de répétition avant qu'ils en ressentent eux-mêmes l'utilité [...] et le professeur propose ces blocs aux seuls élèves qui en font la demande » (MEN, 2016c, p. 13). Nous retrouvons dans cette manière « naturelle » et « à la demande » de mettre en jeu les connaissances certains principes du modèle des situations adidactiques de Brousseau (1986b) (voir section 5.3).

Pour finir, la SOD 6 concerne une recommandation particulière liée au concept de variable. En effet, elle préconise d'introduire cette notion, de façon métaphorique, en la comparant à une étiquette apposée sur une boite qui peut contenir des valeurs différentes au cours de l'exécution du programme.

Activités pour les mathématiques au cycle 4

Le Tableau 2-15 présente les SOD liées à l'indicateur activité.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
7	Démarche	Faire travailler les élèves dans une démarche de projet : dans le cadre d'un projet suivi sur quelques séances
8	Démarche	Faire résoudre des problèmes à travers le développement de programmes simples
9	Démarche	Exposer une situation problème en début de séance : une séance peut commencer par quelques minutes où le professeur expose une situation-problème qui introduit la notion visée ; au cours d'une première séance le professeur propose une activité ; très courte introduction de présentation d'un concept ou d'un problème
10	Démarche	Laisser les élèves résoudre le problème en développant un programme dans les directions qu'ils choisissent
11	Démarche	Proposer une phase d'institutionnalisation : le professeur peut proposer une mise en commun des concepts et des techniques utilisées par les uns et les autres ; la séance peut se terminer par la récapitulation des instructions éventuellement sous forme d'une fiche ; phase finale d'institutionnalisation des nouveaux concepts, de mise au propre et par écrit des notions rencontrées

12	Démarche	Valoriser les activités des élèves : mise en ligne ; exposition ; atelier
13	Type activité	Laisser les élèves écrire leur propre programme : les élèves doivent être amenés, le plus souvent possible, à écrire eux-mêmes leurs propres programmes
14	Type activité	Fournir occasionnellement des programmes à compléter, améliorer ou corriger (si pas le temps ou si les élèves ne sont pas en mesure de les programmer) : trame de programme ; bloc personnel
15	Mode prog.	Proposer un mode de programmation par tâtonnement à l'aide de boucles courtes de rétroaction sous forme d'essais-erreurs : observation immédiate de l'effet d'un script ou d'un bloc sur les lutins, c'est l'élève lui-même qui constate s'il s'est trompé ou pas et qui est invité à essayer une autre solution en cas d'échec
16	Mode prog.	Ne pas insister sur une chronologie précise pour les élèves lors de la programmation : analyse d'un problème, conception d'un algorithme puis programmation effective, ces phases avancent en parallèle par essais-erreurs

Tableau 2-15 – SOD en rapport avec l'indicateur activité pour la discipline mathématiques au cycle 4

Commençons par décrire la démarche d'enseignement. La ressource d'accompagnement R1M4 précise tout d'abord ce qu'il est déconseillé de faire : « une séance d'apprentissage de l'algorithmique et de la programmation ne saurait se dérouler sous forme d'un cours descendant, magistral, où les élèves resteraient passifs » (MEN, 2016c, p. 3). Les SOD 7 et 8 indiquent la direction à suivre : il s'agit de proposer une démarche de mini-projets s'appuyant sur la résolution de problèmes à travers le développement de programmes simples : « L'enseignement de l'algorithmique et de la programmation se prête particulièrement à la pédagogie de projet. L'objectif est de permettre à chaque élève de construire des connaissances et des compétences [...] dans le cadre d'un projet suivi sur quelques séances » (MEN, 2016c, p. 4) ; « Écrire, mettre au point (tester, corriger) et exécuter un programme en réponse à un problème donné » (MEN, 2020b, p. 136), et « Au cycle 4, les élèves s'initient à la programmation, en développant dans une démarche de projet quelques programmes simple » (MEN, 2020b, p. 136). La ressource d'accompagnement R1M4 propose une chronologie précise en quatre phases permettant de mettre en œuvre cette démarche, c'est l'objet des SOD 9 à 12. Dans une première phase, l'enseignant peut proposer une situation-problème qui met en jeu les notions qu'il cible : « Une séance peut commencer par quelques minutes où le professeur expose une situation-problème qui introduit la notion visée » (MEN, 2016c, p. 4). S'ensuit une phase durant laquelle les élèves sont en activité de façon autonome : « une deuxième séance peut permettre à chaque élève de développer son programme dans les directions qu'il aura lui-même choisies » (MEN, 2016c, p. 4). La troisième phase est celle de la mise en commun : « Le professeur peut aussi proposer une mise en commun des concepts et des techniques utilisés par les uns et les autres » (MEN, 2016c, p. 4), cette phase est aussi nommée « institutionnalisation des nouveaux concepts » (MEN, 2016c, p. 5). Il est également possible, lors d'une ultime étape, de valoriser les travaux des élèves à travers différentes activités : « une mise en ligne de certains projets, sans distinction de niveau d'expertise, peut être envisagée afin de mettre en valeur les travaux des élèves. On peut également valoriser ces travaux par des ateliers-jeux, des expositions… » (MEN, 2016c, p. 4).

Ensuite, les SOD 13 et 14 traitent du type des activités. Les élèves doivent autant que possible, développer des programmes ex nihilo : « Ceux-ci [les élèves] doivent être amenés, le plus souvent possible, à écrire eux-mêmes leurs propres programmes » (MEN, 2016c, p. 6). Il est néanmoins envisageable de fournir occasionnellement des programmes à compléter : « il est possible de préparer la séance en proposant aux élèves […] une "trame" de programme, qu'ils seront amenés à compléter » (MEN, 2016c, p. 5), et également de « proposer aux élèves un programme incomplet qu'ils seraient chargés de compléter, d'améliorer ou de corriger » (MEN, 2016c, p. 5). Ce type d'activités doit être privilégié dans l'éventualité où les élèves n'auraient pas le temps ou les connaissances suffisantes pour s'engager dans une activité : « Cette trame doit être limitée à ce que les élèves n'auraient pas le temps de préparer, ou aux objets qu'ils ne sont pas encore en mesure de programmer eux-mêmes » (MEN, 2016c, p. 5).

Décrivons maintenant le mode de programmation préconisé pour les élèves, c'est-à-dire la manière dont ils doivent procéder pour construire le programme qui résoudra le problème posé. Les élèves sont encouragés à travailler par tâtonnement, à l'aide de boucles courtes de rétroactions prenant la forme d'essais-erreurs (SOD 15) : « la mise au point comprend la phase d'essais-erreurs où l'élève construit petit à petit un programme qui répond au problème » (MEN, 2016c, p. 3). De plus, il n'est pas attendu qu'ils respectent une chronologie particulière dans la conception de leur programme (SOD 16) : « il ne faut pas nécessairement insister sur une chronologie précise, ces trois phases [analyse du problème, conception et programmation d'un algorithme] avançant souvent en parallèle » (MEN, 2016c, p. 3).

Dispositifs pour les mathématiques au cycle 4

Penchons-nous désormais sur les dispositifs. Nous avons compilé dans le Tableau 2-16 les SOD en rapport avec cet indicateur.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
17	Lieu	Faire travailler les élèves en salle de classe, en salle informatique ou dans la cour de récréation
18	Orga. espace Interaction	Faire travailler les élèves en groupe ou individuellement
19	Interaction	Intervenir le moins possible : faire référence à un autre programme pouvant aider l'élève ; proposer des éléments en fonction des besoins de l'élève ; proposer de nouvelles pistes pour résoudre une difficulté
20	Interaction	Intervenir oralement et ne pas manipuler la machine de l'élève : ne pas taper au clavier ou manipuler la souris de l'élève ; se contenter de répondre oralement aux questions des élèves

Tableau 2-16 – SOD en rapport avec l'indicateur dispositif pour la discipline mathématiques au cycle 4

La SOD 17 évoque les lieux de travail, il s'agit d'investir la salle de classe ou la salle informatique : « Les modalités de l'apprentissage correspondant peuvent être variées : […] en salle informatique ou en salle banale » (MEN, 2016c, p. 2). De façon marginale, la ressource d'accompagnement R1M4 propose d'investir la cour de récréation pour des activités d'informatique débranchée.

L'organisation de l'espace et les interactions entre individus sont très sommairement décrits dans la SOD18. Il est précisé que les élèves peuvent travailler

en groupe ou individuellement : « Les modalités de l'apprentissage correspondant peuvent être variées : [...] individuel ou en groupe » (MEN, 2016c, p. 2). Le travail de groupe sous-entend une certaine collaboration entre les élèves.

En revanche, le rôle du professeur est codifié assez précisément dans les SOD 19 et 20. D'abord, la ressource d'accompagnement R1M4 précise qu'il doit intervenir au minimum durant les phases de travail en autonomie : « cela signifie qu'il accepte pour ses élèves la nécessité de phases de tâtonnement et d'essais-erreurs, durant lesquelles il intervient le moins possible » (MEN, 2016c, p. 5). Ensuite, lors de ces interventions, il est invité à faire référence à d'autres programmes pouvant aider les élèves, à proposer des éléments en fonction des besoins des élèves, ou à soumettre de nouvelles pistes pour résoudre une difficulté : « il est recommandé [...] de lui rappeler [à l'élève] éventuellement tel ou tel script déjà écrit qui pourrait l'inspirer ou encore de lui suggérer de nouvelles pistes pour résoudre la difficulté qu'il rencontre » (MEN, 2016c, p. 5), et « la séance permet au professeur d'outiller l'élève selon ses besoins pour faire aboutir son projet. Il peut aussi aider l'élève à le redéfinir si la piste sur laquelle il s'est engagé est inadaptée » (MEN, 2016c, p. 4). Enfin, il ne doit pas intervenir directement sur les machines mais uniquement en répondant à l'oral aux questions des élèves : « il est recommandé de résister à l'envie de taper sur le clavier ou de manipuler la souris à la place de l'élève » (MEN, 2016c, p. 5), et « il est recommandé de se contenter de répondre oralement aux questions de l'élève » (MEN, 2016c, p. 5).

Artefacts pour les mathématiques au cycle 4

Pour terminer, les SOD en lien avec l'indicateur artefact sont listées dans le Tableau 2-17.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
21	Support	Ne pas faire utiliser de langage de description d'algorithmes (pseudo-langage ou organigramme) ; faire utiliser le logiciel de programmation par blocs Scratch
22	Support	Faire travailler les élèves sur tablette ou ordinateur
23	Support Cible	Proposer des activités débranchées : simuler physiquement les déplacements des lutins sur un quadrillage

Tableau 2-17 – SOD en rapport avec l'indicateur artefact pour la discipline mathématiques au cycle 4

Les SOD 21, 22 et 23 donnent des indications au sujet des éléments matériels, logiciels et symboliques en lien avec l'activité de programmation. Concernant le formalisme des programmes/algorithmes, la ressource d'accompagnement R1M4 proscrit l'utilisation d'un langage de description d'algorithmes : « il ne s'agit pas de définir un langage de description d'algorithme (ni pseudo-langage ni organigramme) » (MEN, 2016c, p. 3). Les algorithmes, les programmes et les scripts se doivent d'être formulés directement dans un langage de programmation graphique sous la forme de blocs. L'artefact-support choisi pour développer les programmes est le langage de programmation graphique Scratch. De nombreux arguments sont avancés pour justifier ce choix. Ainsi, ce langage est présenté comme : gratuit, multiplateforme, simple, fiable, robuste et capable de mettre œuvre tous les concepts à enseigner : « Il s'agit d'un logiciel gratuit et disponible sur toutes les plates-formes usuelles, choisi pour sa simplicité, sa fiabilité et sa robustesse dans la mise en œuvre. Il permet de

travailler tous les concepts figurant au programme » (MEN, 2016c, p. 2). De plus, il favorise la démarche de programmation par essais-erreurs en permettant des rétroactions instantanées et sans réprobations :

> En cliquant sur un bloc ou un script on peut observer immédiatement l'effet sur les lutins [...] aucun jugement de valeur n'est porté sur l'élève ou ses performances, la machine se contente d'effectuer les commandes et c'est l'élève lui-même qui constate s'il s'est trompé ou pas, et qui est invité à essayer une autre solution en cas d'échec. (MEN, 2016c, p. 5)

Scratch est à la fois un langage de programmation et un environnement d'édition de programmes. Les artefacts-supports recommandés pour l'exécuter sont l'ordinateur ou la tablette : « Les modalités de l'apprentissage correspondant peuvent être variées : [...] sur tablette ou sur ordinateur » (MEN, 2016c, p. 2).

Enfin, la ressource d'accompagnement R1M4 évoque la programmation débranchée, « c'est-à-dire sans utilisation d'un dispositif informatique » (MEN, 2016c, p. 2). Elle en propose un exemple mettant à profit un quadrillage dessiné dans la cour de récréation : « Les mouvements relatifs [des lutins] correspondent bien à des instructions de déplacement données à un être humain : elles peuvent être simulées physiquement, par exemple sur un quadrillage dessiné dans la cour du collège » (MEN, 2016c, p. 10). Dans cette configuration, les élèves jouent alternativement les rôles d'artefacts-supports puis d'artefacts-cibles des « programmes ».

2.8.2 Technologie au cycle 4

Nous exposons maintenant nos analyses concernant la technologie au cycle 4. Les sous-sections suivantes, structurées selon les quatre indicateurs, présentent les analyses détaillées des SOD.

Contenus pour la technologie

Débutons par l'étude des SOD relatives aux contenus regroupées dans le Tableau 2-18.

N°	Synthèses d'organisations didactiques (SOD)
1	Présenter la notion d'algorithme : formalisme permettant de décrire sa pensée ou de représenter la solution à un problème
2	Présenter la notion de programme : suite d'instructions qui pilote une machine dans l'objectif de résoudre un problème
3	Ne pas former des élèves experts en programmation
4	Modéliser les systèmes en chaîne d'énergie et chaîne d'information : acquérir l'information -> traiter l'information -> communiquer l'information -> chaine d'énergie ; il est primordial de mettre à disposition des élèves, à tout moment, la chaîne d'information complète, l'action de cette information sur la chaîne d'énergie doit aussi être mise en évidence
5	Introduire les notions de manière progressive : 5ème - nombre limité d'entrées-sorties, boucles itératives, 4ème - Plusieurs entrées-sorties, 3ème - comptage, boucles conditionnelles imbriquées, décomposition en sous-problèmes ; 1) l'action dépend uniquement d'une combinaison des informations d'entrées, 2) l'action dépend d'un enchaînement de décisions articulées par les structures conditionnelles et les boucles non bornées

Tableau 2-18 – SOD en rapport avec l'indicateur contenu pour la discipline technologie au cycle 4

Commençons par évoquer les notions d'algorithme et de programme, c'est l'objet des SOD 1, 2 et 3. À l'instar des mathématiques au cycle 4, ces notions font partie des connaissances visées en fin de cycle par le programme d'enseignement de technologie PT4 : « Notions d'algorithme et de programme » (MEN, 2020b, p. 116). L'analogie ne va pas plus loin car nous ne disposons pas, pour cette discipline, d'indications explicites allant dans le sens d'une indifférenciation entre ces deux notions. Néanmoins, des extraits du programme d'enseignement permettent de déceler des significations divergentes qui se rapprochent du savoir savant. Ainsi, la notion d'algorithme fait plutôt référence à un formalisme de description de la pensée ou de représentation de solutions à un problème posé : « Exprimer sa pensée à l'aide d'outils de description adaptés : [...] Notion d'algorithme » (MEN, 2020b, p. 120), et « Imaginer des solutions pour produire des objets et des éléments de programmes informatiques en réponse au besoin : [...] ; représentation de solutions (croquis, schémas, algorithmes) » (MEN, 2020b, p. 119). Alors que la notion de programme désigne la suite d'instructions qu'exécute une machine en vue de résoudre un problème : « dans le cadre des projets, les élèves [...] conçoivent tout ou partie d'un programme, le compilent et l'exécutent pour répondre au besoin du système » (MEN, 2020b, p. 123). À ce propos, la SOD 3 indique que les élèves ne doivent pas devenir des experts en programmation : « En outre, un enseignement d'informatique, est dispensé [...]. Celui-ci n'a pas pour objectif de former des élèves experts, mais de leur apporter des clés de décryptage d'un monde numérique en évolution constante » (MEN, 2020b, p. 116). Concernant l'aspect quantitatif, dans les textes qui constituent notre corpus pour la technologie, le terme « algorithme » est utilisé à quatre reprises contre vingt-quatre pour le terme « programme ». La notion de programme est donc largement majoritaire (24/28=86%).

La SOD 4 précise que les systèmes étudiés doivent être modélisés par une chaîne d'information (acquérir l'information -> traiter l'information -> communiquer l'information) : « si les éléments du chapitre d'informatique s'inscrivent naturellement dans le bloc de traitement de l'information, il est primordial de mettre à disposition des élèves, à tout moment, la chaîne d'information complète » (MEN, 2016a, p. 18). Cette modélisation des systèmes en chaînes mène à considérer les programmes informatiques sous l'angle de leurs entrées-sorties. C'est-à-dire comme des modules qui traitent de l'information en entrée dans l'objectif de contrôler des actionneurs en sortie.

Enfin, la cinquième SOD a pour objet la progression pédagogique. Cette progression est liée à la chronologie d'introduction des différentes notions algorithmiques, au nombre des entrées-sorties qui sont en jeu dans les situations, mais également à la complexité des problèmes posés. C'est ce qui ressort du programme d'enseignement PT4 :

> Repères de progressivité. En 5ème : traitement, mise au point et exécution de programmes simples avec un nombre limité de variables d'entrée et de sortie, développement de programmes avec des boucles itératives. En 4ème : traitement, mise au point et exécution de programmes avec introduction de plusieurs variables d'entrée et de sortie. En 3ème : introduction du comptage et de plusieurs boucles conditionnelles imbriquées, décomposition en plusieurs sous-problèmes. (MEN, 2020b, p. 124)

On retrouve un point de vue proche dans la ressource d'accompagnement R1T4 :

> Les activités liées au traitement de l'information au cycle 4 doivent être progressives. Dans un premier temps, les problèmes doivent se limiter à montrer qu'une action peut résulter uniquement d'une combinaison des informations d'entrées […]. Il convient par la suite de complexifier la programmation en montrant qu'une action dépend d'un enchaînement de décisions. Les structures de programmation telles que « si…alors…sinon », « faire…tant que… » permettent d'initier les élèves aux comportements des systèmes automatiques. (MEN, 2016a, p. 18)

Activités pour la technologie

Traitons maintenant les indications en lien avec les activités. Le Tableau 2-19 consigne les SOD ayant trait à cet indicateur.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
6	Démarche	Proposer une démarche d'investigation : étudier un système existant dans le but de le découvrir et de le comprendre ; faire observer et décrire le comportement et les éléments de programmation d'un système
7	Démarche	Proposer une démarche de résolution de problèmes techniques : résoudre un problème exprimé sous forme d'un besoin en agissant sur un système existant ; faire modifier un programme existant afin d'améliorer son comportement, ses performances
8	Démarche	Proposer une démarche de projet : concevoir et développer un système en réponse à un cahier des charges ; Faire écrire, mettre au point, exécuter un programme et vérifier le comportement attendu ; Faire décomposer le problème posé en sous-problème ; répondre au besoin du système et des fonctions à réaliser ; imaginer des solutions pour produire des objets et des éléments de programmes informatiques en réponse au besoin ; répondre à une problématique donnée, à partir d'un cahier des charges de fonctionnement ; faire agencer un robot : capteurs et actionneurs
9	Type activité	Faire écrire un programme : en programmant un système technique
10	Type activité	Faire modifier un programme existant : en améliorant un système technique
11	Mode prog.	Demander aux élèves de schématiser leurs algorithmes avant de les programmer

Tableau 2-19 – SOD en rapport avec l'indicateur activité pour la discipline technologie au cycle 4

Les SOD 6, 7 et 8 discernent clairement trois types de démarches : la démarche d'investigation, la démarche de résolution de problèmes et la démarche de projet. Dans le cas de la démarche d'investigation, il s'agit d'étudier un système existant dans le but de le découvrir et de le comprendre. Cette démarche se fonde prioritairement sur des activités d'observation et de description : « Observer et décrire le comportement d'un robot ou d'un système embarqué. En décrire les éléments de sa programmation » (MEN, 2020b, p. 123). La deuxième démarche, dite de résolution de problème, consiste à résoudre un problème exprimé sous la forme d'un besoin en agissant sur un système existant. Les activités tournent donc autour de la modification : « Modifier un programme existant dans un système technique, afin d'améliorer son comportement, ses performances pour mieux répondre à une problématique donnée » (MEN, 2020b, p. 123). Enfin, la démarche de projet est la plus complète, elle repose sur la conception et le développement d'un système en réponse à un cahier des charges. Elle engendre plusieurs types d'activités. D'abord l'activité de conception : « Analyser le comportement attendu d'un système réel et décomposer le problème posé en sous-problèmes afin de structurer un programme de commande » (MEN, 2020b, p. 123),

ensuite l'assemblage de composants matériels « Agencer un robot (capteurs, actionneurs) pour répondre à une activité et un programme donné » (MEN, 2020b, p. 123), et enfin l'écriture de programmes : « Écrire, à partir d'un cahier des charges de fonctionnement, un programme afin de commander un système » (MEN, 2020b, p. 123)

Au sujet des types d'activités de programmation, les SOD 9 et 10 décrivent deux recommandations, celle de rédiger un programme en entier en vue de piloter un système technique : « Écrire, mettre au point (tester, corriger) et exécuter un programme commandant un système réel » (MEN, 2020b, p. 123), et celle de modifier un programme existant dans le but d'améliorer un système technique : « Modifier un programme existant dans un système technique, afin d'améliorer son comportement, ses performances pour mieux répondre à une problématique donnée » (MEN, 2020b, p. 123).

La SOD 11 concerne le mode de programmation des élèves, c'est-à-dire la manière dont ils doivent procéder pour rédiger un programme répondant aux attendus. Il s'agit d'abord de formaliser un algorithme sous forme d'un algorigramme (nous développons ce formalisme dans la section liée aux artefacts). Cette modélisation peut donc être la première étape de la démarche globale de programmation. Elle sera suivie de sa traduction en programme. C'est ce procédé de programmation qui est décrit dans les ressources d'accompagnement R1T4 et R2T4 : « La définition des E/S d'un système automatique est une étape préliminaire pour rédiger un algorigramme ou un algorithme, ce qui permettra d'écrire le programme informatique du système » (MEN, 2016b, Onglet P2_1) et « Résoudre un problème technologique consiste donc à identifier des données et des contraintes, à définir des opérations et des moyens à mettre en œuvre. Il est souvent possible de formaliser par un organigramme la succession des étapes à respecter pour atteindre l'objectif. » (MEN, 2016a, p. 7).

Dispositifs pour la technologie

Cette sous-section traite des dispositifs conseillés dans les textes officiels. Les SOD en lien avec cet indicateur sont présentées dans le Tableau 2-20.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
12	Lieu	Faire travailler les élèves dans le laboratoire de technologie
13	Orga. espace	Proposer une disposition en îlots : mettre en place des îlots de 4 ou 5 élèves de niveau hétérogène, chaque îlot forme une équipe dans laquelle chacun aura un rôle complémentaire de celui des autres
14	Orga. espace	Proposer des temps communs au groupe classe : disposer d'un espace « pôle de réunion » pour les moment de mise en commun ou de créativité collective.
15	Interaction	Proposer des activités collaboratives : mises en commun de réalisations complémentaires ; information de l'équipe sur l'avancement des travaux de chacun et la préparation de documents de synthèse ; décomposer les tâches à programmer de manière fonctionnelle puis distribuer les tâches aux différents membres de l'équipe
16	Interaction	Proposer des activités personnelles : des phases de travail autonome viennent compléter des phases de travail avec l'équipe au complet

Tableau 2-20 – SOD en rapport avec l'indicateur dispositif pour la discipline technologie au cycle 4

Commençons par évoquer les lieux d'enseignement. Comme l'indique la SOD 12, les cours de technologie prennent place dans le laboratoire du même nom.

Les SOD 13 et 14 précisent l'organisation de l'espace. Elles décrivent deux configurations distinctes détaillées dans la ressource d'accompagnement R1T4 : la disposition « en îlot » propice aux travaux de groupes et la disposition en « pôle réunion » adaptée aux activités en classe entière. Évoquons d'abord la configuration en îlot qui regroupe une « équipe » d'élèves complémentaires : « Quatre ou cinq élèves, qui travaillent sur un îlot, ne doivent pas constituer un groupe, mais bien une équipe, dans laquelle chacun aura un rôle complémentaire à celui des autres membres » (MEN, 2016a, p. 11). Ces îlots se doivent de regrouper des élèves de niveaux différents, ce qui permet d'encourager les échanges : « L'hétérogénéité au sein d'une même équipe ne doit pas être considérée par le professeur comme un handicap mais bien au contraire comme un point fort sur lequel il pourra s'appuyer pour favoriser la mutualisation des connaissances et des compétences » (MEN, 2016a, p. 11). Enfin, certaines activités destinées au groupe classe sont plus adaptées à une disposition classique en salle banalisée ou au sein d'un « pôle réunion annexé ou intégré au laboratoire de technologie organisé en îlots » (MEN, 2016a, p. 11).

Enfin, concernant les interactions entre individus, (SOD 15 et 16), les textes officiels préconisent à la fois les travaux collaboratifs et individuels. Ainsi, le programme d'enseignement donne la ligne directrice : « cet enseignement doit se traduire par la réalisation de productions collectives (programme, application, animation, sites, etc.) dans le cadre d'activités de création numérique, au cours desquelles les élèves développent leur autonomie, mais aussi le sens du travail collaboratif » (MEN, 2020, p. 117). Ensuite, ces activités communes sont complétées par des travaux personnels, comme le précise la ressource d'accompagnement R1T4 : « le professeur propose des activités collaboratives [...] tout en intégrant du travail personnel aux différents membres de l'équipe. Des phases de travail autonome viennent compléter des phases de travail avec l'équipe » (MEN, 2016a, p. 11).

Artefacts pour la technologie

Finalement, intéressons-nous aux indications concernant les artefacts à utiliser en technologie. Le Tableau 2-21 regroupe les SOD caractérisées par cet indicateur.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
17	Cible	Faire programmer des systèmes pluri-technologiques réels didactisés ou non : appareils nomades, robots, systèmes embarqués, systèmes programmables de la vie courante, objets et systèmes présents dans le laboratoire, objet de la domotique (éclairage automatique, systèmes de surveillance)
18	Cible	Faire programmer des modélisations numériques
19	Support	Faire modéliser les algorithmes à l'aide d'organigrammes/algorigrammes
20	Support	Faire utiliser un langage de programmation graphique : interprété dans une base robotique ou des platines microcontrôleurs
21	Support	Faire programmer les systèmes technologiques à l'aide d'ordinateurs ou de tablettes

Tableau 2-21 – SOD en rapport avec l'indicateur artefact pour la discipline technologie au cycle 4

Les SOD 17 et 18 concernent les artefacts-cibles, c'est-à-dire ceux contrôlés par les programmes tandis que les SOD 19,20 et 21 portent sur les artefacts-supports, ceux qui permettent la création d'algorithmes et de programmes.

Pour la première catégorie d'artefacts, comme l'indique le programme d'enseignement, la programmation informatique en technologie vise à contrôler essentiellement « des systèmes pluri-technologiques réels didactisés ou non » (MEN, 2020b, p. 123). La ressource d'accompagnement R1T4 précise : « L'informatique et la programmation ne peuvent être enseignées hors d'un contexte technologique. Les concepts […] seront abordés […] autant qu'il est possible de le faire à partir des objets et systèmes présents dans le laboratoire » (MEN, 2016a, p. 4). Il s'agit : de robots, d'appareils nomades, de systèmes programmables de la vie courante ou d'objets domotiques (éclairage automatique, systèmes de surveillance). Si, pour diverses raisons (coût, place, etc.), ces objets ne sont pas accessibles, le programme indique que : « ils peuvent être complétés par l'usage de modélisations numériques permettant des simulations et des modifications du comportement » (MEN, 2020b, p. 123).

Ensuite, concernant les artefacts-supports, un outil est dédié à l'expression des algorithmes, il est nommé « organigramme » ou « algorigramme ». On peut en effet lire dans les ressources d'accompagnement R1T4 et R2T4 : « il est souvent possible de formaliser par un organigramme la succession des étapes à respecter pour atteindre l'objectif » (MEN, 2016a, p. 7) ou « définir les variables entrées et les variables de sorties du futur programme et écrire ce dernier sous forme d'algorigramme » (MEN, 2016b, Onglet P2_1). Ces organigrammes/algorigrammes sont des représentations graphiques constituées de blocs et de flèches permettant de modéliser l'enchaînement des opérations et des décisions structurant les algorithmes.

Le programme d'enseignement indique que les programmes informatiques doivent être édités à l'aide de logiciels permettant de manipuler des représentations graphiques des instructions, autrement dit, des blocs : « La conception, la lecture et la modification de la programmation sont réalisées au travers de logiciels d'application utilisant la représentation graphique simplifiée des éléments constitutifs de la programmation » (MEN, 2020b, p. 123). Ces logiciels d'édition de blocs peuvent être utilisés sur différents supports : « la programmation est pilotée par ordinateur ou une tablette numérique » (MEN, 2020b, p. 123). Finissons par une remarque : aucun langage ni logiciel de programmation graphique particulier n'est mis en avant pour la technologie. En effet, la ressource d'accompagnement R1T4 rappelle que les programmes devront être exécutés ou interprétés par les artefacts-cibles : « Les langages de programmation doivent se limiter à de la programmation graphique qui est interprétée et compilée dans une base robotique ou des platines microcontrôleurs » (MEN, 2016a, p. 18). C'est donc l'artefact-cible choisi qui impose le langage et donc le logiciel de programmation graphique qui permet de le contrôler. Ce domaine ne faisant l'objet d'aucune standardisation, ni d'interopérabilité, les langages de programmation par blocs utilisés varient selon le matériel : Blockly, ArduBlock, Scratch, mBloc, etc.

2.8.3 Mathématiques en seconde

Nous analysons maintenant les indications données par le programme d'enseignement et les ressources d'accompagnement de mathématiques en seconde. Nous présentons dans la suite les SOD en fonction des différents indicateurs pédagogico-didactique.

Contenus pour les mathématiques en seconde

La Tableau 2-22 ci-dessous contient les SOD relatives aux contenus.

N°	Synthèses d'organisations didactiques (SOD)
1	Présenter la notion d'algorithme : ensemble de constructions élémentaires permettant de résoudre un problème
2	Présenter la notion de programme : suite d'instructions d'un langage de programmation
3	Ne pas exiger de technicité au niveau de la syntaxe Python : la compréhension est privilégiée par rapport à l'optimisation du code des programmes ; ne pas former des experts dans un langage de programmation
4	Mobiliser les notions de façon naturelle dans les activités
5	Introduire progressivement l'imbrication des différentes instructions
6	Présenter les variables de façon métaphorique : une variable est présentée comme une étiquette collée sur une boite qui contient des valeurs différentes au cours de l'exécution du programme, l'exécution séquentielle d'un programme est présentée comme une suite temporelle de configurations des boites et de leurs contenus
7	Présenter la notion de fonction comme un sous-programme indépendant plus petit et réutilisable
8	Être attentif à la différence entre affectation et test d'égalité

Tableau 2-22 – SOD en rapport avec l'indicateur contenu pour la discipline mathématiques en seconde

Les SOD 1, 2 et 3 portent sur les notions d'algorithme et de programme. Du point de vue du sens, un algorithme est défini dans la ressource d'accompagnement R2MS comme une suite de constructions élémentaires décrivant la procédure de résolution d'un problème :

> Un algorithme est une procédure de résolution de problème […] [Il] produit, en un nombre fini d'étapes constructives, effectives, non-ambigües et organisées, la réponse au problème […]. De la même façon qu'un script Scratch se construit en accolant des briques élémentaires, un algorithme s'appuie sur un ensemble très réduit de constructions : l'affectation d'une variable, la séquence d'instructions, l'instruction conditionnelle, les boucles (bornées ou non bornées), les fonctions. (MEN, 2017b, p. 2)

La notion de programme est désignée, dans le programme d'enseignement, par une suite d'instructions exprimées dans un langage de programmation textuel : « un programme simple écrit dans un langage de programmation textuel » (MEN, 2019b, p. 15), ou « un langage de programmation simple d'usage est nécessaire pour l'écriture des programmes informatiques » (MEN, 2019b, p. 15). La définition de l'algorithme étant ici assez proche de celle du programme, c'est donc leur forme, plus que leur sens, qui différencie ces deux notions. En effet, comme nous le verrons dans la partie qui traite des artefacts, les algorithmes doivent être formalisés en pseudo-langage alors que les programmes sont écrits en Python. À ce propos, les ressources d'accompagnement R1MS et R2MS précisent que l'objectif n'est pas de former des experts dans un langage de programmation ou une bibliothèque particulière : « Le professeur gardera à l'esprit que l'enseignement de la partie algorithmique et programmation n'a pas pour objectif de former des experts dans tel ou tel langage de programmation ou dans la connaissance détaillée de telle ou telle bibliothèque de programme » (MEN, 2017b, p. 1), et « Aucune technicité au niveau de la syntaxe du langage Python n'est attendue. La compréhension est privilégiée par rapport à

l'optimisation du code des programmes » (MEN, 2019a, p. 2). Concernant l'aspect quantitatif, le terme « algorithme » est mentionné dix-huit fois contre vingt-sept pour le terme « programme » dans les textes pris en compte. Cette répartition (40% - 60%) est en faveur de la notion de programme, elle est cependant plus équilibrée qu'au cycle 4. Ces références plus nombreuses à la notion d'algorithme pourraient s'expliquer par une existence propre et autonome rendue possible par l'introduction d'un pseudo-langage permettant leurs descriptions. De plus, comme l'indique le programme (voir section 2.4), le concept d'algorithme existe historiquement par lui-même en mathématiques. Il est légitime sans forcément avoir de lien avec l'informatique. Pensons par exemple à l'emblématique algorithme d'Euclide.

Ensuite, les SOD 4 et 5 concernent la manière de mettre en jeu les notions dans les activités. De la même manière que pour les mathématique au cycle 4, la ressource d'accompagnement R2MS insiste sur le fait que les notions doivent être convoquées de manière « naturelle » dans les activités : « il n'est pas recommandé de les présenter [les constructions algorithmiques] de façon magistrale aux élèves, qui les ont déjà rencontrées au cycle 4, mais de les mobiliser de façon naturelle dans les activités travaillées » (MEN, 2017b, p. 2). D'autre part, une indication est donnée au sujet de la progression, ces constructions doivent être utilisées : « en introduisant progressivement leur imbrication. » (MEN, 2017b, p. 2). Ainsi les concepts fondamentaux de la programmation doivent être implémentés dans un premier temps seuls avant d'envisager leurs imbrications.

Enfin, les SOD 6, 7 et 8 donnent des indications relatives à des notions particulières. On retrouve d'abord dans la SOD 6, comme au cycle 4, la présentation métaphorique des variables à travers le modèle d'une étiquette collée sur une boîte : « un modèle rigoureux de la variable informatique consiste à dire qu'une variable est une étiquette collée sur une boîte qui peut contenir différentes valeurs. Le contenu de chaque boîte varie au cours de l'exécution d'un programme » (MEN, 2017b, p. 2). La SOD 7 porte sur la manière de présenter une fonction informatique, la ressource d'accompagnement R1MS précise :

> Une fonction informatique est une portion de code représentant un sous-programme effectuant une tâche ou un calcul relativement indépendant du reste du programme. Le découpage d'un programme en sous-programmes plus petits et réutilisables permet d'en traiter séparément différentes parties. (MEN, 2019a, p. 2)

Il s'agit donc d'exposer cette notion comme un sous-programme indépendant, plus petit et réutilisable. Enfin, la SOD 8 constitue un point de vigilance pour les enseignants. La ressource d'accompagnement R2MS attire l'attention sur la différence syntaxique qu'il existe en informatique, à la différence des mathématiques, entre l'affectation et le test d'égalité :

> On observe également que le symbole = a été choisi pour l'opération d'affectation en Python. Le symbole = n'est donc pas disponible pour le test d'égalité qui s'écrit ==. On sera attentif à cette différence : l'affectation est absente en mathématiques et omniprésente en informatique, le test d'égalité d'un usage moins fréquent en informatique. (MEN, 2017b, p. 3)

Activités pour les mathématiques en seconde

Nous évoquons ici les aspects liés aux activités et présentons dans le Tableau 2-23 les SOD qui s'y rapportent.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
9	Démarche	Faire résoudre des exercices et les problèmes de programmation
10	Démarche	Faire travailler les élèves dans une démarche de mini-projet
11	Type activité	Demander de tester un programme : demander d'exécuter un programme et d'en interpréter les sorties
12	Type activité	Demander d'expliquer un algorithme ou un programme : demander d'interpréter une ou plusieurs lignes d'un programme informatique fourni
13	Type activité	Demander de modifier un algorithme ou un programme : demander de changer quelques lignes de code pour répondre à une nouvelle question, de corriger une erreur ou d'améliorer un programme
14	Type activité	Demander de compléter un algorithme ou un programme : demander de compléter le programme pour manifester sa compréhension des notions mathématiques ou informatiques mises en jeu
15	Type activité	Demander d'écrire un algorithme ou un programme : demander de programmer une fonction ou une suite d'instructions répondant à un problème donné ; les élèves s'exercent à décrire des algorithmes en langage naturel
16	Type activité	Adapter le type d'activité à la complexité des algorithmes : demande d'écrire un programme dans les cas simples, et de tester, expliquer, modifier ou compléter un programme dans les cas plus complexes
17	Mode prog.	Demander de traduire un algorithme en programme : entraîner les élèves à passer du langage naturel (pseudo-code) à Python
18	Mode prog.	Favoriser l'approche fonctionnelle au détriment des entrées-sorties : écrire les programmes dans de fonctions rédigées dans les fichiers ; interpréter ces fichiers ; appeler les fonctions depuis la console en passant des arguments

Tableau 2-23 – SOD en rapport avec l'indicateur activité pour la discipline mathématiques en seconde

Deux types de démarches sont préconisés en mathématique en seconde. D'abord la résolution d'exercices et des problèmes (SOD 9). Ainsi, on peut lire dans le programme d'enseignement : « L'utilisation régulière de ces outils [les logiciels de programmation] peut intervenir [...] à l'occasion de la résolution d'exercices ou de problèmes » (MEN, 2019b, p. 3). Ensuite, les textes officiels indiquent la possibilité de mettre en place des mini-projets : « L'écriture de sous-programmes peut être confiée à différents groupes d'élèves, favorisant [...] la mise en place d'activités de projet » (MEN, 2019a, p. 2) et « Les modalités de mise en œuvre peuvent être variées : [...] mini-projets pouvant être réalisés en groupe » (MEN, 2017b, p. 7).

Les SOD 10 à 16 traitent du type des activités de programmation que les enseignants peuvent proposer aux élèves. Les textes que nous avons à notre disposition en distinguent cinq différents (SOD 9 à 13) :

- tester un programme : « il est alors demandé d'exécuter un programme et d'en interpréter les sorties » (MEN, 2019a, p. 1) ;
- expliquer un algorithme ou un programme : « il est alors demandé d'interpréter une ou plusieurs lignes d'un programme informatique fourni » (MEN, 2019a, p. 1) et « interpréter [...] des algorithmes » (MEN, 2019b, p. 15) ;
- modifier un algorithme ou un programme : « il est alors demandé de changer quelques lignes de code pour répondre à une nouvelle question, de corriger une

erreur ou d'améliorer un programme » (MEN, 2019a, p. 1) et « modifier des algorithmes » (MEN, 2019b, p. 15) ;

– compléter un algorithme ou un programme : « il est alors demandé de compléter le programme » (MEN, 2019a, p. 2) et « compléter [...] des algorithmes » (MEN, 2019b, p. 15) ;

– écrire un algorithme ou un programme : « il est alors demandé de programmer une fonction ou une suite d'instructions répondant à un problème donné » (MEN, 2019a, p. 2) et « décrire des algorithmes » (MEN, 2019b, p. 15).

Ces types d'activités constituent un gradient pour l'élève, le portant de l'interprétation à la production d'algorithme ou de programme. Elles nécessitent vraisemblablement une compréhension de plus en plus fine des concepts de la programmation, et surtout, une maîtrise de plus en plus importante de la syntaxe du langage de programmation. La SOD 14 indique que le choix du type de l'activité doit dépendre de la complexité de l'algorithme sous-jacent à la situation proposée aux élèves. Ainsi, le programme d'enseignement indique : « en réaliser quelques-uns [des algorithmes] à l'aide d'un programme simple écrit dans un langage de programmation textuel ; interpréter, compléter ou modifier des algorithmes plus complexes » (MEN, 2019b, p. 15). Un peu plus loin dans le programme on peut lire : « Programmer, dans des cas simples, une boucle bornée, une boucle non bornée. Dans des cas plus complexes : lire, comprendre, modifier ou compléter un algorithme ou un programme » (MEN, 2019b, p. 15). L'activité de type « écrire » semble donc être réservée aux algorithmes et programmes les plus « simples », concernant les cas « plus complexes », il s'agit de proposer des activités du type « tester », « expliquer », « modifier » ou « compléter ».

Ensuite, les SOD 17 et 18 donnent des détails concernant le mode de programmation conseillé aux élèves. Autrement dit, la marche à suivre afin de rédiger un programme. Lorsque l'objectif de l'activité est de produire un programme répondant à un problème donné, il peut être demandé aux élèves de concevoir, dans un premier temps, un algorithme en pseudo-langage (nous détaillons cet aspect dans la partie artefact) avant de le traduire en langage de programmation (SOD17) : « Concevoir des algorithmes et les traduire dans un langage de programmation » (MEN, 2017b, p. 2), et « Les élèves sont entraînés à passer du langage naturel à Python » (MEN, 2019b, p. 15). Ensuite, lors de la traduction d'un algorithme ou de la rédaction directe d'un programme, la ressource d'accompagnement R2MS recommande un mode de programmation en trois temps qui privilégie l'approche fonctionnelle au détriment des entrées-sorties[14] (SOD 18) :

– les élèves écrivent leur code dans des fonctions enregistrées dans des fichiers-modules : « Les élèves écriront des fonctions qui pourront être enregistrées dans des fichiers, appelés scripts ou modules, qu'on peut développer, enrichir et réutiliser lors de séances successives » (MEN, 2017b, p. 5) ;

– le module est interprété, sans effet, dans le seul but de rendre accessibles les fonctions dans l'environnement de la console : « L'exécution d'un module comportant la définition d'une fonction *f* ne produit aucun affichage

[14] Cette approche semble inspirée du paradigme de programmation fonctionnel que nous avons présenté dans la section 1.4.3. Cela se rapproche également du mode de programmation utilisé dans les notebooks et les consoles Python en mode REPL (Read-Evaluate-Print Loop)

particulier dans la console, même si la définition a été prise en compte » (MEN, 2017b, p. 5)

– les fonctions du module sont appelées depuis la console qui affiche leurs retours : « on peut maintenant, dans la console, faire des appels du type $f(2)$ et la console affiche la valeur renvoyée par la fonction » (MEN, 2017b, p. 5)

L'emploi de ce mode opératoire est ensuite justifié dans la ressource d'accompagnement R2MS. Il permet ainsi d'éviter l'usage des fonctions d'entrée-sortie « input » et « print » qui ne relèveraient pas de la pensée algorithmique[15] :

> On notera que les notions d'entrées-sorties (fonctions input et print) ne sont pas développées dans ce document : elles ne relèvent pas de la pensée algorithmique et l'accent mis par le programme sur la notion de fonction permet de s'en libérer complètement. (MEN, 2017b, p. 5)

La ressource d'accompagnement R1MS va également dans ce sens. Elle affirme que ce mode de programmation, basé sur les fonctions, permet de « prendre très tôt de bonnes habitudes de programmation » (MEN, 2019a, p. 2) en obligeant les élèves à mettre en œuvre de bonnes pratiques concernant les entrées-sorties. Ainsi, cette méthode dispense les élèves d'utiliser l'instruction « input » permettant la saisie des entrées par l'utilisateur : « l'usage des fonctions évite de faire appel à des instructions du type (int(input("a ?"))) qui obscurcissent le code. Les entrées apparaissent comme paramètres de la fonction » (MEN, 2019a, p. 2). De plus, la méthode évite l'emploi de l'instruction « print » pour les sorties : « l'exécution de celle-ci [la fonction] fournit les sorties souhaitées sans avoir recours à l'instruction print » (MEN, 2019a, p. 2).

Dispositifs pour les mathématiques en seconde

Nous nous penchons ici sur les dispositifs préconisés dans les textes officiels, le Tableau 2-24 présente les SOD correspondantes.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
19	Lieu	Faire travailler les élèves en salle de classe banalisée, en salle informatique ou au CDI
20	Orga espace	Projeter l'activité du professeur à la classe entière à l'aide d'un dispositif de visualisation collective
21	Orga espace Interaction	Mettre les élèves en activité individuellement ou en groupe
22	Interaction	Favoriser le travail collaboratif : en confiant l'écriture de sous-programmes à des groupes différents

Tableau 2-24 – SOD en rapport avec l'indicateur dispositif pour la discipline mathématiques en seconde

Les lieux indiqués pour pratiquer la programmation informatique sont les salles de classes classiques, les salles informatiques ainsi que le CDI (SOD 18). Le programme d'enseignement indique en effet que : « L'utilisation régulière de ces outils [logiciel de programmation] peut intervenir [...] par les élèves, en classe, à l'occasion de la résolution d'exercices ou de problèmes ; dans le cadre du travail personnel des élèves

[15] Bien qu'il ne soit pas défini dans le document, le terme « pensée algorithmique » semble faire ici référence au « computational thinking » de Wing (2006).

hors du temps de classe (par exemple au CDI ou à un autre point d'accès au réseau local) » (MEN, 2019b, p. 3). La ressource d'accompagnement R2MS précise : « Les modalités de l'apprentissage correspondant peuvent être variées : […] en salle informatique ou en salle banale » (MEN, 2017b, p. 1).

L'organisation de l'espace est l'objet des SOD 20 et 21. Plusieurs dispositifs sont mis en avant. Une première modalité est suggérée dans laquelle le professeur projette ses activités à la classe entière au tableau. Nous trouvons dans les différents textes : « L'utilisation régulière de ces outils [logiciel de programmation] peut intervenir […] par le professeur, en classe, avec un dispositif de visualisation collective adapté » (MEN, 2019b, p. 3) ou «Les modalités de mise en œuvre peuvent être variées : […] vidéo-projection en classe entière » (MEN, 2017b, p. 7) . Ensuite, les textes prévoient une organisation de la classe individuelle ou en groupe : « Les modalités de l'apprentissage correspondant peuvent être variées : travail individuel ou en groupe » (MEN, 2017b, p. 1).

Enfin, les interactions entre les différents acteurs de l'enseignement sont évoquées dans les SOD 21 et 22, il s'agit de proposer du travail en autonomie « L'utilisation régulière de ces outils [logiciel de programmation] peut intervenir dans le cadre du travail personnel des élèves » (MEN, 2017b, p. 7) ou des tâches favorisant la collaboration : « L'écriture de sous-programmes peut être confiée à différents groupes d'élèves, favorisant le travail collaboratif » (MEN, 2019a, p. 2).

Artefacts pour les mathématiques en seconde

Pour finir avec les mathématiques au niveau seconde, nous évoquons ci-dessous les organisations didactiques relevant de l'indicateur artefact. Le Tableau 2-25 liste les SOD qui en découlent.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
23	Support	Exprimer les algorithmes sous la forme d'un pseudo-langage en français
24	Support	Faire utiliser le langage de programmation textuel Python
25	Support	Faire programmer sur tablette, ordinateur fixe ou portable ou, calculatrice
26	Support	Faire écrire des algorithmes au tableau ou sur papier

Tableau 2-25 – SOD en rapport avec l'indicateur artefact pour la discipline mathématiques en seconde

Nous remarquons ici que seule la catégorie artefacts-supports apparaît. Comme l'indique la SOD 23, les algorithmes peuvent être exprimés en « langage naturel ». On retrouve ainsi dans le programme d'enseignement : « Dans le cadre de cette activité [l'algorithmique et la programmation], les élèves s'exercent à décrire des algorithmes en langage naturel » (MEN, 2019b, p. 15). Comme nous l'avons évoqué dans la section 1.4.2, en science informatique le terme « langage naturel » fait généralement référence à un langage utilisé par les humains pour communiquer entre eux, tel le français ou l'anglais. Toutefois, nous retrouvons l'élément suivant dans le programme d'enseignement : « Affectation (notée ← en langage naturel) » (MEN, 2019b, p. 15). Cette flèche est emblématique des pseudo-langages utilisés dans le but de décrire les algorithmes indépendamment d'un langage d'implémentation. Cette indication laisse penser que la dénomination « langage naturel » désigne en fait ici un pseudo-langage francisé de description d'algorithmes. On retrouve d'ailleurs ce formalisme dans une

précédente ressource d'accompagnement portant sur l'algorithmique datant de 2009 : « Les "instructions" sont les "briques de base" des algorithmes [...]. Nous les présenterons dans un pseudo-langage "en français" » (MEN, 2009b, p. 7).

L'artefact-support choisi pour construire les programmes est le langage de programmation Python (SOD 24). Ce choix est justifié dans les divers textes par de nombreux arguments. Python y est décrit comme un langage interprété, concis, simple, supporté par de nombreux environnements et disposant d'une large communauté d'utilisateurs mettant à disposition de nombreuses ressources : « Le langage choisi est Python, langage interprété, concis, largement répandu et pouvant fonctionner dans une diversité d'environnements» (MEN, 2019b, p. 15) et « le langage Python, choisi pour la concision et la simplicité de sa syntaxe, la taille de la communauté d'utilisateurs (en particulier dans le cadre éducatif), ainsi que la richesse des ressources disponibles » (MEN, 2017b, p. 1). De plus, il donne l'occasion aux élèves d'utiliser une syntaxe précise et rigide tout en bénéficiant du contrôle de l'interpréteur :« Le choix d'un langage textuel, comme Python, [...] permet aux élèves de se confronter à la précision et la rigidité d'une syntaxe proche de celle des expressions mathématiques, avec l'avantage de pouvoir bénéficier du contrôle apporté par l'analyseur syntaxique » (MEN, 2017b, p. 1). Enfin, Python est présenté comme ayant des similitudes avec Scratch, ce qui faciliterait la transition entre les deux langages : « En classe de seconde, le passage de Scratch à Python peut être immédiat ou progressif, suivant les choix pédagogiques de l'enseignant. Les deux langages comportent, au-delà des différences évidentes de forme, des similitudes qui facilitent la transition » (MEN, 2017b, p. 6).

Aucun environnement de développement ni outil d'édition n'est conseillé pour le développement des programmes. En revanche, les machines supportant ces environnements de programmation sont décrites (SOD25). Il peut s'agir de tablettes, d'ordinateurs portables (classe mobile) ou d'ordinateur fixes : « Les modalités de mise en œuvre peuvent être variées : [...] travail sur machine, utilisation de tablettes ou classes mobiles » (MEN, 2017b, p. 7). Enfin, les artefacts-supports mobilisés pour la modélisation des algorithmes en pseudo-langage sont le papier ou le tableau de la salle de classe (SOD26) : « Les modalités de mise en œuvre peuvent être variées : [...] travail sur papier ou au tableau » (MEN, 2017b, p. 7).

2.8.4 SNT en seconde

Nous analysons ici les SOD établies pour la discipline SNT en classe de seconde. Les sous-sections suivantes détaillent nos analyses en fonction des indicateurs. Nous faisons remarquer au préalable que pour cette discipline, nous disposons de peu d'éléments concernant les modalités d'enseignement dans les textes officiels. Nous faisons l'hypothèse que cette discipline venant d'être créée, les rédacteurs des programmes et des ressources d'accompagnement ne disposaient sans doute pas du recul nécessaire permettant de fournir des conseils de mise en œuvre.

Contenus pour les SNT

Les SOD relatives aux contenus sont au nombre de deux, elles sont présentées dans le Tableau 2-26.

N°	Synthèses d'organisations didactiques (SOD)
1	Présenter la notion d'algorithme : spécification abstraite des traitements à effectuer sur des données à partir d'opérations élémentaires
2	Présenter la notion de programme : la traduction d'un algorithme exécutable par un ordinateur

Tableau 2-26 – SOD en rapport avec l'indicateur contenu pour la discipline SNT en seconde

Nous pouvons remarquer que les notions d'algorithme et de programme sont bien différenciées. Comme le précise la SOD 1, un algorithme est défini comme une spécification des traitements à effectuer sur des données à partir d'opérations élémentaires : « les algorithmes, qui spécifient de façon abstraite et précise des traitements à effectuer sur les données à partir d'opérations élémentaires » (MEN, 2019e, p. 2). Un programme est défini comme la traduction d'un algorithme dans un langage exécutable par un ordinateur (SOD 2) : « les langages, qui permettent de traduire les algorithmes abstraits en programmes textuels ou graphiques de façon à ce qu'ils soient exécutables par les machines » (MEN, 2019e, p. 2). Notons que ces définitions sont très proches du savoir savant, et semblent découler directement des quatre concepts de l'informatique de Dowek (2011). D'un point de vue quantitatif, nous rencontrons, dans le programme d'enseignement, 32 fois le terme « algorithme » contre 20 fois le terme « programme ». Pour cette discipline, c'est donc la notion d'algorithme qui est la plus mobilisée dans les textes officiels (32/52=62%). Cette prédominance peut s'expliquer par la place centrale qu'a acquis cette notion dans toutes les sphères de la société (voir section 1.2.1). On retrouve donc, en cohérence avec les évolutions sociétales, des références aux algorithmes dans chacun des sept thèmes du programme : algorithmes de routage dans les réseaux, algorithmes de recommandations sur Internet, algorithmes d'exploitation des données, algorithmes de contrôle des objets connectés, algorithmes de calculs d'itinéraires, algorithmes de prise de vues et de traitement d'images.

Activités pour les SNT

Le Tableau 2-27 regroupe les SOD en lien avec les activités.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
3	Type activité	Faire écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes liés aux thèmes du programme
4	Type activité	Faire identifier des algorithmes
5	Type activité	Faire expliciter des algorithmes
6	Démarche	Proposer aux élèves de faire des exposés
7	Démarche	Proposer des mini-projets

Tableau 2-27 – SOD en rapport avec l'indicateur activité pour la discipline SNT en seconde

Ici encore, nous ne disposons que de peu d'éléments. D'abord, le programme d'enseignement préconise trois types d'activités : « écrire et développer des programmes pour répondre à des problèmes » (MEN, 2019e, p. 3), « identifier des algorithmes de contrôle des comportements physiques » (MEN, 2019e, p. 17) et « expliciter des algorithmes associés à la prise de vue » (MEN, 2019e, p. 19). Ensuite,

le programme insiste sur la variété des démarches à proposer aux élèves et cite quelques possibilités : « Cet enseignement a vocation à multiplier les occasions de mise en activité des élèves, sous des formes variées (exposés, […], mini-projets, […].) [souligné par l'auteur] » (MEN, 2019e, p. 3).

Dispositifs pour les SNT

Les SOD en lien avec les dispositifs sont présentées dans le Tableau 2-28.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
8	Interactions	Demander des productions individuelles
9	Interactions	Proposer des travaux en groupes : demander des productions collectives

Tableau 2-28 – SOD en rapport avec l'indicateur dispositif pour la discipline SNT en seconde

Les éléments relatifs aux dispositifs sont quasi inexistants dans le programme d'enseignement. Il est simplement recommandé d'en varier la forme en passant d'activités individuelles (SOD 8) à des travaux en groupes (SOD 9) : « Cet enseignement a vocation à multiplier les occasions de mise en activité des élèves […] travaux en groupe, […], productions individuelles ou collectives » (MEN, 2019e, p. 3). Nous pouvons donc ici aussi supposer que ces productions collectives en groupes nécessitent une certaine collaboration des élèves entre eux.

Artefacts pour les SNT

Enfin, le Tableau 2-29 contient les recommandations ayant trait aux artefacts.

N°	Catégories	Synthèses d'organisations didactiques (SOD)
10	Support	Faire utiliser aux élèves le langage Python
11	Support	Faire utiliser aux élèves les langages HTML et CSS
12	Cible	Faire programmer des objets connectés : réaliser une IHM, gérer les entrées-sorties à travers les ports

Tableau 2-29 – SOD en rapport avec l'indicateur artefact pour la discipline SNT en seconde

Le programme d'enseignement prescrit l'utilisation de l'artefact-support Python (SOD 10). Certains arguments justifiant ce choix sont similaires à ceux exposés dans le programme d'enseignement de seconde de mathématiques. Ainsi ce langage est décrit comme simple, interprété, concis, multiplateforme et disposant d'une grande communauté d'utilisateurs. D'autres motifs sont cependant avancés, Python est libre, gratuit et riche en bibliothèques appropriées aux activités envisagées dans les sept thèmes du programme :

> Un langage de programmation est nécessaire pour l'écriture des programmes : un langage simple d'usage, interprété, concis, libre et gratuit, multiplateforme, largement répandu, riche de bibliothèques adaptées aux thématiques étudiées et bénéficiant d'une vaste communauté d'auteurs dans le monde éducatif est nécessaire. Au moment de la conception de ce programme, le langage choisi est Python. (MEN, 2019e, p. 3)

Un autre artefact-support est à utiliser (SOD 11), il s'agit du couple HTML/CSS qui permet de décrire les pages Web et leur style. À ce sujet, le programme d'enseignement précise : « Contenus : Langage HTML et CSS. Distinguer ce qui relève du contenu d'une

page et de son style de présentation. Étudier et modifier une page HTML simple » (MEN, 2019e, p. 8).

Enfin, il est suggéré dans le programme d'enseignement d'utiliser les langages de programmation pour piloter des artefacts-cibles, en l'occurrence des objets connectés (SOD 12) : « Partie Informatique embarquée et objets connectés : Commande d'un actionneur, acquisition des données d'un capteur. Écrire des programmes simples d'acquisition de données ou de commande d'un actionneur » (MEN, 2019e, p. 17).

2.8.5 Bilan transdisciplinaire par indicateur

Nous venons de présenter les indications contenues dans les textes officiels qui concernent les modalités d'enseignement de la programmation informatique. Nous avons exposé ces différents éléments par disciplines. Nous procédons maintenant à un bilan transdisciplinaire par indicateur. Nous allons pour cela nous appuyer sur le Tableau 2-30 qui reprend de manière synthétique les indications que nous avons évoquées dans cette section.

	Cat.	Mathématiques cycle 4	Technologie cycle4	Mathématiques seconde	SNT seconde
Contenu		Algorithme = programme Pas de connaissances expertes Introduire les concepts naturellement selon le besoin	Algorithme ≠ programme Ne pas former d'experts Introduire graduellement les concepts	Algorithme ≠ programme Ne pas former d'experts Introduire les concepts naturellement dans les activités et progressivement leur imbrication	Algorithme ≠ programme
Activité	Démarche	Résolution de problèmes Mini projet	Démarche d'investigation Résolution de problèmes Démarche projet	Résolution d'exercices ou de problèmes Mini-projets	Mini-projet Exposé
	Type Activité	Écrire un programme Compléter, améliorer, corriger un programme	Écrire un algorithme ou un programme Modifier un programme	Tester un programme Expliquer, modifier, compléter, écrire un algo. ou un programme	Écrire un programme Identifier, expliciter un algorithme
	Mode progr.	Directement en blocs Par tâtonnements et boucles courtes d'essais-erreurs	Concevoir un algorithme sous la forme d'un algorigramme Traduction en langage de prog.	Concevoir un algorithme en pseudo-langage Traduction en langage de prog.	∅
Dispositif	Lieu	Salle de classe Salle informatique	Laboratoire de technologie	Salle de classe Salle informatique / CDI	∅
	Orga. espace	Groupes possibles	Îlots : petits groupes Pôle réunion : classe entière	Le professeur projette ses activités au tableau à la classe entière Groupes possibles	∅
	Interaction	Travail individuel Collaboration entre élèves L'enseignant intervient a minima	Travail individuel Collaboration entre élèves	Travail individuel Collaboration entre élèves	Travail individuel Collaboration entre élèves
Artefact	Support	Langage de prog. par blocs : Scratch Ordinateurs ou tablettes	Algorigramme et langage de prog. par blocs : dépend du matériel Ordinateurs ou tablettes	Pseudo -langage et langage de prog. textuel : Python Ordinateurs, tablettes	langage et langage de prog. textuel : Python, HTML/CSS
	Cible	∅	Système technologiques : robots, objets domotiques, etc.	∅	Objets connectés : actionneurs et capteurs

Tableau 2-30 – Tableau récapitulatif des modalités d'enseignement de la programmation par indicateur et discipline

Commençons par l'indicateur des contenus qui caractérise les indications ministérielles directement en lien avec les concepts. Nous pouvons d'abord observer que la distinction entre algorithme et programme est totale au lycée et que leurs acceptions sont une transposition directe du savoir savant (un algorithme est une procédure systématique de résolution de problème, un programme est une suite d'instructions à destination des machines ayant pour but d'implémenter un algorithme). Pour le cycle 4, en mathématiques, il s'agit de ne pas différencier ces deux concepts qui désignent tous les deux un programme ou un script. En technologie ces concepts sont distincts mais le concept d'algorithme qualifie plus largement un outil d'aide à la pensée. Tous les textes officiels, exceptés ceux de SNT, insistent sur le fait de ne pas former des experts dans un langage de programmation ou une bibliothèque particulière. Il s'agit avant tout d'introduire (cycle 4) et d'approfondir (seconde) les concepts fondamentaux de la programmation. Les mêmes textes précisent que ces concepts doivent être introduits « naturellement » en réponse à des besoins apparus lors de la résolution de problèmes. Il s'agit également de veiller à le faire progressivement.

Ensuite, l'indicateur des activités est divisé en trois catégories. Débutons par la démarche. Il est possible d'observer une dichotomie, avec d'un côté, les disciplines mathématiques qui proposent des activités de résolution d'exercices et de problèmes, accompagnés de « mini-projets ». Et de l'autre côté, les disciplines technologie et SNT tournées vers des démarches plus variées (investigation, projet plus conséquent, exposé). Au sujet du type d'activité, nous retrouvons de nouveau la même séparation. En mathématiques, les variations sont plus riches, il s'agit de compléter, améliorer, corriger, modifier, expliquer, écrire ou test un programme, là où la technologie et les SNT proposent moins d'alternatives : écrire, modifier, identifier et expliciter. Concernant les modes de programmation, nous trouvons d'un côté les mathématiques au cycle 4, avec un mode opératoire qui consiste à programmer directement en blocs sur le mode du tâtonnement et de l'essai-erreur. Il est clair que la souplesse d'usage et la facilité d'exécution de Scratch favorise ce mode de programmation. De l'autre côté, nous trouvons la technologie et les mathématiques en seconde qui préconisent un mode opératoire plus réfléchi, nécessitant une expression algorithmique de la solution du problème à résoudre (algorigramme ou pseudo-langage) suivi d'une traduction en langage de programmation.

L'indicateur des dispositifs comprend également trois catégories. D'abord, les lieux dans lesquels s'effectuent la programmation informatique. On retrouve la salle de classe et la salle informatique pour les mathématiques, et le laboratoire pour la technologie. L'organisation de l'espace semble être la même pour toutes les disciplines, il s'agit d'alterner les dispositifs en classe entière, et les dispositions en petits groupes ou en ilots. Les interactions entre les différents acteurs sont globalement les mêmes pour toutes les disciplines. Elles vont du travail individuel sans interaction à des tâches collaboratives qui favorisent les échanges entre élèves. La conduite de l'enseignant vis-à-vis de ses élèves est uniquement précisée en mathématiques au cycle 4, avec des indications allant dans le sens d'une intervention minimale consistant à répondre aux questions sans prendre le contrôle des machines des élèves.

Enfin, en ce qui concerne les artefacts, nous avons fait la distinction entre les supports et les cibles. Concernant les artefact-supports permettant de construire les

programmes, nous avons clairement une scission entre le cycle 4 et le lycée à propos des langages de programmation. En effet, au cycle 4 est prescrit l'usage de langage de blocs tels Scratch ou mBlock, alors qu'au lycée les élèves doivent utiliser la modalité textuelle avec le langage Python, et ponctuellement en SNT le duo HTML/CSS. Le matériel permettant de mettre en œuvre les programmes est le même pour toutes les disciplines, il s'agit des ordinateurs fixes ou portables et des tablettes. Les artefacts-cibles, qui sont ceux contrôlés par les programmes informatiques sont uniquement présents en technologie et en SNT. Il s'agit d'objets programmables tels que des robots ou du matériel domotique en technologie et des composants de plus bas niveaux en SNT comme des actionneurs et des capteurs.

2.8.6 Synthèse de la section

Q2.5 : Quelles sont les indications des textes officiels concernant les modalités d'enseignement de la programmation informatique ? Qu'en ressort-il en termes de contenu, d'activité, de dispositif et d'artefact ?

En nous appuyant sur les textes officiels mis à disposition par le ministère de l'Éducation nationale nous avons analysé les **indications** concernant **les modalités d'enseignement de la programmation informatique** dans les trois disciplines qui nous intéressent. Les résultats en fonction des **indicateurs pédagogico-didactiques** sont les suivants :

- concernant les **contenus**, toutes les disciplines opèrent la distinction entre algorithme et programme excepté les mathématiques au cycle 4, il s'agit de ne pas former d'experts dans un langage particulier mais de se concentrer sur les concepts fondamentaux de la programmation qui doivent être introduit progressivement et en réponse à des besoins ;
- concernant les **activités**, la démarche visée est plutôt la résolution d'exercices et de problèmes en mathématiques alors que la technologie et les SNT proposent des démarches d'investigation et des projets plus importants de même, les types d'activités de programmation sont plus divers en mathématiques qu'en technologie et SNT enfin, le mode de programmation est orienté vers les essais-erreurs en blocs en mathématiques au cycle 4 alors qu'il s'agit de passer par une phase de formalisation algorithmique dans les autres disciplines ;
- concernant les **dispositifs**, les lieux de programmation sont le laboratoire pour la technologie et la salle de classe ou la salle informatique pour les autres disciplines, l'organisation de l'espace varie entre la classe entière et les petits groupes d'élèves, les élèves peuvent être amenés à interagir entre eux lors de tâches collaboratives ;
- concernant les **artefacts**, les supports de la programmation sont les langages de blocs au cycle 4 et les langages textuels au lycée, les machines d'édition sont les ordinateurs et tablettes pour toutes les disciplines, les programmes mis au point permettent de contrôler des objets programmables en technologie et SNT.

Nous venons d'exposer notre analyse des indications concernant les modalités d'enseignement de la programmation informatique, il s'agit maintenant de faire le bilan de tous les aspects développés dans ce chapitre afin d'en établir les conséquences pour la transition collège-lycée.

2.9 Discontinuités entre le collège et le lycée

Dans cette dernière partie, nous répondons à la question de recherche Q2.6 qui porte sur les discontinuités dans l'enseignement de la programmation informatique lors de la transition du collège au lycée. Pour ce faire, nous parcourons les résultats établis dans ce chapitre en quête de différences entre ces deux institutions. Nous avançons, à cette occasion, des hypothèses concernant les difficultés qu'elles pourraient causer pour les élèves transitant de l'une à l'autre.

2.9.1 Disciplines hôtes

Concernant les disciplines hôtes de la programmation et les éléments légitimant cette présence, aucune différence claire ne semble se dégager entre le collège et le lycée.

2.9.2 Habitats

Au sujet des habitats de la programmation, on retrouve une distinction entre collège et lycée. Nous avons, du côté du collège, une programmation plutôt déconnectée des autres domaines du programme. Elle est dédiée à des applications ludiques en mathématiques et à la programmation de systèmes techniques en technologie. De côté du lycée, nous retrouvons une programmation intriquée dans tous les domaines des programmes, au service des mathématiques et servant d'outils dans les divers domaines des SNT. Ce passage de la programmation informatique autonome mise en œuvre dans les applications ludiques ou tangibles à une programmation au service des autres domaines du programme dans des activités plus scolaires et abstraites est en mesure d'affecter l'engagement et la motivation des élèves dans les activités (Gopalan et al., 2017).

2.9.3 Concepts ciblés

Les concepts de programmation ciblés sont en partie les mêmes au collège et au lycée. On retrouve en effet un noyau correspondant aux concepts de variable, de structure conditionnelle et de boucles. L'enjeu pour ces concepts réside donc principalement dans le passage d'une implémentation en blocs vers une mise en œuvre sous forme de texte. Deux concepts font leur apparition au lycée, il s'agit du typage des variables et des fonctions.

2.9.4 Niches

Les niches écologiques de certains concepts sont différentes au collège et au lycée. D'abord au niveau des structures de contrôle, les structures conditionnelles sont implémentées avec plus de branches et servent à mettre en œuvre des traitements contingents plus complexes au lycée qu'au collège. De plus, les boucles sont utilisées à des fins itératives au collège, notamment à travers une utilisation plus importante du while. Au lycée, elles donnent lieu à des parcours dans des collections indexées, en particulier grâce aux variables de boucles présentes dans les langages textuels. Nous notons également au lycée l'abandon de la programmation évènementielle remplacée

par une méthode plus linéaire et séquentielle. L'utilisation massive des bibliothèques externes au lycée permet d'étendre les fonctionnalités natives des langages afin de servir les autres domaines des programmes en mathématiques et en SNT. Ces changements, qui vont plutôt dans le sens d'une complexification des usages sont en capacité de mettre en difficulté certains élèves aux acquis fragiles.

2.9.5 Modalités d'enseignement

Enfin, s'agissant des modalités d'enseignement, la différence majeure entre le collège et le lycée se situe au niveau des artefacts-supports utilisés pour produire les programmes : les langages de bloc d'un côté et les langages textuels de l'autre. Cette transition d'une modalité à l'autre est à même de constituer un obstacle cognitif pour les élèves dans la mesure où elle recouvre de nombreux aspects : paradigmes de programmation, sémiotique des instructions, erreurs de programmation, environnements de programmation, etc. (voir Chapitre 3). Les lignes de fracture parcourant les autres indicateurs sont à trouver tantôt entre les mathématiques (cycle 4 et seconde) et le couple technologie/SNT (démarche, type d'activités et programmation d'artefacts-cibles), tantôt entre les mathématiques au cycle 4 et les autres disciplines (distinction algorithme-programme et mode de programmation).

2.9.6 Synthèse de la section

Q2.6 : Quelles discontinuités trouve-t-on dans les indications officielles entre le collège et le lycée ? Quelles difficultés sont-elles susceptibles d'engendrer pour les élèves ?

Dans cette section, nous avons parcouru les résultats établis dans les sections précédentes afin d'en tirer un bilan concernant les différences entre le collège et le lycée, nos constats sont les suivants :

- les **disciplines hôtes** de la programmation informatique et les éléments légitimant sa présence ne présentent pas de différences évidentes ;
- les **habitats** relèvent des différences avec une programmation autonome permettant des activités ludiques et concrètes au collège et des activités en lien avec les programmes, plus scolaires et abstraites au lycée, cela est en mesure d'atténuer l'engagement et la motivation des élèves ;
- les **concepts** de programmation de variable, structure conditionnelle et boucles sont ciblés au collège comme au lycée, leur transposition de la modalité bloc vers texte peut être source de difficultés pour les élèves ;
- les **niches** des concepts des structures de contrôle deviennent plus complexes, la programmation évènementielle et parallèle est abandonnée pour laisser la place à une méthode linéaire et séquentielle et, des bibliothèques externes sont employées afin d'étendre les possibilités des langages ;
- les **modalités d'enseignement** diffèrent surtout concernant les artefacts-supports à utiliser pour produire des programmes, le passage de

> la programmation par blocs à la programmation textuelle cache une diversité de changements à même de poser des difficultés cognitives.

Ce chapitre nous a permis d'analyser différents aspects des textes officiels concernant l'enseignement de la programmation informatique. Après avoir repéré les disciplines hôtes, nous y avons étudié l'habitat de la programmation, avant de cerner les concepts de programmation visés et d'établir leurs niches écologiques. Nous avons ensuite examiné les modalités d'enseignement puis dressé un bilan propre aux discontinuités entre le collège et le lycée.

Il ressort de ce bilan de nombreux éléments liés au passage de la programmation par blocs en Scratch à la programmation sous sa forme textuelle en Python. Le chapitre suivant présente un étude approfondie des différences qui existent entre ces deux modalités de programmation.

Chapitre 3

La transition de Scratch vers Python

La programmation dans un langage de blocs comme Scratch et dans un langage basé sur du texte tel que Python est différente à bien des égards mais partage également certaines similitudes. Cette activité de programmation met en jeu plusieurs aspects allant des caractéristiques inhérentes à chaque langage aux fonctionnalités offertes par leur environnement de programmation. Nous utilisons donc dans ce chapitre le terme *modalité de programmation* pour désigner deux choses. À la fois les aspects intrinsèques des langages de programmation utilisés, mais également les environnements d'édition qui permettent de créer les programmes.

L'objectif de ce chapitre est d'analyser les écarts entre les modalités de programmation Scratch et Python ainsi que leurs conséquences pour les apprenants qui passent d'une modalité à l'autre.

Nous commençons par décliner nos questions de recherche ainsi que la méthodologie que nous avons adoptée (section 3.1), nous présentons ensuite brièvement l'historique et les caractéristiques principales des modalités de programmation Scratch et Python (section 3.2), avant de détailler les aspects relatifs au langages eux-mêmes (section 3.3) puis aux environnements de programmation (section 3.4).

3.1 Questions de recherche et méthodologie

Nous étudions dans ce chapitre la question de recherche Q3 : Quelles sont les différences entre les langages de programmation Scratch et Python ? Quelles sont les conséquences pour les apprenants qui transitent de l'un à l'autre ?

Cette question se décline en plusieurs sous-questions :

- Q3.1 : Quelles sont les différences intrinsèques entre les langages de programmation Scratch et Python ? Quelles sont les conséquences pour les apprenants en transition d'un langage à l'autre ?
- Q3.2 : Quelles sont les différences entre les environnements d'édition des langages de programmation Scratch et Python ? Quels obstacles peuvent rencontrer les apprenants lors du passage de l'un à l'autre ?

Notre méthodologie est ainsi composée de deux aspects. D'abord, une étude approfondie des deux modalités de programmation, puis une analyse des conséquences de la transition de l'une à l'autre.

Concernant le premier point, nous nous appuyons sur des exemples d'utilisation que nous produisons dans le logiciel Scratch 3.0 et dans l'environnement de programmation IDLE (Integrated Development and Learning Environment) fourni avec la distribution standard de Python 3.9. Nous avons choisi IDLE car il dispose des fonctionnalités classiques des éditeurs de code (coloration syntaxique, complétion

automatique, débogueur, etc.). Nous rapportons ces cas d'utilisation à travers la production de nombreuses copies d'écran. Nous basons également nos analyses sur les publications scientifiques des auteurs de Scratch qui détaillent et justifient sa conception (Maloney et al., 2010; Resnick et al., 2003, 2009), et sur des ouvrages généralistes et la littérature scientifique décrivant les divers aspects du langage Python (Lutz, 2013; Martelli et al., 2023; Oliphant, 2007; Pérez et al., 2011).

Pour le deuxième point, nous nous appuyons sur des résultats de recherche afin d'évaluer les conséquences de la transition d'une modalité à l'autre pour les apprenants.

Commençons par planter le décor en exposant brièvement l'origine et l'essentiel des fonctionnalités de Scratch et de Python.

3.2 Présentation des modalités Scratch et Python

Cette section a pour objectif de présenter sommairement les modalités de programmation Scratch et Python en donnant quelques éléments relatifs à leur genèse ainsi qu'un aperçu de leurs principales fonctionnalités.

3.2.1 Scratch : un langage de blocs à visée pédagogique

Scratch[16] est un logiciel développé au MIT à partir de l'année 2003 par l'équipe Lifelong Kindergarten Group dirigée par Mitchel Resnick (Resnick et al., 2003). Sa conception a été largement influencée par les travaux de Seymour Papert dans les années 70 et 80 autour du langage de programmation Logo et de la théorie constructionniste (Papert, 1980). Le projet Scratch visait initialement un public jeune dans les centres périscolaires de communautés économiquement défavorisées (Resnick et al., 2003).

Ce logiciel est à la fois un langage de programmation basé sur des blocs, un environnement de développement, et une plateforme de partage des productions. Comptant plus de 100 millions d'utilisateurs et traduit dans plus de 70 langues (*Scratch statistics*, 2023), c'est actuellement l'un des logiciels éducatifs les plus utilisés au monde.

Scratch a, au-delà de sa niche périscolaire de départ, l'objectif de rendre la programmation informatique accessible à un public très large en offrant un moyen d'expression créatif :

> We wanted to develop an approach to programming that would appeal to people who hadn't previously imagined themselves as programmers. We wanted to make it easy for everyone, of all ages, backgrounds, and interests, to program their own interactive stories, games, animations, and simulations, and share their creations with one another. (Resnick et al., 2009, p. 60)

Comme nous pouvons le voir dans la Figure 3-1, les scripts Scratch sont basés sur une collection de blocs que les utilisateurs peuvent assembler à la manière de briques LEGO®. Ces blocs agissent sur des artefacts numériques nommés *lutins* (ou sprites) tel que l'emblématique chat orange et blanc.

[16] Le nom « Scratch » provient de la technique de scratching utilisée par les DJs de hip-hop qui font tourner des disques vinyles d'avant en arrière en mélangeant des pistes musicales de manière créative.

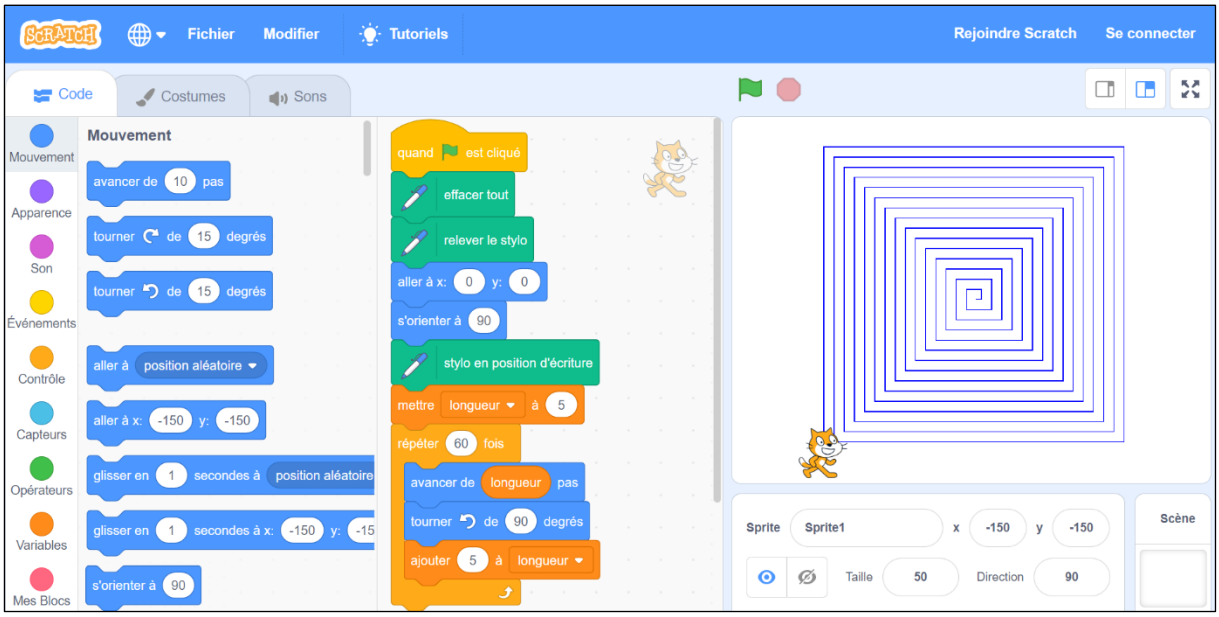


Figure 3-1 – Interface du logiciel Scratch 3.0 et exemple de script

Ces lutins évoluent dans une *scène* sur la partie droite de l'interface. Il existe neuf types de blocs caractérisés par leur couleur selon leur sémantique. Certains types de blocs sont dédiés au contrôle des lutins : « Mouvement », « Apparence », « Son », « Évènements » et « Capteurs » ; d'autres permettent de mettre en œuvre les concepts fondamentaux de la programmation : « Contrôle », « Opérateurs », « Variables » et « Mes Blocs ».

Scratch est conçu pour être interactif. Ainsi, un clic sur le drapeau vert permet de lancer les scripts, il suffit de cliquer sur un ensemble de blocs pour qu'il s'exécute, et les scripts peuvent être modifiés de façon dynamique pendant leur exécution. De plus, la zone centrale d'édition permet de stocker temporairement des blocs ou des morceaux de scripts qui pourront être utilisés ultérieurement dans des scripts.

Présentons maintenant le langage Python et son environnement de développement standard IDLE.

3.2.2 Python : un langage textuel généraliste

Python[17] est un langage de programmation textuel de haut niveau et interprété. Il a été initialement conçu au début des années 90 par Guido Van Rossum au CWI (Centrum voor Wiskunde en Informatica) d'Amsterdam comme un langage de script pour un système d'exploitation distribué (Pérez et al., 2011). Trente ans plus tard, c'est l'un des langages de programmation généralistes les plus utilisés dans l'industrie informatique (Martelli et al., 2023).

Nous présentons dans la Figure 3-2 un exemple de programme Python dans l'interface de l'éditeur IDLE.

[17] Le nom « Python » est inspiré de la série télévisée britannique des années 70 « Monty Python's Flying Circus » appréciée par Guido Van Rossum, l'auteur du langage.

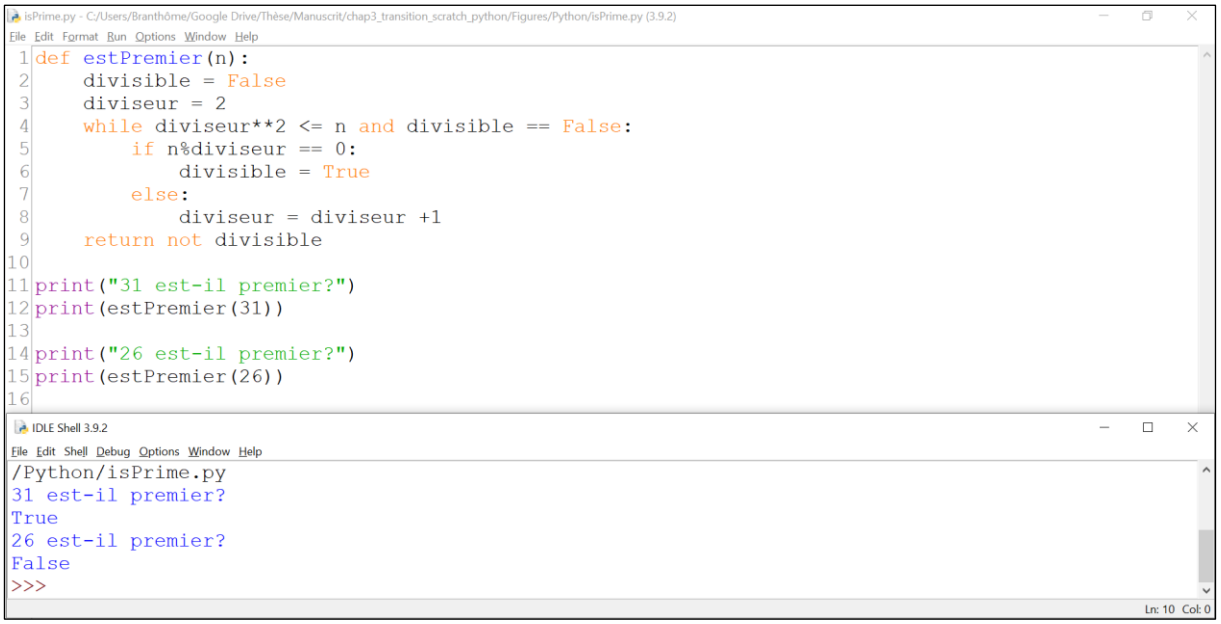


Figure 3-2 – Interface de l'éditeur Python IDLE et exemple de programme

Dans IDLE, les programmes sont rédigés dans une fenêtre de l'éditeur offrant la coloration syntaxique ainsi que la complétion et l'indentation automatique. Ces programmes peuvent ensuite être exécutés dans une autre fenêtre comportant un terminal de type REPL (Read-Eval-Print Loop). L'interprétation est lancée via le menu « Run » de l'éditeur ou directement en ligne de commande via la commande Python « exec ».

Comme nous pouvons le constater dans l'exemple fourni dans la Figure 3-2, la syntaxe de Python est particulièrement compacte car dépourvue de points-virgules, d'accolades, ou de parenthèses pour séparer les différentes instructions et les différents blocs de code. La structuration des programmes est basée sur les retours à la ligne et l'indentation. Sa syntaxe claire et expressive fait dire à Oliphant que Python peut être considéré comme du « pseudo-code exécutable » (2007).

Ajoutons que Python est un langage open-source qui est interprétable sur de nombreuses plateformes. Il est pourvu d'une bibliothèque standard offrant de nombreuses fonctionnalités, et d'un grand nombre de bibliothèques tierces permettant de mettre en œuvre des interfaces graphiques, des traitements de données, des serveurs Web, etc. (Lutz, 2013).

Nous venons de donner un aperçu des modalités de programmation Scratch et Python. Il est déjà possible de remarquer que Scratch est un langage qui a été conçu pour l'enseignement là ou Python a été pensé pour les informaticiens. Mettons désormais la focale sur les différences intrinsèques entre ces deux langages de programmation.

3.3 Différences intrinsèques

Nous traitons dans cette section la question de recherche Q3.1 en analysant les différences propres aux langages de programmation eux-mêmes. Pour ce faire, nous prenons en considération successivement quatre aspects : les paradigmes de

programmation, les concepts de programmation implémentables, les registres sémiotiques, ainsi que les erreurs de programmation.

3.3.1 Paradigmes de programmation

Commençons par analyser les paradigmes de programmation mis en œuvre par Scratch et Python. Bien qu'appartenant à la famille des langages impératifs, ces deux langages implémentent des paradigmes et des méthodes de programmation différentes.

D'abord, Scratch permet une programmation orientée objet. En effet, les scripts sont portés par des artefacts numériques, les lutins, et agissent en modifiant leurs attributs (position, apparence, sons, etc.). Chaque script est nécessairement attaché à un de ces objets et peut donc être considéré comme l'une de ces méthodes. Cette approche objet est pleinement assumée par les concepteurs, même s'ils parlent de langage « basé » sur les objets plutôt que langage « orienté » objet stricto sensu : « Sprites are objects : they encapsulate state (variables) and behavior (scripts). However, since Scratch has neither classes nor inheritance, it is an object-based language but not an object-oriented one » (Maloney et al., 2010, p. 10).

D'autre part, la modalité de programmation Scratch permet la mise en œuvre des méthodes de programmation événementielle et parallèle. Ainsi, par construction, les exécutions de blocs sont inévitablement déclenchées par des événements (clic sur une zone, touche pressée, message reçu, etc.), et cela peut conduire à l'exécution concurrente de plusieurs scripts. Les concepteurs de Scratch prennent l'exemple d'un lutin exécutant trois scripts en parallèle, et justifient l'implémentation de cette fonctionnalité par un monde quotidien baigné dans le multi-tâche :

> Concurrency (or « multi-threading ») is often considered an advanced programming technique. Yet our everyday world is highly concurrent, so Scratch users are not surprised that a sprite can do several things at once. For example, in a simple paddle game, one script might move the ball, another might make it bounce when it hits the paddle, and another might end the game when the player misses the ball. (Maloney et al., 2010, p. 12)

Évoquons désormais la modalité de programmation Python. Le langage de programmation Python est généraliste, il permet d'implémenter les paradigmes orienté objet, procédural et fonctionnel, ainsi que les méthodes de programmation évènementielle et parallèle. Néanmoins, nous avons vu dans la section 2.8.5 que dans le contexte d'introduction qui nous occupe, il est utilisé de manière procédurale et séquentielle. En effet, il s'agit généralement de programmer une fonction principale dans un fichier exécutant une séquence d'instructions appelant éventuellement des sous-fonctions, puis de l'exécuter dans la console Python.

La recherche a montré que les changements de paradigmes et de méthodes de programmation peuvent représenter des obstacles pour les apprenants car ils nécessitent des adaptations cognitives (Khazaei & Jackson, 2002; White & Sivitanides, 2005).

Au-delà des paradigmes de programmation employés, étudions maintenant les concepts fondamentaux de la programmation qui peuvent être implémentés dans chacune des deux modalités.

3.3.2 Implémentation des concepts fondamentaux

Nous nous intéressons ici à l'implémentation des concepts fondamentaux de la programmation. Nous partons pour cela du corpus de concepts que nous avons établi dans la section 1.5.3. Le Tableau 3-1 synthétise les analyses que nous détaillons à la suite en indiquant les possibilités de mise en œuvre des différents concepts.

Concepts fondamentaux	Scratch	Python
Variables	✓	✓
Types primitifs : nombres, caractères, booléens	Nombres (ovales) Booléens (hexagones) Pas de type caractère	Nombres entiers (int) Nombres flottants (float) Booléens (bool) Pas de type caractère
Structure conditionnelle	✓	✓
Boucle For	✓	✓
Boucle While	✓	✓
Entrées-sorties programmes	✓	✓
Fonctions et paramètres	✓	✓
Récursivité	Partielle	✓
Type composites simples : **chaînes de caractères**, tableaux et enregistrements	Chaînes indifférenciées des nombres (type ovale) Pas de tableau ni d'enregistrement	Chaînes des caractères (str) Tableaux (tuple) Enregistrements (dict)
Type composites complexes : listes	Listes	Listes (list)

Tableau 3-1 – Possibilités d'implémentation des concepts fondamentaux de la programmation dans les langages Scratch et Python, en gras, les concepts ciblés à la transition collège-lycée

Nous donnons ci-dessous des précisions concernant les implémentations de chaque concept dans les deux langages.

Les variables sont implémentables dans les deux langages. Du côté de Scratch, leur cycle de vie commence par leur déclaration via une fenêtre de dialogue (voir Figure 3-3-a) qui pour effet de les initialiser à la valeur zéro. Il est ensuite possible de les affecter ou de les incrémenter à l'aide de blocs appropriés (voir Figure 3-3-b).

En Python, les variables ne sont pas déclarées explicitement, la première affectation fait office de déclaration et d'initialisation (voir Figure 3-3-c). L'incrémentation fait également l'objet d'une syntaxe particulière à l'aide de l'opérateur « += ». Notons au passage que la déclaration de variable en Scratch met en jeu la notion de portée : elles peuvent être déclarées locales au lutin (« pour ce sprite uniquement ») ou globales au projet (« pour tous les sprites »). En Python, c'est le lieu de sa première affectation (dans une fonction, dans un boucle, au début d'un script, etc.) qui définit implicitement la portée d'une variable.

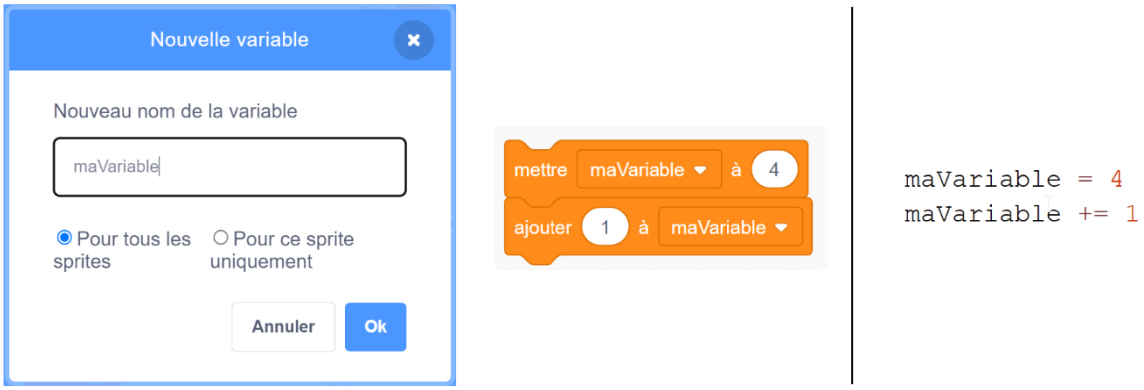


Figure 3-3 – Mise en œuvre des variables en Scratch (a et b) et en Python (c)

Ensuite, les types primitifs sont de deux sortes en Scratch : forme ovale pour les nombres, et forme hexagonale pour les booléens. Cependant, comme l'illustre la Figure 3-4-a, les variables ne peuvent être que de type ovale (entier, flottant[18], etc.). Elles ne peuvent pas prendre le type booléen. Ce type est réservé aux expressions logiques que l'on retrouve imbriquées dans les structures conditionnelles et les boucles while. Ces expressions logiques sont formées de blocs retournant des booléens (« touche espace pressée ? », « couleur blanche touchée ? », etc.) et d'opérateurs logiques (« > », « < », « = », « et », « ou », etc.). En Python, comme le montre la Figure 3-4-b, nous retrouvons les types : nombres entiers (int), nombres flottants (float) et booléens (bool). Notons qu'aucun des deux langages ne considère les caractères comme un type à part entière[19].

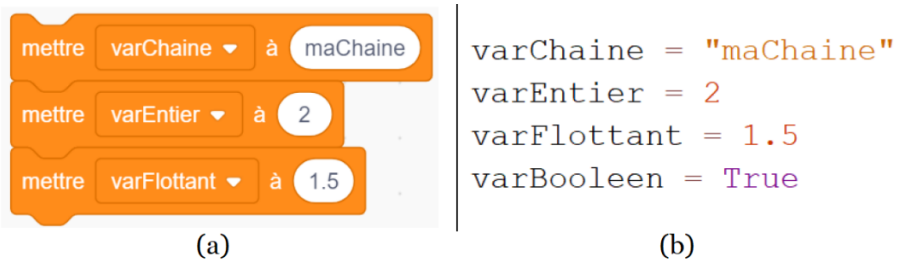


Figure 3-4 – Les différents types des variables en Scratch (a) et Python (b)

Les structures conditionnelles sont implémentables en Scratch. Au-delà de la forme à deux branches « si-alors-sinon » présentée dans la Figure 3-5-a, ce langage propose une forme à une branche « si-alors ». Ajoutons que pour mettre en œuvre des structures à plus de deux branches, il est nécessaire d'imbriquer plusieurs blocs « si-alors-sinon ». En Python (voir Figure 3-5-b), les mots-clés « if », « elif » et « else » permettent d'implémenter des traitements conditionnels à une ou plusieurs branches.

[18] En informatique, le type des nombres à virgule flottante (ou flottants) permet de représenter les nombres décimaux et plus largement une approximation de tout nombre réel.
[19] Il est cependant toujours possible en Python de stocker un unique caractère dans une chaine.

(a)

```
var1 = 4
if var1 < 0:
    print("Négatif")
else:
    print("Positif")
```

(b)

Figure 3-5 – Mise en œuvre de la structure conditionnelle en Scratch (a) et Python (b)

Les boucles bornées (for) existent dans la modalité Scratch. Elles permettent de répéter des instructions un certain nombre de fois, notons qu'il n'y a pas de variable de boucle gérée automatiquement (voir Figure 3-6-a). En revanche en Python, la boucle bornée s'appuie sur des séquences de nombres retournées par la fonction « range() » qui servent également à mettre à jour la variable de boucle (voir Figure 3-6-b).

(a)

```
for var1 in range(10):
    print(var1)
```

(b)

Figure 3-6 – Mise en œuvre de la boucle for en Scratch (a) et Python (b)

Les boucles non-bornées (while) sont mises en œuvre en Scratch à travers le bloc « répéter jusqu'à » et, en Python, par l'utilisation du mot clé « while » (voir Figure 3-7). Remarquons que la condition d'arrêt est inversée en Scratch « jusqu'à ce que » contre « while » (tant que) en Python.

(a)

```
var1 = 10
while var1>0:
    print(var1)
    var1 = var1-1
```

(b)

Figure 3-7 – Mise en œuvre de la boucle while en Scratch (a) et Python (b)

Comme nous pouvons le voir dans la Figure 3-8, les entrées-sorties sont réalisables en Scratch en utilisant les blocs « demander » et « dire ». Cet interfaçage des programmes s'effectue via les lutins qui posent des questions et affichent les sorties dans des bulles de dialogues. Le résultat de l'entrée n'est pas un retour du bloc « demander », il est stocké dans une variable particulière nommée « réponse ». En Python, il s'agit d'utiliser les fonctions « input » et « print » qui utilisent les entrées et sorties standards de la console.

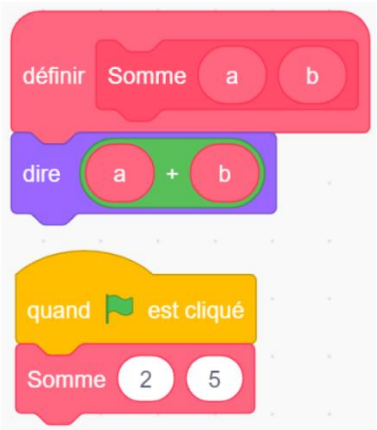
(a)

```
var1 = input("Entrée?")
print(var1)
```

(b)

Figure 3-8 – Mise en œuvre des entrées-sorties en Scratch (a) et Python (b)

Les fonctions sont implémentables dans le langage Scratch via les blocs personnalisables (« Mes blocs »). Comme illustré dans la Figure 3-9, ces blocs peuvent prendre en entrée des paramètres (de type ovale ou hexagonal) mais ne sont pas capables de retourner une valeur. Ces fonctions ne peuvent agir que par « effet de bord[20] » en exerçant une action sur des lutins ou des variables globales. Les blocs créés peuvent ensuite être utilisés dans les scripts de façon traditionnelle en les combinant avec d'autres blocs. En Python, les fonctions peuvent prendre des paramètres et retourner des valeurs, elles sont déclarées à l'aide du mot-clé « def » puis appelées à l'aide de leur identifiant.



(a)

```
def somme(a,b):
    print(a+b)

somme(2,5)
```

(b)

Figure 3-9 – Mise en œuvre d'une fonction en Scratch (a) et Python (b)

Dans le langage Scratch, les appels récursifs de fonction sont possibles. Cependant, étant donné que les fonctions ne peuvent pas retourner de valeur, elles ne permettent pas la mise en œuvre complète de la programmation récursive. Il est néanmoins possible, comme nous l'illustrons dans le script reproduit dans la Figure 3-10, d'implémenter des algorithmes récursifs en passant par des listes afin d'entreposer les résultats partiels des différents appels récursifs.

[20] Une fonction est dite à « effet de bord » (traduction de l'anglais « side effect ») si elle modifie un état extérieur autre que sa valeur de retour.

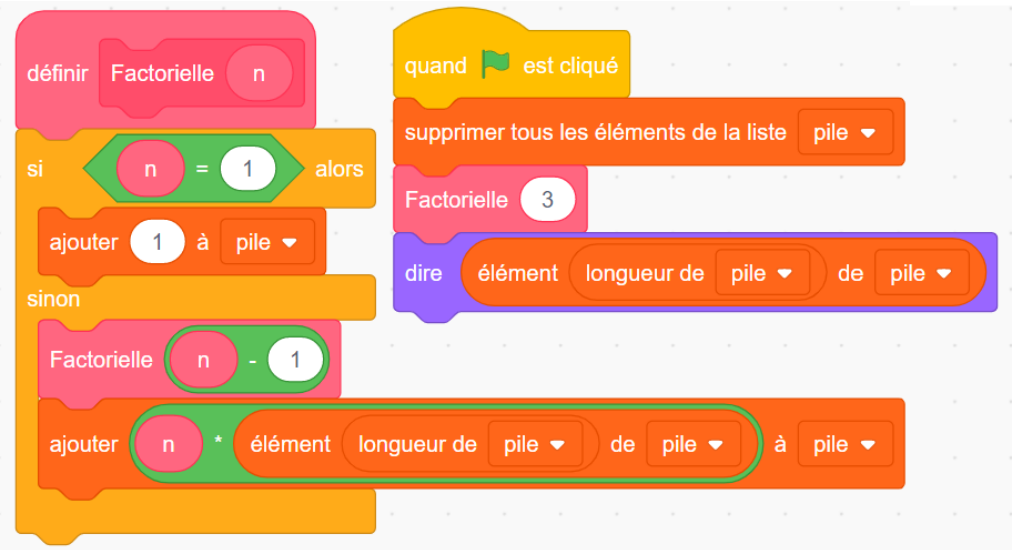


Figure 3-10 – Programme récursif de la factorielle en Scratch

Concernant les types composites simples, les chaînes de caractères sont indifférenciées des nombres en Scratch. C'est-à-dire qu'une variable de type ovale peut contenir un nombre comme une chaîne de caractères (voir Figure 3-4-a). Dans le langage Python, les chaînes de caractères constituent un type à part entière (str). Les structures de données tableau et enregistrement ne sont pas implémentables en Scratch. Python donne la possibilité de mettre en œuvre les tableaux dans des n-uplets (tuple) et les enregistrements dans des dictionnaires (dict).

Enfin, le type composite complexe des listes est implémenté dans Scratch comme dans Python par une structure de données dédiée.

Nous tirons ci-dessous le bilan de ces analyses des différences d'implémentation des concepts entre les langages Scratch et Python. Nous nous focalisons à cette occasion sur les concepts visés à la transition collège-lycée (concepts listés dans la section 2.6.1 et repérés en gras dans le Tableau 2-6). Pour chaque concept, nous conjecturons les conséquences que peuvent avoir ces écarts sur les élèves lors de la transition Scratch-Python.

D'abord, la déclaration des variables ainsi que le choix de leur portée sont des passages obligés du langage Scratch. En Python, ces deux étapes sont implicites lors de la première affectation des variables. La transformation du mode opératoire de création des variables et le caractère tacite de certaines informations peuvent représenter pour les élèves un obstacle épistémologique au sens de Bachelard (1938). Ce type d'obstacles est défini comme un élément de connaissance de l'élève qui s'est avéré satisfaisant dans un certain contexte pour résoudre certains problèmes, mais qui, hors de ce contexte, face à de nouveaux problèmes, s'avère inadéquat et difficile à adapter.

En ce qui concerne les types, Scratch ne fait aucune distinction au sujet de ses variables qui peuvent contenir des nombres comme des chaînes de caractères, là où ils font l'objet de trois types différents en Python. De plus, les variables de type booléen n'existent pas en Scratch contrairement à Python. Il est probable que cette segmentation en différents types puisse également constituer un obstacle épistémologique pour les élèves. Nous avons montré dans la section 2.6.1 que

l'introduction du concept de type était l'un des objectifs du lycée, nous pouvons affirmer que ce but est cohérent avec les langages prescrits.

S'agissant des structures de contrôle, les fonctionnalités sont les mêmes pour les conditionnelles. Les boucles for ont la seule fonction de répétition en Scratch et offrent en plus la gestion d'une variable de boucle en Python. Nous avons vu dans la section 2.7.6 que cette possibilité ouvrait de nouvelles niches telles que les parcours de collection au lycée. Les boucles while ont des conditions d'arrêt inversées en Scratch et Python, cela peut s'avérer être un obstacle épistémologique pour les apprenants lors du passage d'un langage à l'autre. Enfin, les fonctions prennent des paramètres dans les deux langages mais ne sont pas en mesure de retourner de valeurs en Scratch. Il existe néanmoins en Scratch des blocs de type « capteurs » qui retournent des valeurs (« volume sonore », « souris x », « distance de », etc.) mais ils prennent directement la forme de variables. Même si les fonctions ne sont pas au programme du cycle 4 et constituent un des objectifs du lycée, il est probable que le concept de retour soit plus difficile à appréhender que celui de paramètre, dans la mesure où les blocs Scratch agissent tous par effet de bord, c'est-à-dire sans retour.

Nous avons analysé dans cette sous-section l'implémentation des différentes notions de la programmation ainsi que les écarts fonctionnels entre les langages. Nous évoquons dans la sous-section suivante les différences sémiotiques, c'est-à-dire liées à la forme des commandes.

3.3.3 Registres sémiotiques

La distinction entre un concept et ses représentations est un élément important pour la compréhension et la conceptualisation des objets théoriques. Duval (1993) nomme *registres sémiotiques* des systèmes structurés de signes permettant de représenter des concepts. Ces registres permettent en particulier d'effectuer des traitements et des conversions d'un registre à l'autre.

La programmation en Scratch et la programmation en Python font appel à des registres sémiotiques bien différents pour implémenter les concepts de la programmation. Nous les nommons respectivement *registre des blocs* et *registre des instructions*. Le registre des blocs est constitué de formes graphiques de différentes couleurs contenant des mots-clés et des locutions en français (« tourner », « dire », « quand ce sprite est cliqué »), des listes déroulantes, des champs de saisie de texte et de chiffres. Alors que le registre des instructions est composé uniquement d'éléments textuels. Martelli et ses collègues (2023) précisent que Python comporte 35 mots-clés en anglais (« and », « return », « for », etc.). Le langage compte également 23 opérateurs (« + », « % », « ** », etc.) et 24 délimiteurs (« { », « [», « ; », etc.) qui sont des symboles mathématiques et typographiques. Ajoutons que les tabulations et les retours à la ligne ont également une importance capitale en Python dans la mesure où ils définissent la structure des programmes.

La conversion d'un registre à l'autre peut être plus ou moins immédiate et dépend de la *congruence sémiotique* (Duval, 1993) des différentes représentations d'un même concept. Dans un travail précédent (Branthôme, 2021), nous avons analysé finement ces congruences pour différents concepts de programmation. Nous avons établi que cette congruence est forte pour les variables, les structures conditionnelles et les fonctions, mais qu'elle est plus faible pour les boucles, en particulier pour les boucles

for. Pour cette raison, des connaissances préalables en Scratch peuvent être une aide pour implémenter les concepts de variables, conditionnelles et fonctions en Python, et plutôt un obstacle pour la mise en œuvre des concepts de boucles dans la modalité texte.

De plus, la recherche montre que les spécificités de la programmation textuelle sont particulièrement problématiques pour les novices. Ainsi Stefik et Siebert (2013) ont démontré qu'un langage constitué de mots-clés générés aléatoirement n'est pas plus intuitif et facile à comprendre pour des débutants que la plupart des langages textuels (Java, Python, etc.). De plus, selon Bau et ses collègues (2017), la syntaxe dense de ces langages peut constituer une difficulté pour les néophytes car elle est en mesure de provoquer une surcharge cognitive (Sweller, 1988). Par contraste, le registre des blocs aide les apprenants en leur montrant comment appréhender les commandes en plus grands morceaux (Bau et al., 2017). Ajoutons que les mots-clés et les locutions utilisés dans le registre des blocs tendent à être plus proches du langage naturel en imitant partiellement leur grammaire (Kölling et al., 2015; Weintrop, 2019). L'exemple de l'incrémentation fourni dans la Figure 3-3 est parlant , « ajouter 1 à maVariable » en Scratch est à comparer à « maVariable += 1 » en Python. La langue utilisée a aussi un impact sur les apprenants. Ainsi Scratch propose une traduction de ses blocs dans plus de 70 langues, là où les mots-clés de Python sont uniquement en anglais. Or, des études ont montré que les instructions en anglais constituent un obstacle supplémentaire pour les étudiants dont ce n'est pas la langue maternelle (Guo, 2018).

Néanmoins, le registre des instructions a aussi des avantages dès lors que les productions deviennent plus importantes. En effet, les instructions prennent moins de place à l'écran en comparaison des blocs (densité supérieure), et il est plus facile de modifier, d'organiser, de naviguer et de faire des recherches dans un ensemble de programmes prenant la forme de textes (Bau et al., 2017).

Nous venons d'exposer les différences entre Scratch et Python concernant la forme des commandes, nous abordons dans la sous-section suivante les divergences relatives aux erreurs de programmation.

3.3.4 Erreurs de programmation

L'activité de programmation va de pair avec d'inévitables erreurs. La plupart des langages connaissent trois niveaux d'erreurs : *lexicales*, *syntaxiques* et *sémantiques* (Scott, 2015). Pour les langages compilés (comme C ou Java), les erreurs lexicales et syntaxiques sont détectées par le compilateur avant l'exécution. Ces deux premiers types d'erreurs consistent à ne pas respecter le lexique ou la grammaire d'un langage. Par exemple, une erreur lexicale consisterait à écrire « fi » à la place de « if » et une erreur syntaxique à programmer la ligne suivante : « maVariable = 1 = 2 ». Les erreurs sémantiques sont liées à la signification des programmes, elles sont, pour la plupart, décelées à l'exécution du code, lorsque les données d'entrées sont connues. Nous pouvons par exemple évoquer la division par zéro ou une tentative d'accès en dehors des bornes d'un tableau. Lorsque les langages de programmation sont interprétés (comme Python ou Javascript), les trois types d'erreurs sont, en l'absence de compilation, mises en évidence lors de l'interprétation du programme.

Les contrôles liés au typage des données peuvent varier d'un langage à l'autre. Un langage de programmation peut avoir un typage *fort* ou *faible* (Scott, 2015). Un langage

est dit fortement typé s'il empêche l'application d'une opération à un objet qui ne la supporte pas. À l'inverse, il est faiblement typé s'il autorise l'application d'opérations à des opérandes d'un type non supporté. Citons par exemple la division d'un entier par une chaîne de caractères. Pour les langages fortement typés, il est possible d'établir une autre distinction : le typage peut être *statique* ou *dynamique* (Gabbrielli & Martini, 2010). Dans les langages typés statiquement, comme C ou Java, les variables doivent être déclarées avec un certain type qui ne peut changer au cours de l'exécution du programme. De plus, le compilateur vérifie que toutes les opérations sont conformes du point de vue du type avant exécution. Pour les langages typés dynamiquement, comme Javascript ou Python, le type des variables n'est pas déclaré, il est implicite et déduit de l'affectation des variables au moment de l'exécution. De plus, ce type peut changer à chaque affectation en cours d'exécution (d'où la dénomination dynamique). Cette vérification dynamique entraîne néanmoins une surcharge au niveau de l'exécution et déporte la découverte des erreurs à l'exécution. Cet inconvénient peut être considéré comme mineur par rapport à l'augmentation potentielle de la productivité des programmeurs du fait de l'allègement de la syntaxe (Scott, 2015).

Analysons maintenant les erreurs de programmation dans les langages Scratch et Python à l'aune de ces caractéristiques faible/fort et statique/dynamique.

Les systèmes basés sur des blocs comme Scratch permettent d'éviter totalement les erreurs lexicales et syntaxiques. En effet, l'assemblage de blocs prédéfinis assure la correction lexicale et le maintien de la grammaire du langage. En effet, les concepteurs de Scratch se sont inspirés des briques de construction LEGO® :

> When people play with Lego bricks, they do not encounter error messages. Parts stick together only in certain ways, and it is easier to get things right than wrong. The brick shapes suggest what is possible, and experimentation and experience teaches what works. [...] Similarly, Scratch has no error messages. Syntax errors are eliminated because, like Lego bricks, blocks fit together only in ways that make sense. (Maloney et al., 2010, p. 5-6)

De façon plus surprenante, Scratch a également été conçu pour ne pas signaler les erreurs sémantiques. Comme nous pouvons le voir dans le script proposé dans la Figure 3-11, une division par zéro mène à la valeur « Infinity », et l'accès à un élément en dehors d'une structure de données de type liste n'entraîne ni erreur ni action.

Figure 3-11 – Programme en Scratch contenant des erreurs sémantiques et affichages des sorties

Ainsi, la volonté des créateurs du langage était que chaque bloc agisse sans blocage même en cas d'erreur sémantique à l'exécution : « Scratch also strives to eliminate run time errors by making all blocks be *failsoft*. Rather than failing with an error message, every block attempts to do something sensible even when presented with out-of-range inputs [mis en valeur par les auteurs] » (Maloney et al., 2010, p. 6). Les auteurs précisent que l'absence de messages d'erreur n'assure pas la correction d'un programme. Ce parti pris d'exécution coûte que coûte doit permettre au programmeur d'avoir toujours un programme qui fonctionne :

> Of course, eliminating error messages does not eliminate errors. The user must still think carefully to write scripts that do what they want and must trouble-shoot scripts that do not work as expected. However, even when a script does not do the right thing, it does something, and that is a good start. A program that runs, even if it is not correct, feels closer to working than a program that does not run (or compile) at all. (Maloney et al., 2010, p. 6)

Toujours à propos du typage, nous retrouvons dans Scratch une situation hybride. Nous avons, d'un côté, un typage statique et fort concernant la distinction entre les expressions booléennes représentées par des hexagones et les autres expressions (entier, flottant et chaînes de caractères) représentées par des ovales. Ainsi, il n'est pas possible de commettre une erreur de typage concernant les booléens. Pour illustrer, dans le script présenté dans la Figure 3-12, il est impossible d'introduire autre chose qu'une expression booléenne entre le « si » et le « alors » de la structure conditionnelle. Il n'est pas non plus possible d'utiliser un opérande booléen dans les deux opérations d'addition. La situation est bien différente pour les variables. D'abord, étant de type ovale, elles peuvent contenir indifféremment des entiers, des flottants ou des chaînes de caractères. Autrement dit, comme nous l'avons vu dans la section 3.3.2, aucun type n'est précisé lors de la création d'une variable. Ce fonctionnement relève du typage dynamique. De plus, aucune erreur d'incohérence de type n'est signalée à l'exécution, c'est la marque d'un typage faible. Le programme reproduit dans la Figure 3-12 en est une bonne illustration.

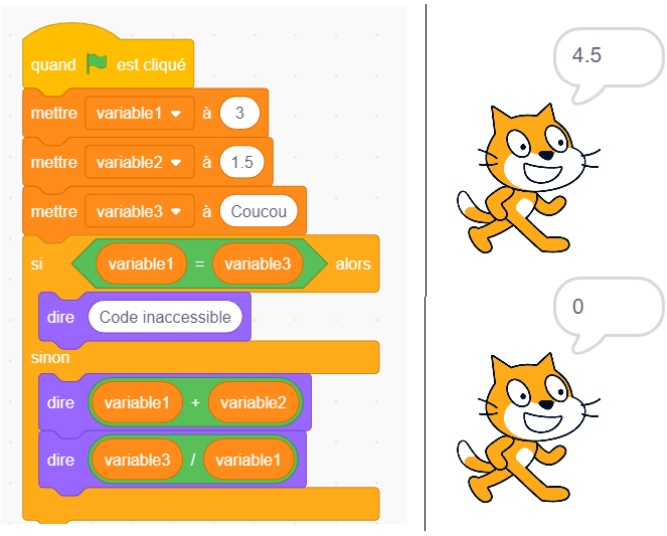


Figure 3-12 – Programme en Scratch comprenant des erreurs de typage et affichage des sorties

En effet, l'addition d'un entier et d'un flottant ne pose aucun problème et la division d'une chaîne de caractères par un entier ne provoque pas d'erreur et donne même un résultat. Pour parer à ces incohérences, des conversions implicites sont effectuées. Par exemple, l'addition d'un entier et d'un flottant donne un flottant, et la division d'une chaîne de caractères par un entier retourne un entier. Nous retrouvons ici l'objectif d'exécution à tout prix voulu par les concepteurs de Scratch. Au final, Scratch possède un typage mixte, d'un côté, fort et statique pour les booléens, et de l'autre, faible pour les autres types mis en œuvre dans les variables. Ce choix de typage faible des variables relève aussi d'une intention de simplification pour les débutants :

> A Scratch variable can hold values of any data type. This avoids requiring that the user specify the type of a variable when it is created. Scratch automatically converts between numbers and strings depending on context. For example, if the string « 123 » is passed to an arithmetic operation, it is converted to a number, while if a number is passed to the « Say » command, it is converted to a string. (Maloney et al., 2010, p. 9-10)

Passons à l'analyse des erreurs dans le langage Python. Rappelons que Python est un langage interprété et que, comme le montre la Figure 3-13, les trois types d'erreurs sont détectés lors de l'interprétation des programmes.

```
# Erreur lexicale
fi 3 == 3:
    print("Foo")
# Erreur syntaxique
maVariable = 1 = 2
# Erreurs sémantiques
monNombre = 12/0
monTab = (1,2,3)
elt = monTab[4]
```

```
SyntaxError:
invalid syntax


SyntaxError:
cannot assign to literal

ZeroDivisionError:
division by zero

IndexError:
tuple index out of range
```

Figure 3-13 – Programme en Python contenant des erreurs lexicales, syntaxiques et sémantiques et messages d'erreur

Dans cet exemple, l'incorrection d'un mot-clé, la formulation d'une expression qui viole la grammaire du langage, ou des erreurs sémantiques provoquent toutes un arrêt de l'interprétation et l'affichage d'un message d'erreur explicatif.

D'autre part, comme l'explique Lutz, Python est un langage au typage fort et dynamique : « Python is *dynamically typed* (it keeps track of types for you automatically instead of requiring declaration code), but it is also *strongly typed* (you can only perform on an object operations that are valid for its type) [mis en valeur par l'auteur] » (2013, p. 68). Comme nous l'avons déjà dit, il n'est donc pas nécessaire de déclarer le type des variables, ce type est implicite et déduit des affectations par l'interpréteur au moment de l'exécution. Nous en donnons un exemple dans le programme reproduit dans la Figure 3-14 : les variables de types différents (float, str et bool) sont simplement affectées sans déclaration de type.

```
var1 = 1.5
var2 = "Foo"
var3 = True
print(var1+var2)
print(var2/var3)
```

```
TypeError: unsupported operand type(s)
for +: 'float' and 'str'

TypeError: unsupported operand type(s)
for /: 'str' and 'bool'
```

Figure 3-14 – Programme en Python contenant des erreurs de typage et messages d'erreur

Bien que dynamique, le typage est néanmoins fort. Ainsi, les messages d'erreurs reproduits dans la Figure 3-14 signalent des opérandes du mauvais type pour les opérations d'addition et de division. Ce typage fort et dynamique a pour avantage une certaine souplesse d'utilisation mais il rend plus difficile la détection des erreurs de type lors de la mise au point des programmes. Ce choix d'un langage typé dynamiquement peut être discutable d'un point de vue didactique dans un niveau scolaire où le typage est un objectif d'apprentissage. Il aurait sans doute été plus avantageux de prescrire l'utilisation d'un langage au typage statique qui favoriserait la prise de conscience du typages des variables par les débutants.

Pour résumer, concernant les erreurs, Scratch a été conçu pour ne produire aucun message d'erreur, ce qui ne garantit pas la correction des programmes. En revanche, Python détecte lors de l'interprétation les erreurs lexicales, syntaxiques et sémantiques et produit des messages d'erreurs explicatifs. Au niveau du contrôle des types, Scratch propose à la fois un typage fort et statique pour les expressions booléennes et un typage faible pour les variables, ce qui permet de simplifier la syntaxe et de ne pas signaler les erreurs de typages. De son côté, Python offre un typage fort et dynamique, qui soustrait le programmeur à la tâche de spécification des types mais qui est en mesure de lever des erreurs à l'exécution.

Lors du passage d'une modalité de programmation à l'autre, les élèves sont donc amenés à découvrir les erreurs de programmation et les messages qui les accompagnent. La recherche a montré que les novices en programmation font d'abord face à beaucoup d'erreurs de syntaxe (Denny et al., 2011), puis qu'ils tendent à faire plus d'erreurs sémantiques qui sont beaucoup plus longues à corriger (Altadmri & Brown, 2015). De plus, il a été établi que les messages d'erreur n'ont pas une formulation claire pour les étudiants (McCall & Kölling, 2014), qu'il peuvent être frustrants, intimidants ou déroutants, et qu'ils ne sont pas toujours lus par les étudiants qui ne leur font pas forcément confiance (Becker et al., 2019). Nous savons également que, pour les débutants, apprendre à interpréter ces messages est coûteux et demande beaucoup de pratique (Becker, 2016).

3.3.5 Synthèse de la section

Q3.1 : Quelles sont les différences intrinsèques entre les langages de programmation Scratch et Python ? Quelles sont les conséquences pour les apprenants en transition d'un langage à l'autre ?

En nous appuyant sur des cas d'utilisation et sur la littérature scientifique, nous avons montré que les langages de programmation Scratch et Python sont différents selon plusieurs aspects :

- concernant les **paradigmes de programmation**, Scratch met en œuvre la programmation orientée objet, événementielle et parallèle alors que Python est utilisé au lycée de manière procédurale et séquentielle, le passage de l'un à l'autre nécessite des adaptations cognitives ;
- concernant les **concepts implémentables**, l'étape de déclaration des variables nécessaire en Scratch n'est pas faite en Python, les variables ne sont pas typées en Scratch au contraire de Python qui distingue trois types différents, les boucles for disposent en Python de variables de boucles qui n'existent pas en Scratch et la condition d'arrêt des boucles while est inversée entre les deux langages, les fonctions agissent uniquement par effet de bord en Scratch et permettent de retourner des valeurs en Python, ces différences conceptuelles peuvent constituer des obstacles épistémologiques (Bachelard, 1938) en mesure de mettre les apprenants en difficulté ;
- concernant les **registres sémiotiques**, les scripts Scratch sont constitués d'éléments graphiques de différentes formes et couleurs ainsi que de locutions françaises et les programmes Python de mots-clés en anglais et de symboles typographiques, le passage d'une forme à l'autre peut poser problème pour les boucles (faible congruence), la forme textuelle des programmes Python est particulièrement problématique pour les novices dans la mesure où elle est dense, peu intuitive, éloignée du langage naturel dans ses formulations et dans une langue étrangère ;
- concernant les **erreurs de programmation**, pour des raisons de simplification d'usage, Scratch ne produit aucun message d'erreur et les variables y sont faiblement typées ; de son côté, Python signale, lors de l'interprétation, tous les types d'erreurs en maintenant un typage fort et dynamique pour ses variables, ces écarts mènent les débutants en Python à commettre de nombreuses erreurs et à être confrontés à des messages d'erreurs mal formulés et déroutants qu'ils ont du mal à interpréter.

Après avoir détaillé les aspects liés aux caractéristiques intrinsèques des langages, comparons maintenant leurs environnements d'édition.

3.4 Différences liées aux environnements

Nous répondons dans cette section à la question de recherche Q3.2 ayant trait aux différences concernant les environnements d'édition des programmes dans les modalités Scratch et Python. À cet effet, nous examinons trois aspects de ces environnements : la présence d'un catalogue de commande, le procédé de composition des programmes, et la visibilité et le contrôle de l'exécution.

3.4.1 Catalogue de commande

Commençons par nous intéresser aux catalogues de commande. Comme le montre la Figure 3-15, l'environnement de développement Scratch comprend une zone fixe listant l'ensemble des blocs utilisables sur le côté gauche de son interface.

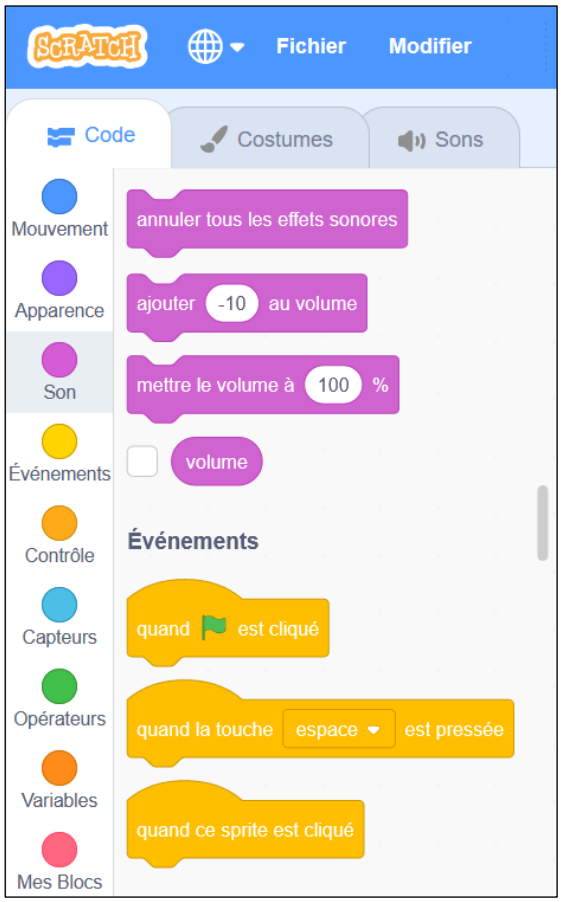


Figure 3-15 – Détail de la partie gauche de l'interface de l'éditeur Scratch comprenant un catalogue de blocs

Ce catalogue est composé d'un menu qui présente les neuf catégories de blocs à travers des pastilles de couleurs et d'une liste qu'il est possible de faire défiler à l'aide d'un ascenseur. Un clic sur une pastille permet d'accéder directement à l'ensemble des blocs de sa catégorie dans la liste. Ce catalogue est visible à tout moment afin de rester accessible au programmeur et pour encourager l'expérimentation (Maloney et al., 2010). Il permet aux novices de parcourir toutes les commandes disponibles afin de se rappeler des fonctionnalités dont ils auraient un souvenir partiel ou de s'inspirer dans la découverte de nouvelles possibilités (Kölling et al., 2015). L'organisation des blocs par thèmes et par concepts simplifie la découverte et l'exploration (Bau et al., 2017; Weintrop, 2019).

Concernant la modalité Python, l'environnement IDLE propose d'accéder à la documentation officielle du langage depuis le menu « Help ». Cette documentation est visible dans la Figure 3-16.

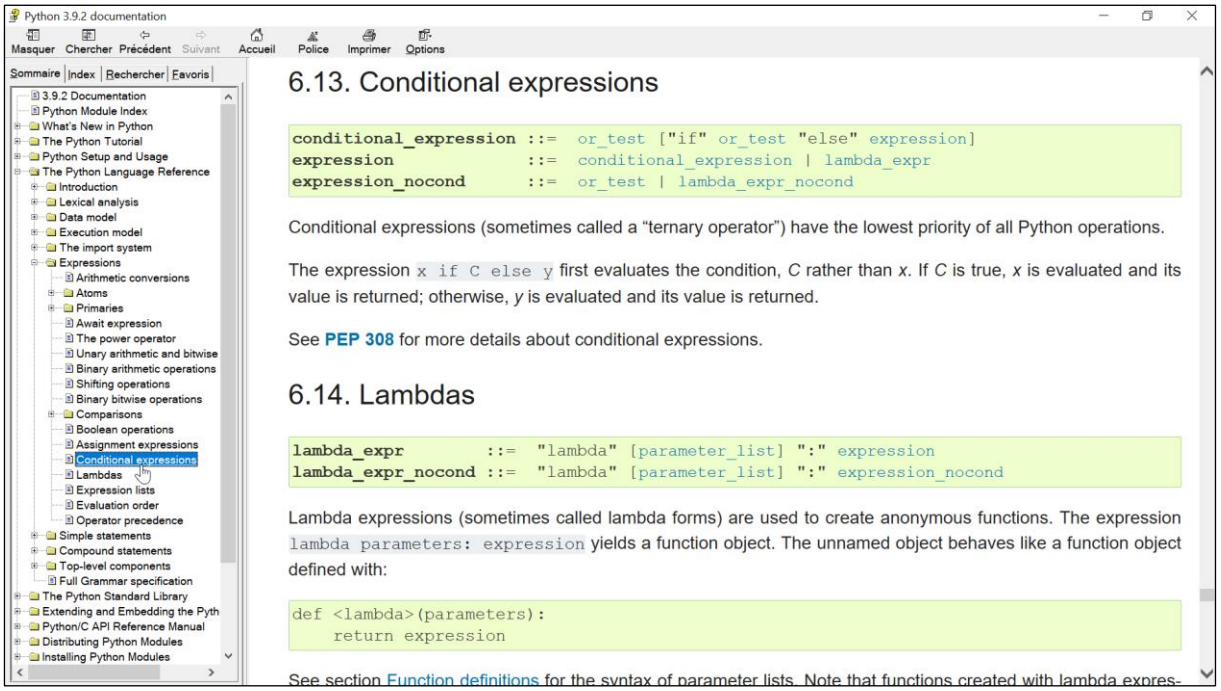


Figure 3-16 – Documentation officielle de Python accessible depuis IDLE

Cette interface est constituée de différentes rubriques, accessibles via un sommaire, un index ou une barre de recherche. Ces rubriques expliquent en anglais et de manière détaillée le fonctionnement de Python. Il est clair, au regard de son caractère exhaustif, que cette documentation s’adresse à un lectorat d’utilisateurs avancés qui seraient à la recherche de détails techniques plutôt qu’à un public de débutants souhaitant découvrir les fonctionnalités du langage. Comme l’affirment Kölling et ses collègues (2015), l’exploration et la navigation dans ce type de documentations importantes nécessitent un apprentissage supplémentaire pour les programmeurs débutants.

L’environnement de développement IDLE propose également la complétion automatique du code. Présentée dans la Figure 3-17, cette fonctionnalité est en mesure de fournir une aide au programmeur par l’intermédiaire d’une liste contextuelle d’instructions susceptibles de parachever celle en cours d’édition.

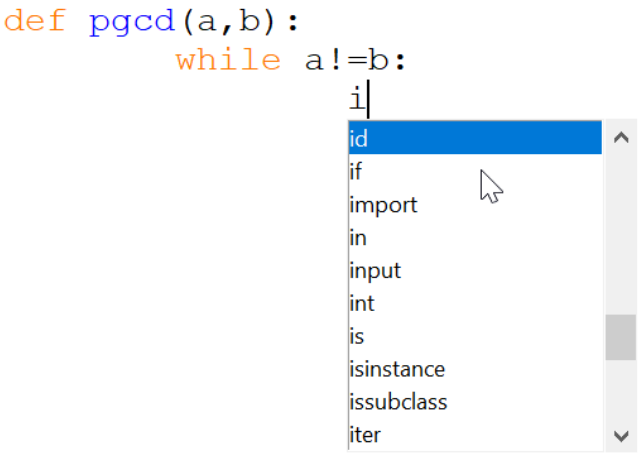


Figure 3-17 – Complétion automatique disponible dans l’environnement IDLE

Toutefois, en comparaison avec la modalité Scratch, ces informations ne sont pas persistantes et sont classées par ordre alphabétique et non par thèmes ou par notions (Bau et al., 2017).

Pour résumer, la modalité Scratch dispose d'un catalogue de blocs persistant et classé par thèmes et concepts, il permet aux débutants de découvrir des concepts et des fonctionnalités ou de se souvenir de ceux qu'ils connaissent déjà. La modalité Python met à disposition des programmeurs la documentation technique complète du langage ainsi qu'une liste contextuelle d'instructions rangées alphabétiquement. Cela nécessite un apprentissage supplémentaire de la part des débutants et n'est pas à même de soutenir la découverte du langage. Ainsi, lors de la transition d'une modalité à l'autre, les apprenants devront mémoriser la syntaxe d'implémentation des différents concepts et fonctionnalités de Python (Weintrop, 2019). Cette mémorisation représente un défi cognitif supplémentaire qui s'ajoute au fait de se rappeler de l'existence des commandes elles-mêmes (Kölling et al., 2015).

Ayant étudié la présence de catalogues de commande dans les environnements d'édition des langages Scratch et Python, intéressons-nous maintenant aux méthodes de composition des programmes.

3.4.2 Composition des programmes

Avant d'exécuter un programme, il est nécessaire de le composer en associant des commandes. Voyons comment cette composition est réalisée dans les environnements d'édition de Scratch et de Python.

Dans la modalité Scratch, il s'agit d'assembler des blocs en les choisissant dans un catalogue puis en les glissant dans la zone d'édition. Ajoutons que les différents connecteurs des blocs contraignent la manière de les assembler. Citons quelques exemples, les structures de contrôle sont en forme de C pour suggérer que des blocs doivent être placés à l'intérieur. Les blocs qui renvoient des informations sont formés en fonction des types de valeurs qu'ils retournent : ovale pour les nombres et les chaînes de caractères, et hexagone pour les booléens. Les blocs conditionnels ont des espaces vides en forme d'hexagones indiquant qu'une expression booléenne est attendue. D'autres fonctionnalités, visibles dans la Figure 3-18, permettent de guider ces associations.

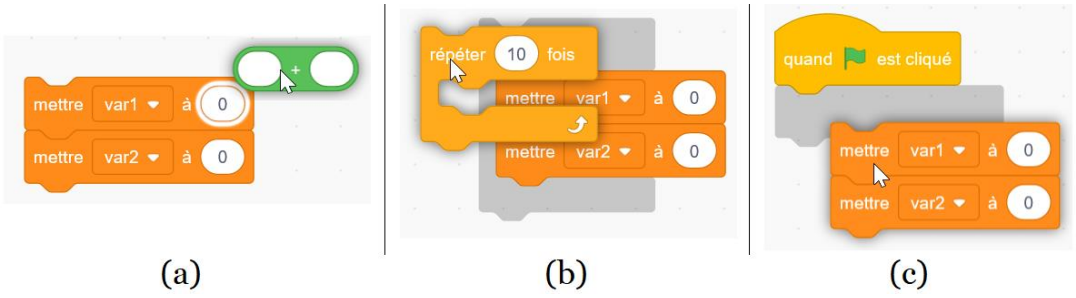


Figure 3-18 – Détails de la composition des scripts scratchs par glisser-déposer

Ainsi, lorsqu'un bloc est approché d'un espace vide de type compatible, ce réceptacle se voit entouré d'une bordure blanche (voir Figure 3-18-a). De plus, en approchant deux blocs qui peuvent s'associer, ils sont comme aimantés. Cela prend la forme d'une

ombre qui matérialise une possibilité d'articulation entre ces deux éléments (voir Figure 3-18-b et Figure 3-18-c).

Dans la modalité Python, les programmes doivent être rédigés au clavier. Deux éléments d'aide peuvent néanmoins soutenir cette rédaction. D'abord, comme nous venons de l'évoquer, la complétion automatique peut être employée afin de faciliter la saisie des mots-clés du langage. Ensuite, l'éditeur IDLE propose un service d'indentation automatique qui place le curseur au bon niveau à chaque retour à la ligne.

Comparons maintenant ces deux modalités dans le contexte de passage de l'une à l'autre. La manipulation directe et contrainte des blocs en Scratch permet d'établir et de maintenir facilement la structure globale des programmes. Cette tâche doit être effectuée manuellement en Python (Bau et al., 2017). Cette structuration manuelle du code, même soutenue par l'indentation automatique, est à l'origine de nombreuses erreurs syntaxiques et sémantiques de la part des débutants (Kölling et al., 2015). C'est d'autant plus le cas lors de l'utilisation d'un langage comme Python pour lequel l'indentation a une signification sémantique. En outre, l'acte purement mécanique de taper le texte d'un programme Python constitue un obstacle cognitif et moteur pour les jeunes apprenants (Kölling et al., 2015). En effet, ils ne possèdent souvent pas les compétences de frappe suffisantes pour être capables de produire des textes dans un temps raisonnable. De surcroit, même pour les apprenants plus âgés ayant une meilleure maitrise de la saisie au clavier, la correction des inévitables erreurs de frappe engendre des distractions cognitives (Kölling et al., 2015).

En faisant le bilan, Scratch permet de composer des programmes par glisser-déposer depuis un catalogue en assemblant les différentes pièces de manière guidée et contrainte. Python nécessite de rédiger des programmes au clavier soutenus par la complétion et l'indentation automatique. Lors de la transition d'une modalité à l'autre, la création et le maintien manuel de la structure globale des programmes est source de nombreuses erreurs en Python, le processus mécanique de saisie du texte au clavier peut représenter un défi cognitif et moteur pour les jeunes apprenants. De plus, la nécessité de corriger les inéluctables erreurs de frappe augmente les distractions cognitives.

Après avoir analysé les aspects relatifs à la composition des programmes, nous abordons dans la sous-section qui suit le sujet de l'exécution de ces programmes et plus particulièrement sa visibilité et son contrôle.

3.4.3 Visibilité et contrôle de l'exécution

Une fois les programmes composés, il s'agit de les exécuter afin qu'ils puissent agir. Étudions les différentes manières de suivre et de contrôler cette exécution dans les modalités Scratch et Python.

Pour commencer, l'environnement de programmation Scratch favorise la visibilité de l'exécution des programmes. En effet, comme nous pouvons l'observer sur la Figure 3-19, un script en cours d'exécution est mis en valeur par une bordure jaune. Cette fonctionnalité trouve tout son intérêt lorsque plusieurs scripts sont attachés au même lutin.

Figure 3-19 – Détails de l'exécution d'un programme dans l'environnement d'édition de Scratch

Selon les concepteurs du logiciel, cette mise en évidence doit permettre aux utilisateurs de repérer les scripts déclenchés et d'évaluer leur temps d'exécution : « Scratch provides visual feedback to show script execution. When a script is running, it is surrounded by a glowing white border. This feedback helps the user understand when scripts are triggered and how long they run » (Maloney et al., 2010, p. 5). D'autre part, comme dans l'exemple de la Figure 3-19, le contenu des différentes variables d'un script est visible à tout moment au niveau de la scène. L'intention des concepteurs était ici de rendre les variables plus concrètes :

> In most text-based programming languages, variables are invisible, abstract, and difficult to understand. Like earlier systems such as Boxer, Scratch turns variables into concrete objects that the user can see and manipulate, making them easier to understand through tinkering and observation. (Maloney et al., 2010, p. 6)

Concernant le contrôle de l'exécution dans la modalité Scratch, comme nous l'avons déjà dit, les scripts sont traditionnellement déclenchés en réaction à des évènements (drapeau cliqué, touche pressée, message reçu, etc.). En complément, Scratch permet l'exécution directe des blocs à tout moment. Ainsi, il suffit de cliquer sur un bloc ou un ensemble de blocs pour qu'ils agissent sur le lutin ciblé. Les créateurs du langage affirment que cela encourage l'engagement des utilisateurs et favorise un mode de programmation expérimental, ascendant et incrémental. L'influence de Papert et du constructionniste transparait clairement ici :

> A key feature of Scratch is that it is always *live*. There is no compilation step or edit/run mode distinction. Users can click on a command or program fragment at any time to see what it does. In fact, they can even change parameters or add blocks to a script while

> it is running. [..] Scratch helps users stay engaged in testing, debugging, and improving their projects. We say that Scratch is *tinkerable* because it lets users experiment with commands and code snippets the way one might tinker with mechanical or electronic components. Tinkerability encourages hands-on learning and supports a bottom-up approach to writing scripts where small chunks of code are assembled and tested, then combined into larger units [mis en valeur par les autres]. (Maloney et al., 2010, p. 4)

Ajoutons que Scratch dispose d'un « mode turbo » activable depuis le menu « modifier ». Ce mode permet d'améliorer le contrôle de l'exécution en laissant la possibilité de l'accélérer pour des projets nécessitant beaucoup de calculs mathématiques ou de nombreuses animations.

Dans la modalité Python, l'environnement IDLE dispose d'un débogueur. Cette fonctionnalité, présentée dans la Figure 3-20, permet d'avoir à la fois une excellente visibilité de l'exécution grâce à la mise en valeur de la ligne en cours d'interprétation et également d'accéder aux contenus des variables (en bas à droite). Le débogueur permet aussi le contrôle précis de l'exécution au moyen d'un mode pas à pas (boutons « Go », « Step », « Over », « Out », etc.). Notons cependant que l'activation et l'utilisation de ce débogueur sont assez complexes (nombreuses fonctionnalités, vocabulaire technique en anglais), et qu'il n'est pas forcément adapté à des programmeurs débutants mais plutôt destiné à des experts.

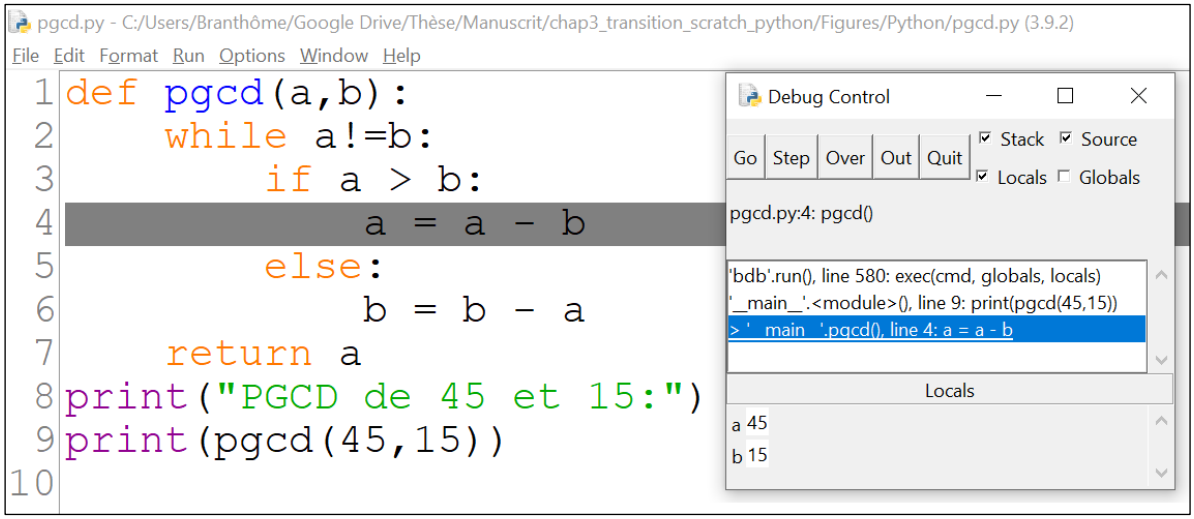


Figure 3-20 – Système de débogueur de l'environnement de programmation IDLE

De ce fait, les novices risquent de n'avoir accès qu'à une exécution opaque et peu contrôlable, consistant à lancer leur programme depuis le menu « Run » et à en observer les sorties dans la console Python.

Lors du passage de Scratch à Python, si l'apprenant parvient à utiliser le débogueur, il retrouvera dans la modalité Python les facilités de visibilité et de contrôle qui lui permettaient d'établir des liens entre les scripts Scratch et leurs actions. S'il n'a accès qu'aux sorties des programmes sans contrôle de l'exécution, il rencontrera des difficultés pour comprendre ce qu'il se joue réellement lors des exécutions. Bau et ses collègues (2017) ajoutent que la fonctionnalité d'exécution directe des blocs Scratch aide à comprendre les actions passées ou futures et permet de les rendre concrètes en les appliquant immédiatement à l'état actuel.

Pour résumer, la modalité Scratch facilite le contrôle et le suivi de l'exécution de ses scripts en indiquant le code exécuté, en affichant l'état des variables, en rendant possible l'exécution directe des blocs à tout moment, et en permettant de contrôler la vitesse d'exécution. La modalité Python facilite également la compréhension de l'exécution à travers son débogueur qui signale la ligne en cours d'exécution, affiche les contenus des variables, et propose une exécution pas à pas. Ces fonctionnalités permettent aux novices de mieux comprendre comment les programmes sont exécutés et ce qu'ils font. De plus, l'exécution directe des blocs permise par Scratch rend les exécutions plus tangibles et aide au repérage temporel.

3.4.4 Synthèse de la section

Q3.2 : Quelles sont les différences entre les environnements d'édition des langages de programmation Scratch et Python ? Quels obstacles peuvent rencontrer les élèves lors du passage de l'un à l'autre ?

En nous appuyant sur des exemples d'utilisation et sur la littérature scientifique, nous avons analysé les différences entre les environnements d'édition Scratch et Python, nos résultats sont les suivants :

- au sujet des **catalogues de commandes**, Scratch dispose d'un menu persistant listant tous les blocs classés par thèmes et concepts, il permet aux débutants de découvrir ou de se souvenir des concepts et des fonctionnalités du langage, l'environnement de Python met à disposition des programmeurs la documentation technique complète du langage ainsi qu'une complétion automatique des mots-clés, cela ne permet pas de soutenir la découverte du langage et oblige les apprenants à mémoriser les fonctionnalité du langage et leur syntaxe ;
- au sujet de la **composition des programmes**, en Scratch, les scripts sont constitués en saisissant des blocs depuis un catalogue puis en les assemblant de manière guidée et contrainte, en Python, les programmes sont rédigés au clavier assisté par la complétion et de l'indentation automatique. La création et le maintien de la structure globale des programmes Python est source d'erreurs, et le processus mécanique de saisie au clavier peut représenter un défi cognitif et moteur pour les jeunes apprenants ;
- au sujet de la **visibilité** et du **contrôle de l'exécution**, la modalité Scratch met en valeur le code exécuté et affiche l'état des variables, il rend possible l'exécution directe des blocs à tout moment et permet de contrôler la vitesse d'exécution. La modalité Python facilite le suivi de l'exécution à travers son débogueur qui signale la ligne en cours d'exécution, affiche le contenu des variables, et propose une exécution pas à pas. Ces fonctionnalités permettent aux novices de mieux comprendre comment les programmes sont exécutés, de plus, l'exécution directe des blocs Scratch rend les exécutions plus concrètes et aide à se positionner dans le temps.

Dans ce chapitre, nous avons étudié les divergences entre les modalités de programmation Scratch et Python, à la fois au niveau des langages eux-mêmes mais aussi dans les environnements d'édition des programmes. Nous avons également évoqué les conséquences de ces écarts pour les apprenants dans un contexte de transition d'une modalité à l'autre. Dans le chapitre suivant, nous présentons les résultats d'une enquête en ligne réalisée auprès d'enseignants de mathématiques, technologie et SNT, portant sur leurs pratiques d'enseignement de la programmation informatique.

Chapitre 4

Enquête sur les pratiques déclarées des enseignants

Ce quatrième chapitre clôture la première phase des analyses préalables de la méthodologie d'ingénierie didactique présentée en introduction. Il s'agit ici d'étudier les pratiques déclarées des enseignants concernant l'enseignement de la programmation informatique à la transition collège-lycée. En nous appuyant sur les analyses curriculaires réalisées dans le Chapitre 2, et en structurant nos questions selon les quatre indicateurs dégagés (contenus, activités, dispositifs et artefacts), nous menons une enquête en ligne auprès d'enseignants de mathématiques et de technologie en troisième, et les enseignants de mathématiques et de SNT en seconde. Nous analysons ensuite les 480 réponses récoltées et mesurons les écarts des pratiques déclarées avec les indications officielles. Enfin, nous en tirons les conséquences à même de nourrir la deuxième phase de la thèse : la conception d'une ressource d'enseignement permettant d'accompagner la transition collège-lycée.

La section 4.1 présente d'abord une revue de littérature internationale en lien avec les pratiques d'enseignement de la programmation dans le secondaire, suivie par la section 4.2 qui détaille les questions de recherche et la méthodologie d'enquête en ligne adoptée dans ce chapitre. Ensuite, la section 4.3 décrit l'élaboration et l'administration du questionnaire, puis la section 4.4 expose les résultats de l'enquête et les analyses qui en découlent. Enfin, la section 4.5 tire des enseignements de ces analyses pour la conception d'une ressource d'enseignement.

4.1 Revue de littérature

Notre revue de littérature porte sur des travaux centrés sur l'analyse des pratiques d'enseignement de la programmation informatique dans le secondaire (grade 6 à 12). Nous exposons d'abord les contributions portant sur les curriculums internationaux avant de présenter la production en lien avec le curriculum français qui nous intéresse dans ce chapitre.

4.1.1 Curriculums internationaux

L'étude des pratiques enseignantes fait l'objet de nombreuses publications à l'international. C'est en particulier vrai pour l'enseignement de la programmation informatique qui a fait progressivement son apparition dans l'enseignement secondaire au niveau planétaire (Storte et al., 2019). Nous présentons ici les résultats issus de trois publications représentatives qui sont des méta-revues (Garneli et al., 2015; Saeli et al., 2011) ou des analyses d'entretiens avec des enseignants (Nijenhuis-Voogt et al., 2021).

Nous retrouvons dans ces travaux une multitude de pratiques d'enseignement que nous pouvons catégoriser à l'aide de trois de nos quatre indicateurs pédagogico-

didactiques : activité, dispositifs et artefacts. Concernant les activités, plusieurs démarches sont évoquées, comme la réalisation de projets ou la résolution de problèmes (Garneli et al., 2015). Les types d'activités de programmation sont assez variés et graduels : compréhension d'un programme, modification puis création de programmes (Saeli et al., 2011). De nombreux dispositifs sont décrits en lien avec l'organisation de l'espace et les rapports entre individus : l'enseignant donne des indications ou discute des solutions devant le groupe classe, les élèves coopèrent dans des travaux de groupe, l'enseignant encadre et aide individuellement les élèves (Nijenhuis-Voogt et al., 2021). Au sujet des artefacts utilisés, les artefacts-supports permettant l'expression des algorithmes sont nombreux : langage naturel, pseudo-code, organigramme, langage de programmation graphique et textuel (Nijenhuis-Voogt et al., 2021). La programmation par blocs (Logo, Scratch et Alice) est présentée comme favorisant la motivation des élèves et facilitant la rédaction des programmes alors que la programmation textuelle est décrite comme difficile mais permettant de donner le sentiment aux élèves d'utiliser un outil professionnel au service de projets authentiques (Garneli et al., 2015). Enfin, un artefact-cible est cité, celui d'objet physique tangible. Nous retenons que les pratiques évoquées dans ces articles sont très variées et recouvrent pour partie les indications présentes dans les textes officiels français (voir section 2.8.5).

Nous présentons désormais un ensemble de travaux ayant pour objet les pratiques d'enseignement de la programmation informatique au collège et au lycée dans l'environnement scolaire français.

4.1.2 Curriculum français

Dans le contexte français, nous trouvons des travaux autour de l'enseignement de la programmation dans le secondaire (Delmas-Rigoutsos, 2020; Drot-Delange & Tort, 2022b; Journault et al., 2020; Libert & Vanhoof, 2020; Meyer & Modeste, 2022). Cependant à notre connaissance, peu d'entre eux portent spécifiquement sur les pratiques des enseignants.

Néanmoins, Drot-Delange et Tort (2018) ont étudié l'adoption par des enseignants du concours d'informatique Castor[21] comme ressource pédagogique pour la spécialité Informatique et Sciences du Numérique[22] (ISN). Les résultats montrent que les seuls énoncés des défis ne permettent pas toujours aux enseignants de cerner quelles sont les notions informatiques mobilisées ou potentiellement mobilisables dans les défis. Drot-Delange (2019) a également analysé l'organisation documentaire des enseignants de la spécialité ISN en fonction de leurs connaissances professionnelles. Elle a établi que les différentes opérations de classification visent un besoin de réassurance des connaissances lors de la préparation des cours ou de l'activité en classe.

Citons également le projet de recherche Formation et Identité professionnelle Des Enseignants d'informatique au Lycée[23] (FIDEL). Lancé en 2021, ce projet a pour

[21] Le concours CASTOR est organisé par l'association France IOI et couvre les divers aspects de l'informatique. Il se déroule sur le temps scolaire du CM1 à la terminale à l'initiative des enseignants.
[22] Cette spécialité était proposée aux élèves de terminale scientifique de 2012 à 2019 à raison de deux heures par semaine avant d'être remplacée par la spécialité NSI à la rentrée 2019.
[23] https://msh-ange-guepin.univ-nantes.fr/la-recherche/fidel et https://fidel.hypotheses.org/

objectif d'étudier l'émergence des nouveaux enseignements informatiques dans les lycées français (SNT et NSI). La problématique du projet est triple. Elle consiste à étudier les transformations curriculaires consécutives à la réforme des lycées de septembre 2019, à analyser les dispositifs de formation spécifiquement mis en œuvre (DIU et CAPES), et à interroger la reconfiguration des groupes professionnels et la construction d'une identité professionnelle spécifique des enseignants d'informatique en France. Au moment où nous écrivons ces lignes, les publications scientifiques issues de ce projet ne sont pas encore disponibles.

Au regard de ces publications, nous pouvons identifier un manque de recherches ciblant les pratiques déclarées des enseignants dans le cadre du curriculum français comme nous le retrouvons dans les travaux internationaux. L'analyse des données issues du questionnaire que nous avons administré devrait être en mesure d'éclairer ces pratiques d'enseignement.

Suite à cette revue de littérature, la section suivante présente les questions de recherche et la méthodologie utilisée.

4.2 Questions de recherche et méthodologie

Nous détaillons dans cette section les différents questionnements qui dirigent les recherches présentées dans ce chapitre ainsi que la méthodologie mise en œuvre.

4.2.1 Questions de recherche

Dans ce quatrième chapitre, nous étudions la question de recherche Q4 : Quelles sont les pratiques déclarées des enseignants concernant l'enseignement de la programmation informatique en troisième et en seconde ? Quelles répercussions pour la création d'une ressource d'enseignement ?

Cette question se subdivise en plusieurs sous-questions :

- Q4.1 : Quelles sont les pratiques déclarées des enseignants concernant les contenus d'enseignement de la programmation informatique ?
- Q4.2 : Quelles sont les pratiques déclarées des enseignants concernant les activités d'enseignement de la programmation informatique ?
- Q4.3 : Quelles sont les pratiques déclarées des enseignants concernant les dispositifs d'enseignement de la programmation informatique ?
- Q4.4 : Quelles sont les pratiques déclarées des enseignants concernant les artefacts utilisés pour l'enseignement de la programmation informatique ?
- Q4.5 : Quels enseignements peut-on tirer des réponses aux questions précédentes concernant la création d'une ressource d'apprentissage ?

4.2.2 La méthodologie de l'enquête

La méthodologie employée dans ce chapitre est basée sur une enquête par questionnaire auprès des enseignants. Afin de guider l'élaboration et l'administration du questionnaire, puis l'analyse des réponses, nous suivons les dix étapes proposées par Vilatte (2007) :

- (E1) définir l'objet de l'enquête ;

- (E2) définir les hypothèses de l'enquête ;
- (E3) déterminer la population de l'enquête ;
- (E4) déterminer l'échantillon de l'enquête ;
- (E5) établir le projet de questionnaire ;
- (E6) réaliser un prétest du questionnaire ;
- (E7) rédiger définitivement le questionnaire ;
- (E8) administrer le questionnaire ;
- (E9) dépouiller et coder les données ;
- (E10) analyser les résultats.

Nous précisons ci-dessous les détails méthodologiques concernant certaines étapes de cette méthodologie.

Échantillonnage de la population (E4)

Afin de déterminer l'échantillon des enquêtés, il s'agit d'abord de choisir une méthode de sélection des individus de la population, puis de décider de la taille de l'échantillon.

D'après Berndt (2020), il existe plusieurs méthodes d'échantillonnage classées dans deux catégories principales. L'*échantillonnage probabiliste*, qui garantit que chaque individu de la population a une probabilité égale d'être sélectionné, et l'*échantillonnage non probabiliste*, pour lequel l'échantillon est sélectionné sur la base du jugement subjectif du chercheur. L'auteur ajoute que les questions de recherche quantitatives se prêtent davantage aux méthodes d'échantillonnage probabilistes car elles permettent une meilleure généralisation des résultats ainsi qu'un contrôle de la marge d'erreur. En revanche, les questions de recherche qualitatives, exploratoires et descriptives correspondent mieux aux méthodes d'échantillonnage non probabilistes. L'objet de notre recherche dans ce chapitre étant plutôt descriptif, nous envisageons ce deuxième type de méthodes, d'autant plus que nous ne disposons pas des moyens permettant de construire un échantillon probabiliste. Parmi les méthodes non probabilistes décrites par Berndt (2020), nous retenons l'*échantillonnage par auto-sélection* pour des raisons de coût et de simplicité de mise en œuvre. Dans cette méthode, le chercheur spécifie les critères d'inclusion et d'exclusion de l'échantillon, et les individus de la population choisissent de participer en fonction de leur libre arbitre. Toujours selon Berndt (2020), les avantages de l'auto-sélection sont qu'elle permet de gagner du temps et que les personnes qui choisissent de participer sont susceptibles d'être plus engagées dans la recherche et de fournir des réponses plus sincères. Néanmoins, cette méthode favorise le biais de sélection qui peut conduire à un échantillon moins représentatif ou à des résultats exagérés ou trompeurs.

Concernant la taille de l'échantillon, étant donné que nous ne mettons pas en œuvre de méthodes probabilistes, les techniques de détermination basées sur l'intervalle de confiance souhaité et la taille de la population (Lakens, 2022) ne sont pas pertinentes. Nous nous fixons simplement l'objectif d'atteindre le maximum de répondants.

Projet de questionnaire (E5)

Afin d'établir notre projet de questionnaire, nous choisissons de nous appuyer sur les quatre indicateurs du fonctionnement pédagogico-didactique des disciplines afin de structurer les groupes de questions : contenus, activités, dispositifs et artefacts (voir section 2.2.2). Les questions découlent ensuite directement des sous-indicateurs (ou catégories) que nous avons établis dans la section 2.2.2 et de nos résultats concernant l'habitat des concepts (section 2.5.5) et les modalités d'enseignement (section 2.8.5).

Au sujet de la forme des questions, nous privilégions les *questions fermées*, « celles où les personnes interrogées doivent choisir entre des réponses déjà formulées à l'avance » (de Singly, 2020, p. 66), pour des raisons d'économie. En effet, ces questions sont à la fois plus rapides à traiter pour les enquêtés et plus rapides à dépouiller pour le chercheur car elles ne nécessitent pas ou peu de codage. Ainsi, nous avons limité le nombre de *questions ouvertes*, « celles où, au contraire, les personnes interrogées sont libres de répondre comme elles le veulent » (de Singly, 2020, p. 66), Comme l'indique de Singly, ce type de question est à utiliser lorsque « ce sont les catégories (les mots) des personnes interrogées qui intéressent le chercheur plus que les informations proprement dites » (2020, p. 68), ce qui n'est pas notre cas ici. De plus, nous formulons les questions du questionnaire comme des *questions factuelles*, c'est-à-dire qu'elles « tente[nt] de cerner une dimension de la pratique » (de Singly, 2020, p. 61) à la différences des *questions d'opinion* « qui demande[nt] un jugement sur cette pratique » (de Singly, 2020, p. 61).

Concernant les réponses aux questions, pour les questions fermées, nous choisissons de proposer de nombreuses *modalités*, ce qui a l'avantage de permettre d'obtenir des réponses plus personnelles et moins conformistes :

> Proposer des réponses multiples augmente les chances d'obtenir des réponses plus personnelles. Avec cette ouverture, les personnes sentent moins la pression, imaginaire, de chercher la bonne solution : dès qu'une question a plusieurs « solutions », elle ressemble moins à un problème scolaire. Cette possibilité accroît la probabilité de voir apparaître des réponses moins conformistes. (de Singly, 2020, p. 73)

Les différentes modalités que nous soumettons aux répondants sont issues des résultats de nos analyses du Chapitre 2 portant sur les textes officiels. Nous complétons ces modalités par des recherches sur Internet et dans des catalogues commerciaux de matériel pédagogique à destination des enseignants (nous donnons plus de détails à ce sujet dans la suite). Concernant la mise en œuvre des questions fermées, nous utilisons des boutons radio, des listes déroulantes ou des curseurs à placer entre deux extrémités pour les *questions à choix unique* (QCU), et des listes de cases à cocher pour les *questions à choix multiples* (QCM). Comme le conseille de Singly, nous ajoutons également pour la majorité des questions fermées une réponse « Autre » qui déclenche l'apparition d'un champ texte permettant aux enquêtés de compléter leur réponse si celles proposées ne sont pas suffisantes : « Prévoir, dans la liste des réponses, une catégorie "autres" avec de la place pour ces réponses libres "imprévues" » (2020, p. 69). Pour les questions ouvertes, nous utilisons des champs textes ou numériques libres.

Pré-test du questionnaire (E6)

Après avoir établi le projet de questionnaire, Vilatte (2007) suggère de le tester auprès d'un petit groupe d'individus. Ce pré-test a pour but d'évaluer la formulation, l'ordre et la disposition des questions, mais aussi d'identifier des manques et de vérifier les contraintes temporelles :

> Il s'agit d'évaluer la clarté et la précision des termes utilisés et des questions posées, la forme des questions, l'ordre des questions, l'efficacité de la mise en page, éliminer toutes les questions ambiguës ou refusées, repérer les omissions, voir si le questionnaire est jugé trop long, ennuyeux, indiscret, etc. (2007, p. 9)

Rédaction définitive (E7)

Pour l'implémentation définitive du questionnaire, nous choisissons la modalité de l'enquête en ligne. En effet, selon Gingras et Belleau (2015), cela présente de nombreux avantages opérationnels tels que : une collecte rapide des données, une réduction des erreurs de saisie, une souplesse géographique et temporelle :

> Les sondages en ligne présentent de nombreux avantages [...]. Leurs coûts d'utilisation sont faibles, et ils permettent de collecter rapidement les données. L'envoi de rappel est facilité par l'utilisation du courriel, et le mode Web élimine le risque d'erreur lors de la saisie des données [...]. Contrairement aux sondages qui nécessitent la présence physique des répondants, les sondages en ligne ont aussi l'avantage de permettre aux sondés d'amorcer les questionnaires au moment et dans le lieu de leur choix, et de le compléter dans un délai qui leur convient. [...] Grâce aux envois par courriel, ce type de méthode de collecte de données permet aussi de couvrir un bassin géographique de répondants plus large. (2015, p. 6)

Cependant, toujours selon ces auteurs, cette modalité connait également des limites liées au contrôle de l'échantillonnage (biais de sélection), à la confidentialité et à l'anonymat, au taux de réponse plus faible et au désengagement en cours d'enquête. En ce qui concerne les biais d'échantillonnage dus à la méthode d'auto-sélection, nous avons déjà expliqué qu'au regard de l'objectif descriptif de notre enquête il ne s'agissait pas ici d'un écueil majeur. Ensuite, afin de parer aux inquiétudes liées à la confidentialité des données : « quelques études soutiennent que le format Web des sondages engendrerait parfois une plus grande méfiance des individus par rapport à la confidentialité des données et aux nouvelles technologies en général » (Gingras & Belleau, 2015, p. 11), nous nous attachons à rendre le questionnaire anonyme et à le mettre en conformité avec les règlements en vigueur en terme de protection des données (nous reviendrons en détails sur ces aspects dans la suite).

Ensuite, la recherche montre également que la situation d'enquête peut être perçue par les répondants comme une tentative de contrôle par l'institution jugée intrusive :

> La situation d'enquête est appréhendée comme une situation officielle de perquisition. Les enquêtes ont comme présupposés non seulement que toute personne a une opinion (effet d'imposition de la problématique), mais aussi qu'elle a envie de l'énoncer. L'enquête par questionnaire (comparable, à ce niveau, au sondage d'opinion) oublie le fait que les individus ont des réserves d'information, « des faits qui concernent l'individu et dont il entend contrôler l'accès lorsqu'il se trouve en présence d'autrui », selon les termes d'Erving Goffman. (de Singly, 2020, p. 76)

De Singly propose plusieurs subterfuges afin de limiter cette perception de perquisition officielle. Il préconise notamment de rappeler le caractère général du traitement des informations récoltées :

> Plusieurs moyens peuvent donner à l'individu l'impression d'être protégé [...] Tout d'abord, en ne négligeant pas d'expliquer au début de l'enquête que les informations obtenues ne sont jamais traitées à un niveau individuel, qu'elles sont exploitées de manière à respecter l'anonymat des personnes. (de Singly, 2020, p. 76)

Nous communiquons donc sur l'aspect informatif de l'enquête, au service de la recherche scientifique.

Enfin, Gingras et Belleau (2015) situent la durée idéale d'un questionnaire en ligne entre 15 et 20 minutes, c'est l'objectif que nous nous donnons ici.

Administration du questionnaire (E8)

Le questionnaire doit être diffusé auprès de la population cible afin d'être administré. Gingras et Belleau considèrent que la réussite de cette diffusion est liée en particulier au canal utilisé ainsi qu'à la fenêtre temporelle choisie : « Le succès des invitations au sondage dépend de trois facteurs : le lieu où il est diffusé (blogues, sites Internet, etc.), sa visibilité et le moment où il est activé. » (2015, p. 9).

Pour ce qui est de la diffusion, ayant opté pour un échantillonnage par auto-sélection, notre but est de toucher le maximum d'individus qui répondront de façon volontaire au questionnaire. Pour ce faire, nous choisissons de procéder par envoi de courriels d'invitation sur des listes de diffusion professionnelles.

Concernant les aspects temporels, cette enquête étant réalisée au cours d'une période touchée par la crise sanitaire liée à la COVID19, nous visons une fenêtre temporelle favorable. Cette fenêtre de réponse doit idéalement se situer en dehors de tout confinement et dans une période durant laquelle l'activité scolaire est le moins perturbée possible.

Enfin, de Singly recommande de conclure l'enquête par des remerciements et la proposition de mise à disposition des résultats : « À la fin du questionnaire, la personne qui le passe doit remercier l'enquêté d'avoir donné de son temps. Elle peut, dans certaines enquêtes, lui proposer de lui envoyer un petit compte rendu. » (2020, p. 72).

Dépouillement et codage des données (E9)

Pour le dépouillement des données, nous procédons à l'exportation des réponses depuis la plateforme de questionnaire en ligne vers un fichier tableur au format xlsx.

Nous nous attachons ensuite au codage de ces réponses. Pour les questions fermées, qui sont composées de plusieurs modalités, nous :

- agrégeons les modalités ayant une faible proportion à la modalité « Autre » ;
- créons de nouvelles modalités issues des réponses données dans les champs libres « Autres/préciser » si elles représentent une proportion significative .

Les seuils d'agrégation et de création ne sont pas fixes mais dépendent du nombre de modalités déjà prises en compte.

Pour les questions ouvertes, nous codons les réponses récupérées dans les champs textes en créant des modalités qui les résument en quelques mots avant de procéder à une éventuelle phase d'agrégation.

Analyse des résultats (E10)

Afin d'analyser les données précédemment codées, nous utilisons les statistiques descriptives en effectuant quelques traitements calculatoires simples avant de les représenter graphiquement.

Ces traitements et représentations dépendent du type de données dont nous disposons. Puisque nous basculons dans le domaine des statistiques, nous considérons ici les réponses aux différentes questions comme des variables. Selon Larini et Barthes (2018), il est possible de distinguer trois types de variables. D'abord, les variables *qualitatives*, « qui expriment un état. Elles s'expriment à l'aide de modalités » (2018, p. 33). Ces variables qualitatives peuvent être *nominales* (NO), c'est-à-dire qu'elles « se contentent de donner un état » (2018, p. 33), ou *ordinales* (OR), « dans lequel il est possible de définir un ordre, une hiérarchie » (2018, p. 33). Ensuite, les variables *numériques* (NU), « qui exprime[nt] une mesure, elle[s] se traduit[ssent] par un nombre entier ou un nombre réel » (2018, p. 33). Une seconde distinction concerne le nombre de valeurs que peut prendre une variable : elle peut être *univaluée* (U) pour les QCU ou *multivaluée* (M) pour les QCM (Robertson, 1991, p. 61). Nous donnons quelques exemples afin d'illustrer ces catégories. Une variable contenant l'âge d'une personne en années est numérique et univaluée (NU_U), une variable portant sur les langues qu'elle parle (FR, EN, ES, DE, IT, etc.) est nominale et multivaluée (NO_M), et une variable représentant sa taille vestimentaire (S,M,L,XL, etc.) est ordinale et univaluée (OR_U).

Précisons maintenant les traitements calculatoires que nous effectuons. Pour les variables nominales (NO) ou ordinales (OR), nous calculons les proportions de chaque modalité en pourcentage pour chacune des quatre disciplines de la population de répondants. Pour les variables numériques (NU), nous calculons la moyenne des valeurs ainsi que leur répartition en utilisant la méthode Kernel Density Estimation (KDE) (Parzen, 1962; Węglarczyk, 2018) qui permet d'estimer la fonction de répartition sous-jacente d'un ensemble de données discrètes.

Ensuite, pour les présentations graphiques des données, nous procédons différemment en fonction du type de variables. Pour les variables qualitatives univaluées (OR_U et NO_U) nous utilisons des *diagrammes en barres empilées* (Albarello et al., 2010) qui permettent de comparer les répartitions entre les disciplines. Nous en donnons un exemple dans la Figure 4-1-a qui pourrait représenter la répartition des tailles vestimentaires des enseignants par discipline. Pour les variables qualitatives multivaluées (OR_M ou NO_M), les *diagrammes en barres juxtaposées* ou en « tuyaux d'orgue » (Albarello et al., 2010, p. 13) permettent à fois de représenter les multiples modalités et de comparer leurs distributions selon les disciplines (voir par exemple la Figure 4-1-b qui pourrait représenter la répartition des langues parlées par les enseignants par discipline). Enfin, pour les variables numériques (NU_U), il serait possible de visualiser la distribution des valeurs à l'aide d'histogrammes ou de polygones de fréquence, mais cela nécessiterait la constitution de classes (Albarello et al., 2010, p. 27). Nous optons plutôt pour une estimation de la

distribution (KDE) par une *courbe de répartition* continue d'aire sous la courbe d'une unité. Cette courbe présente l'avantage de pouvoir faire figurer plusieurs distributions simultanément sur un même graphique. Le Figure 4-1-c en donne un exemple, elle pourrait représenter la répartition de l'âge des enseignants par discipline.

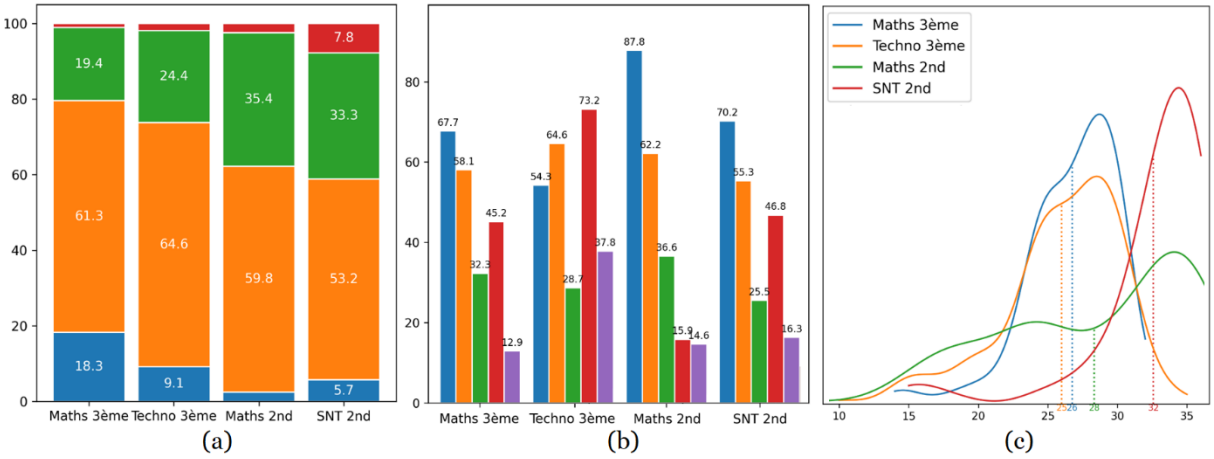


Figure 4-1 – Exemples de représentations graphiques, barres empilées (a), barres juxtaposées (b) et courbes de répartitions (c)

Les traitements calculatoires et la constitution des représentations graphiques sont effectués à l'aide de scripts Python. La bibliothèque Pandas (McKinney, 2011) permet d'importer les données depuis le fichier tableur et d'effectuer les calculs de proportion par discipline, les bibliothèques Matplotlib (Hunter, 2007) et Seaborn (Waskom, 2021) servent à la construction des représentations graphiques.

Dans la section suivante nous exposons les résultats de la première partie de la méthodologie, c'est-à-dire la constitution et l'administration du questionnaire.

4.3 Élaboration et administration du questionnaire

Nous détaillons dans cette section la mise en œuvre des étapes E1 à E8 de notre méthodologie concernant l'élaboration et l'administration du questionnaire de l'enquête.

4.3.1 Objet de l'enquête (E1)

Pour commencer, présentons l'objet de l'enquête. Il s'agit d'une enquête *descriptive* dont la « principale fonction est de décrire une situation, de répondre à un besoin d'information » (Vilatte, 2007, p. 6). Son objet est de connaître les pratiques d'enseignement de la programmation informatique dans l'enseignement secondaire et en particulier à la transition collège-lycée.

4.3.2 Hypothèses de l'enquête (E2)

Notre hypothèse est que les enseignants suivent les prescriptions et les recommandations édictées par le ministère de l'Éducation nationale dans les programmes d'enseignement et les ressources d'accompagnement. Pour rappel, nous avons regroupé ces indications officielles selon quatre indicateurs (voir section 2.8) : contenus, activités, dispositifs et artefacts.

4.3.3 Population de l'enquête (E3)

La population des enquêtés est l'ensemble des enseignants impliqués dans l'enseignement de la programmation informatique à la transition collège-lycée. Nous avons précédemment identifié (voir la section 2.4.1) que cet enseignement était délivré en mathématiques au cycle 4 et en seconde, en technologie au cycle 4, ainsi qu'en SNT en seconde. Nous avons choisi de restreindre l'étude du cycle 4 au seul niveau troisième dans l'objectif de se focaliser sur la transition collège-lycée. La population concernée par notre enquête est donc l'ensemble des professeurs qui enseignent : les mathématiques ou la technologie au niveau troisième, les mathématiques ou les SNT au niveau seconde.

Il est possible d'estimer les effectifs de chacune de ces sous-populations. En effet, en 2021, la France comptait 24 073 enseignants de mathématiques et 9 440 enseignants de technologie au collège ainsi que 12 125 enseignants de mathématiques au lycée (DEPP, 2021b, p. 35). En supposant que la moitié d'entre eux interviennent en troisième et en seconde, nous pouvons évaluer notre population à approximativement : 12 000 enseignants de mathématiques et 5 000 enseignants de technologie en troisième, et 6 000 enseignants de mathématiques en seconde. Nous ne disposons pas des statistiques précises au sujet des enseignants de SNT. Il est cependant possible de d'estimer leur nombre. Ainsi, en 2021, la France disposait de 17 917 sections de seconde (DEPP, 2021b, p. 291), et donc d'autant de groupes de SNT. En considérant que chaque enseignant a en moyenne la charge de deux groupes, il semble raisonnable d'évaluer leur effectif à environ 9 000.

4.3.4 Échantillon de l'enquête (E4)

Nous présentons ensuite notre échantillon. Afin de mettre en œuvre la méthode non probabiliste d'auto-sélection présentée dans la section 4.2, nous avons choisi d'inviter les enseignants à participer volontairement à l'enquête via une diffusion par courriel sur des listes professionnelles. Les critères d'inclusion sont explicités dans le courriel d'invitation (nous détaillons ces aspects dans la suite). Concernant la taille de l'échantillon, nous nous sommes fixé l'objectif d'atteindre le maximum de répondants. Nous avons affiné cet objectif en espérant toucher au moins 1% de la population. Cela correspond à 120 enseignants de mathématiques et 50 enseignants de technologie en troisième, et à 60 enseignants de mathématiques et 90 enseignants de SNT en seconde. Cela représente un total de 320 individus.

4.3.5 Projet de questionnaire (E5)

L'étape suivante consiste à élaborer un projet de questionnaire. Le questionnaire soumis aux enquêtés est reproduit en intégralité en Annexe B. Le Tableau 4-1 en propose un résumé par groupe de questions et par discipline. Nous le commentons à la suite.

Disciplines / **Groupes**	**Maths 3ème**	**Techno 3ème**	**Maths 2nd**	**SNT 2nd**
A : Informations générales	QA1* – Ancienneté (6 classes/QCU) QA2* – Discipline principale (15 modalités + a/QCU) QA3*– Niveaux d'enseignement principaux (7 modalités + a/QCM)			

<table>
<tr><td></td><td colspan="4">QA4* – Discipline secondaire ? (2 modalités/QCU)
QA5 – Discipline secondaire (15 modalités + a/QCU)
QA6 – Niveaux d'enseignement secondaires (7 modalités + a/QCM)
QA7* – Formation initiale programmation informatique ? (2 modalités/QCU)
QA8* – Formation continue progr. informatique (7 modalités + a/QCM)
QA9* – Rapport à l'informatique : usage, intérêt, connaissances (3 curseurs de 0 à 100/QCU)
QA10 – Commentaires ou remarques (champ texte)</td></tr>
<tr><td>B : Discipline et niveau</td><td colspan="4">QB1* – Discipline et niveau d'enseignement réponse enquête (4 mod./QCM)</td></tr>
<tr><td rowspan="3">C : Contenus et connaissances</td><td colspan="4">QC1* – Différence algorithme/programme (4 modalités/QCU)</td></tr>
<tr><td>QC2* – Concert. (4 mod./QCU)</td><td>QC3* – Concert. (4 mod./QCU)</td><td>QC4* – Concert. (4 mod./QCU)</td><td>QC5* – Concert. (4 mod./QCU)</td></tr>
<tr><td colspan="4">QC6 – Commentaires ou remarques (champ texte)</td></tr>
<tr><td rowspan="5">D : Activité d'enseignement</td><td colspan="4">QD1* – Nombre séances programmation informatique par an (champ num.)
QD2* – Ressources utilisées dans la préparation (10 modalités + a/QCM)</td></tr>
<tr><td>QD3* – Liens avec programme (5 mod./QCM)</td><td>QD4* – Liens avec programme (4 mod./QCM)</td><td>QD5* – Liens avec programme (5 mod./QCM)</td><td>QD6* – Liens avec programme (8 mod./QCM)</td></tr>
<tr><td colspan="4">QD7* – Type de démarche (6 modalités + a/QCM)
QD8* – Type d'activité (7 modalités + a/QCM)
QD9* – Type d'activité collaboratives (6 modalités + a/QCM)</td></tr>
<tr><td colspan="2">QD10* – Mode programmation (2 modalités + a/QCM)</td><td colspan="2">QD11* – Mode programmation (3 modalités + a/QCM)</td></tr>
<tr><td colspan="4">QD12 – Activité programmation débranchée (champ texte)
QD13 – Commentaires ou remarques (champ texte)</td></tr>
<tr><td>E : Dispositif d'enseignement</td><td colspan="4">QE1* – Lieu d'enseignement (4 modalités + a/QCM)
QE2* – Nombres d'élèves par classe (champ numérique)
QE3* – Heure dédoublées en demi-groupe ? (2 modalités/QCU)
QE4* – Organisation de l'espace et interactions (7 modalités + a/QCM)
QE5 – Commentaires ou remarques (champ texte)</td></tr>
<tr><td rowspan="5">F : Aspects matériels et logiciels</td><td colspan="2">∅</td><td colspan="2">QF1* – Environnement de dev. (18 modalités + a/QCM)
QF2* – Bibliothèques logicielles (13 modalités + a/QCM)</td></tr>
<tr><td colspan="4">QF3* : Machine de développement (5 modalités + a/QCM)</td></tr>
<tr><td>∅</td><td>QF4* – Cartes prog (7+a/QCM)
QF5* – Rob prog (6+a/QCM).
QF6* – Sys dom (8+a/QCM)
QF7* – Simulat. (2 mod./QCU)</td><td>∅</td><td>QF4* – Cartes programmables (7+a/QCM)
QF5* – Robots programmables (6+a/QCM)
QF7* – Simulat. (2 mod./QCU)</td></tr>
<tr><td colspan="2">QF8* – Langage programmation (5 modalités + a/QCM)</td><td colspan="2">QF9* – Langage programmation (13 modalités + a/QCM)</td></tr>
<tr><td colspan="4">QF10* – Navigateur Internet (6 modalités + a/QCM)
QF11 – Commentaires ou remarques (champ texte)</td></tr>
</table>

G : Situation sanitaire	QG1 – Impact de la situation sanitaire sur l'enseignement (champ texte)			
Total questions	33	37	35	38

Tableau 4-1 – Résumé du questionnaire par groupe de questions, l'astérisque désigne une question obligatoire, QCU désigne une question à choix unique et QCM une question à choix multiples, la modalité « a » correspond à « Autre »

Nous avons fait le choix de structurer les questions selon sept groupes. Un premier groupe concerne les informations générales (groupe A), c'est-à-dire les éléments biographiques et personnels des répondants (ancienneté, disciplines et niveaux d'enseignement, formation initiale et continue, rapport à l'informatique). Ces informations doivent nous permettre de connaitre la constitution de notre échantillon et, de façon secondaire, d'évaluer sa représentativité en comparaison avec des statistiques nationales publiées par la Direction de l'Évaluation, de la Prospective et de la Performance[24] (DEPP).

Comme nous venons de le voir, le questionnaire est destiné à quatre sous-populations selon la discipline et le niveau d'enseignement : maths 3ème, techno 3ème, maths 2nd et SNT 2nd. Le Tableau 4-1 montre que le questionnaire comporte entre 33 et 38 questions en fonction de la sous-population d'appartenance du répondant. Afin de permettre l'adaptabilité du questionnaire via des affichages conditionnels, nous avons créé un groupe particulier (groupe B) contenant une unique question discriminante. Cette question permet au répondant de se positionner dans une sous-population et de déterminer la suite du questionnaire : « QB1 : Dans quelle discipline et à quel niveau enseignez-vous la programmation informatique cette année ? Choisir une possibilité parmi les quatre proposées ci-dessous. Vous répondrez ensuite aux questions qui vont suivre spécifiquement pour cette discipline et ce niveau » (Annexe B.3). Si le répondant appartient à plusieurs sous-populations, il est invité à faire le choix le plus pertinent en lien avec la programmation informatique.

Ensuite, nous avons repris les indicateurs du fonctionnement pédagogico-didactique des disciplines issus des travaux de Reuters (voir section 2.2.2) afin de structurer le cœur du questionnaire. Nous retrouvons donc quatre groupes (groupe C à F) dont les questions découlent directement des sous-indicateurs (ou catégories) que nous avons établis dans la section 2.2.2 et de nos résultats concernant l'habitat des concepts (section 2.5.5) et les modalités d'enseignement (section 2.8.5) :

- contenus et connaissances (différence algorithme-programme, concertation avec les autres disciplines) ;
- activités d'enseignement (nombre de séances impliquant la programmation, ressources utilisées pour préparer les cours, liens avec les autres parties du programme, démarche, type d'activité, mode de programmation, programmation débranchée) ;

[24] La DEPP est un service au sein du ministère de l'Éduction nationale qui exerce ses compétences d'évaluation et de mesure de la performance dans les domaines de l'éducation et de la formation en contribuant à l'évaluation des politiques conduites.

- dispositifs d'enseignement (lieux, nombre d'élèves par classe, organisation de l'espace, interactions) ;
- aspects matériels et logiciels (artefacts-supports : environnement de développement, langage de programmation, bibliothèques utilisées, machine de développement, artefacts-cibles : cartes, robots, objets domotiques, simulations programmables).

Ces questions constituent le noyau du questionnaire, elles doivent permettre de répondre à nos questions de recherche.

Enfin, un dernier groupe de question (groupe G) porte sur l'impact de la situation sanitaire sur les pratiques d'enseignement de la programmation informatique. Rappelons que ce questionnaire a été diffusé au début du mois de mai 2021 (nous y reviendrons plus loin), c'est-à-dire après une année marquée par trois confinements et des mesures sanitaires perturbant fortement le fonctionnement scolaire[25]. Dans le préambule du questionnaire, nous avons fait le choix de demander aux enquêtés de répondre en occultant les adaptations liées à cette situation sanitaire exceptionnelle :

> Cette enquête porte sur vos pratiques d'enseignement habituelles, c'est à dire hors situation sanitaire exceptionnelle liée à la COVID19. Il vous est demandé d'essayer, dans la mesure du possible, de faire abstraction des adaptations rendues nécessaires par cette situation particulière et que vous espérez abandonner dès que possible. En revanche, si vous pensez conserver certaines de ces pratiques lorsque la situation reviendra à la normale, vous pouvez les évoquer. (Annexe B.1)

En contrepartie, nous avons proposé, dans ce dernier groupe, une question facultative portant spécifiquement sur ce thème qui doit permettre aux répondants de faire part, s'ils le souhaitent, des conséquences de la COVID19 sur leurs pratiques d'enseignement.

Concernant la forme des questions, comme indiqué dans la section 4.2, nous avons privilégié les questions fermées et factuelles. Par conséquent, une large majorité de nos questions sont fermées (indication QCM et QCU dans le Tableau 4-1). Néanmoins, nous avons rédigé quelques questions ouvertes. Ainsi, les questions facultatives QA10, QC6, QD13, QE5 et QF11 permettent aux répondants de faire part de commentaires ou remarques pour chaque groupe de questions. Les questions QD12 et QG1, portant sur les activités débranchées et les impacts de la COVID19, pour lesquelles nous n'avions pas ou peu de modalités de réponse à proposer à priori sont également ouvertes. Enfin, les questions QD1 et DE2 qui interrogent des aspects quantitatifs proposent de répondre dans des champs numériques ouverts.

Au sujet des réponses aux questions fermées, les différentes modalités que avons soumises aux répondants sont issues, pour les groupes C, D, et E, des résultats de nos analyses des textes officiels. Concernant le groupe F, en lien avec les artefacts, nous avons mené en complément des recherches sur Internet concernant les environnements de développement Python existants (QF1) et les bibliothèques utilisées (QF2), et dans des catalogues commerciaux de matériel pédagogique à

[25] La France a connu trois confinements successifs avec fermeture ou restriction d'accès aux établissements scolaires en 2020 et 2021 : du 17 mars au 11 mai 2020, puis du 30 octobre au 15 décembre 2020, et enfin du 3 avril au 3 mai 2021. En dehors de ces périodes de confinement, l'accueil scolaire fut également fortement perturbé par des mesures limitant les jauges et les activités autorisées.

destination des enseignants de technologie et SNT pour les questions en rapport avec les artefacts-cibles (QF4 à QF7). Comme nous l'avons expliqué dans la section 4.2, nous ajoutons une modalité « Autre » (« a » dans le Tableau 4-1) pour les questions dont les modalités proposées ne sont pas exhaustives.

4.3.6 Prétest du questionnaire (E6)

Une fois le projet de questionnaire établi, la méthodologie recommande d'effectuer un pré-test. Nous avons donc soumis notre projet de questionnaire à trois enseignants expérimentés et ayant de bonnes connaissances dans le domaine de la programmation informatique. Nous avons également essayé de diversifier notre échantillon afin qu'il contienne des représentants des quatre sous-populations ciblées par l'enquête. Ce groupe de testeurs est donc composé de :

- un enseignant de mathématiques en troisième suivant par ailleurs un master recherche en didactique ;
- un enseignant de mathématiques en seconde engagé dans une formation continue universitaire (DIU) afin d'enseigner la NSI ;
- un enseignant de SNT en seconde également formateur NSI à l'INSPE.

Nous ne sommes malheureusement pas parvenu à intégrer dans ce pré-test un enseignant de technologie intervenant au niveau troisième.

Ces testeurs avaient la consigne d'évaluer la formulation des questions, les modalités de choix proposées pour les réponses, de repérer les éventuels manques, et de mesurer leur durée de passation. Nous avons fourni à chacun d'entre eux un tableau leur permettant de consigner leurs remarques pour chaque question.

Leurs retours nous ont permis de valider globalement notre questionnaire tout en lui apportant quelques modifications : correction de quelques erreurs typographiques, repérage de problèmes techniques, précisions, modifications ou reformulation de certaines questions. Le temps de passation a également été évalué à la hausse, passant de 10 minutes à 15-20 minutes.

4.3.7 Rédaction du questionnaire (E7)

Le questionnaire ayant été amendé et validé par un groupe de pré-testeurs, l'étape suivante consiste à le formaliser. Comme précisé dans la section 4.2.1, nous avons décidé de le mettre en œuvre via une plateforme de questionnaire en ligne. Nous avons pour cela utilisé l'application LimeSurvey (Klieve et al., 2010) disponible dans notre université. Ajoutons que les données récoltés sont hébergées sur les serveurs de l'INSPE de Bretagne.

Comme indiqué précédemment, cette modalité d'administration implique plusieurs inconvénients liés à la confidentialité des données et à la perception de « perquisition officielle ». Afin de palier ces désagréments, nous avons anonymisé les questions (pas de collecte de données personnelles permettant d'identifier directement les enseignants) et vérifié la conformité du questionnaire avec les règlements en vigueur en termes de protection des données. De plus, nous avons inclus dans le préambule du questionnaire le passage suivant :

> Ce questionnaire est strictement anonyme, il ne recueille donc aucune donnée personnelle (nom, prénom, date de naissance, établissement scolaire, adresse IP, etc.). Par conséquent, il est conforme aux textes encadrant l'utilisation des données personnelles en France : la loi n° 2018-493 du 20 juin 2018, qui modifie la loi Informatique et Libertés du 6 janvier 1978 ; le Règlement Général sur la Protection des Données (RGPD), voté par le Parlement Européen et le Conseil Européen en 2016, et entré en application le 25 mai 2018. (Annexe B.1)

Afin de contrecarrer le sentiment de contrôle institutionnel, qui peut être d'autant plus marqué ici que nous avons diffusé l'enquête en utilisant des canaux officiels (voir plus loin), nous avons choisi d'insister dans le préambule sur l'aspect informatif et scientifique de l'enquête : « Il s'agit ici d'une démarche purement informative au service de la recherche et soumise à aucun jugement ni à aucune évaluation. » (Annexe B.1).

Enfin, concernant la durée de l'enquête, notre questionnaire en ligne comporte entre 33 et 38 questions très majoritairement fermées, sa durée de passation a été estimée entre 15 et 20 minutes par le groupe des pré-testeurs, ce qui est conforme aux recommandations méthodologiques (voir section 4.2.2). Nous avons par conséquent communiqué sur une durée de 15 minutes dans le préambule du questionnaire (voir Annexe B.1) et le courriel d'invitation (voir Annexe J.1).

4.3.8 Administration du questionnaire

Le questionnaire finalisé doit ensuite être diffusé auprès de la population cible afin d'être administré. Nous avons vu dans la section 4.2.2 que les canaux et la fenêtre temporelle de diffusion étaient importants.

Comme précisé précédemment, nous avons choisi d'effectuer cette diffusion par envoi de courriels sur des listes de diffusion professionnelles. Le courrier électronique d'invitation est reproduit en Annexe J.1, il reprend de façon synthétique : l'objectif du sondage, les critères d'inclusion dans l'enquête, des éléments pratiques (temps de passation, anonymat), et le lien vers l'enquête en ligne. Pour la diffusion, nous avons eu accès aux listes académiques des enseignants de l'académie de Rennes par l'intermédiaire d'un inspecteur académique de mathématiques, et à des listes professionnelles hébergées sur la plateforme RENATER[26] ou l'association PAGESTEC[27] auxquelles nous nous sommes inscrit. Au final, la propagation de l'enquête a été effectuée sur les listes de diffusion suivantes :

- listes académiques de mathématiques (2 595 inscrits), de technologie (416 inscrits) et de SNT (189 inscrits) de l'académie de Rennes ;
- listes nationales RENATER de mathématiques (208 inscrits), de technologie (99 inscrits), de SNT (1 502 inscrits) et de NSI (1 245 inscrits) ;
- liste nationale PAGESTEC des enseignants de technologie au collège (770 inscrits).

[26] Le Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche (RENATER) est un groupement d'intérêt public (GIP) qui assure la maîtrise d'ouvrage du réseau national de communications électroniques pour la technologie, l'enseignement et la recherche.
[27] L'association PAGESTEC est une association professionnelle qui œuvre pour le développement, la promotion et la défense de la discipline Technologie au collège.

Concernant les aspects temporels, comme indiqué au-dessus, cette enquête a été réalisée pendant une période perturbée par la crise sanitaire liée à la COVID19. Nous avons donc attendu la sortie du troisième confinement (fermeture des collèges et lycée du 03 avril au 03 mai) afin que les enseignants puissent répondre dans les meilleures conditions. Cette enquête a donc été soumise aux enseignants du 07 mai 2021 au 15 juin 2021.

Enfin, en suivant la méthodologie détaillée dans la section 4.2.2, nous avons ajouté, suite aux remerciements d'usage, une invitation à recevoir les conclusions de l'enquête ainsi qu'à être informé de la création de futures ressources numériques : « Si vous souhaitez connaître les résultats de ce questionnaire, ou être informé de la conception de ressources numériques conçues pour l'enseignement de la programmation dans le cadre de ma recherche, vous pouvez m'adresser un mail à l'adresse suivante: matthieu.branthome@univ-brest.fr » (Annexe B.9)

Les deux dernières étapes de l'enquête (E9 et E10) consistent à traiter et à analyser les réponses des enquêtés. La section qui suit expose ces résultats.

4.4 Résultats de l'enquête

Nous présentons dans cette section l'analyse des réponses collectées suite à la passation du questionnaire. Nous commençons par exposer quelques statistiques générales concernant l'enquête avant de détailler les analyses pour les cinq groupes de questions.

4.4.1 Statistiques générales

L'enquête a reçu 684 réponses dont 204 partielles et 480 complètes. Comme nous pouvons le voir dans la Figure 4-2-a, le taux de réponses partielles est de 29,8%. Cela correspond au taux d'abandon moyen de 30% constaté dans la littérature pour les enquêtes en ligne sur invitation non nominatives (Galesic, 2006, p. 313).

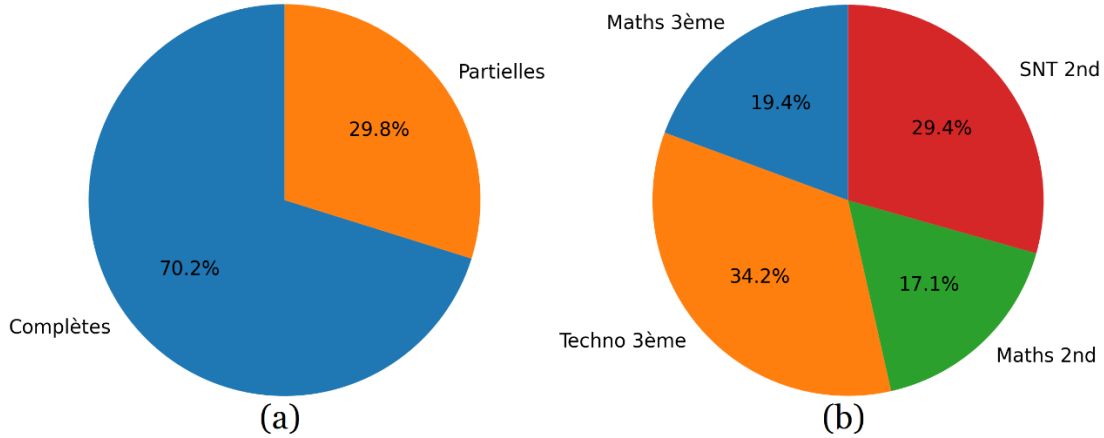


Figure 4-2 – Type de réponses de l'enquêtes, (a) répartition des réponses par complétude, (b) répartition des réponses complètes par discipline

Parmi les 480 réponses complètes, la répartition par discipline est la suivante :

- mathématiques en troisième : 93 réponses (0,8 % de la population) ;
- technologie en troisième : 164 réponses (3,3 % de la population) ;

– mathématiques en seconde : 82 réponses (1,4% de la population) ;

– SNT en seconde : 141 réponses (1,6% de la population).

Comme le montre la Figure 4-2-b, cette répartition n'est pas uniforme. La sur-représentation des enseignants de technologie et de SNT (respectivement 34,2% et 29,4% des répondants) est sans doute liée à l'utilisation des listes de diffusion PAGESTEC et RENATER SNT et NSI qui comprennent toutes les trois beaucoup d'inscrits. Notons par ailleurs que nous atteignons approximativement notre objectif d'échantillonnage de 1% de la population.

La fenêtre temporelle de réponse était étalées sur 40 jours du 07 mai 2021 au 15 juin 2021. Cependant, comme l'indique la Figure 4-3, les réponses se sont concentrées sur les dix premiers jours de l'enquête. Nous aurions ainsi pu clôturer l'enquête dès le 18 mai 2021.

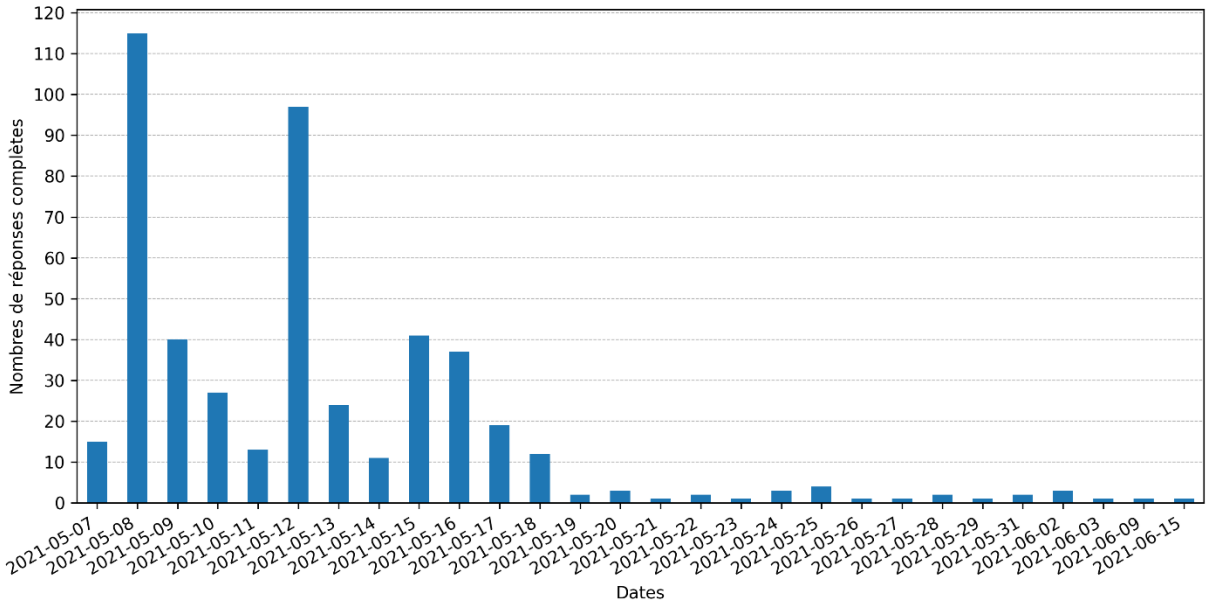


Figure 4-3 – Nombres de réponses complètes au questionnaire par date

La durée moyenne de passation du questionnaire est de 16 minutes et 18 secondes, ce qui respecte globalement l'estimation de 15 minutes communiquée aux répondants.

Nous ne disposons pas de l'origine des répondants selon les canaux de diffusion employés. Nous ne sommes par conséquent pas en mesure de comparer le rendement de ces listes quant à leur taux de réponse. Nous supposons néanmoins une surreprésentation des enseignants de l'académie de Rennes au regard des listes académiques utilisées.

Enfin, nous avons proposé en fin de questionnaire une prise de contact par courriel afin de faire parvenir à ceux qui le souhaitaient les résultats de l'enquête et des informations concernant la production de ressources numériques. Suite à cette proposition, 31 personnes nous ont contacté. Parmi elles, 11 personnes souhaitaient obtenir les résultats de l'enquête et 20 personnes désiraient être informées de la création de ressources numériques. Nous verrons dans le chapitre suivant que nous avons utilisé ces contacts afin de trouver des terrains d'expérimentation pour la ressource d'enseignement que nous avons conçue.

Avant de présenter le détail des analyses par groupe de questions, nous récapitulons dans le Tableau 4-2 les traitements et représentations graphiques que nous avons effectué. En suivant la méthodologie développée dans la section 4.2.2, nous donnons, pour chaque question du questionnaire :

- le type de variables qui contient la réponse : ordinale univariée (OR_U), nominale univariée (NO_U), nominale multivariée (NO_M), ou numérique univariée (NU_U) ;
- le traitement calculatoire effectué sur les données : proportion des modalités en pourcentage par discipline, estimation de la répartition par KDE, ou moyenne des valeurs ;
- le type de représentations graphiques utilisé : barres empilées, barres juxtaposées, ou courbes de répartition.

Questions	Types	Traitements	Représentations
QA1 – Ancienneté	OR_U	% par modalité et discipline	Barres empilées
QA2 – Discipline principale	NO_U	% par modalité et discipline	Barres empilées
QA3 – Niveaux enseignement princ.	NO_M	∅	∅
QA5 – Discipline secondaire	NO_U	% par modalité et discipline	Barres empilées
QA6 – Niveaux enseignement sec.	NO_M	∅	∅
QA7 – Formation initiale	NO_U	% par modalité et discipline	Barres empilées
QA8 – Formation continue	NO_M	% par modalité et discipline	Barres juxtaposées
QA9 – Rapport à l'informatique	NU_U	KDE et moy. par discipline	Courbes répartition
QA10 – Commentaires et remarques	NO_M	∅	∅
QC1 – Diff. algorithme/programme	OR_U	% par modalité et discipline	Barres empilées
QC2-5 – Concertation enseignants	NO_U	% par modalité et discipline	Barres empilées
QC6 – Commentaires et remarques	NO_M	∅	∅
QD1 – Nombre séances par an	NU_U	KDE et moy. par discipline	Courbes répartition
QD2 – Ressources utilisées	NO_M	% par modalité et discipline	Barres juxtaposées
QD3-6 – Liens programmes	NO_M	% par modalité et discipline	Barres juxtaposées
QD7 – Type de démarche	NO_M	% par modalité et discipline	Barres juxtaposées
QD8 – Type d'activité	NO_M	% par modalité et discipline	Barres juxtaposées
QD9 – Type d'activité collab.	NO_M	% par modalité et discipline	Barres juxtaposées
QD10-11 – Mode programmation	NO_M	% par modalité et discipline	Barres juxtaposées
QD12 – Activité prog. débranchée	NO_U	% par modalité et discipline	Barres empilées
QD13 – Commentaires et remarques	NO_M	∅	∅
QE1 – Lieu d'enseignement	NO_M	% par modalité et discipline	Barres juxtaposées
QE2 – Nombres d'élèves par classe	NU_U	KDE et moy. par discipline	Courbes répartition
QE3 – Heures dédoublées en ½ gr.	NO_U	% par modalité et discipline	Barres empilées
QE4 – Dispositif d'enseignement	NO_M	% par modalité et discipline	Barres juxtaposées
QE5 – Commentaires et remarques	NO_M	∅	∅
QF1 – Environnement de dev.	NO_M	% par modalité et discipline	Barres juxtaposées

QF2 – Bibliothèques logicielles	NO_M	% par modalité et discipline	Barres juxtaposées
QF3 - Machine de développement	NO_M	% par modalité et discipline	Barres juxtaposées
QF4 – Cartes programmables	NO_M	% par modalité et discipline	Barres juxtaposées
QF5 – Robots programmables	NO_M	% par modalité et discipline	Barres juxtaposées
QF6 – Systèmes domotiques	NO_M	% par modalité et discipline	Barres juxtaposées
QF7 – Simulation numériques	NO_U	% par modalité et discipline	Barres juxtaposées
QF8-9 – Langage programmation	NO_M	% par modalité et discipline	Barres juxtaposées
QF10 – Navigateur Internet	NO_M	% par modalité et discipline	Barres juxtaposées
QF11 – Commentaires et remarques	NO_M	∅	∅
QG1 – Situation sanitaire	NO_M	∅	∅

Tableau 4-2 – Type, traitement et représentation des données par question

Notons que les questions QA3 et QA6 portant sur les niveaux d'enseignement n'ont pas été analysées dans la mesure où les réponses étaient redondantes avec la question discriminante QB1 permettant l'affichage conditionnel du questionnaire. Ensuite, nous avons parcouru les réponses aux questions ouvertes à la fin de chaque groupe (QA10, QC6, QD13, QE5 et QF11) sans qu'elles n'aient véritablement d'influence sur les résultats. Il s'agissait souvent pour les enseignants de préciser des informations synthétisées dans les modalités à choisir (par exemple concernant leur formation initiale ou les activités proposées aux élèves). Enfin, la question ouverte concernant la situation sanitaire (QG1) n'a pas été traitée faute de temps et en raison de son éloignement avec le sujet principal de cette thèse.

En ce qui concerne les autres questions, nous détaillons dans les sous-sections suivantes l'analyse des réponses selon les différents groupes.

4.4.2 Informations générales

Nous commençons par analyser les réponses au groupe de questions portant sur les éléments biographiques et personnels des répondants (Groupe A).

Ancienneté

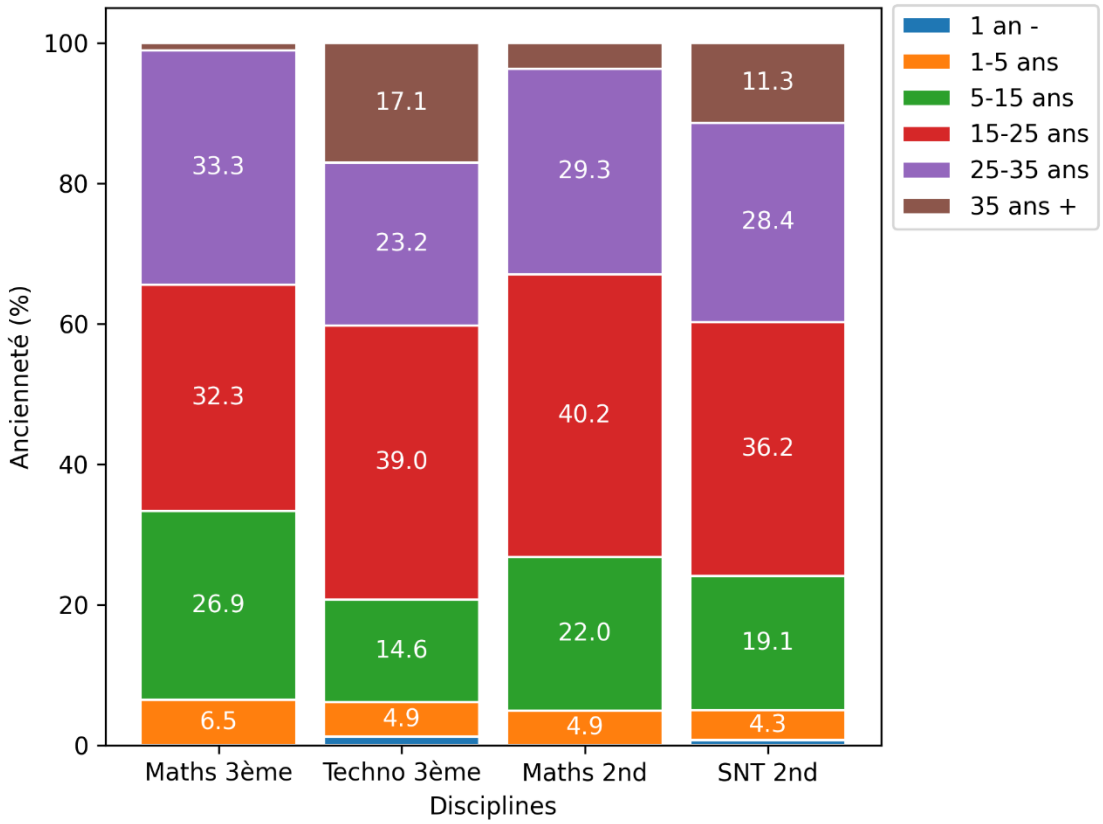


Figure 4-4 – Ancienneté des enseignants en proportion par discipline / QA1 : Quelle est votre ancienneté dans le métier d'enseignant ?

Comme le montre la Figure 4-4, la répartition des enseignants interrogés dans les différentes classes d'anciennetés est assez similaire selon les disciplines. Nous remarquons cependant que les enseignants de mathématiques sont globalement plus jeunes en comparaison de ceux de technologie et de SNT, en particulier en troisième. Notons la quasi absence d'enseignants totalement débutants.

L'ancienneté moyenne des enseignants en 2021 était de 17 ans dans le second degré (DEPP, 2022a, p. 161). Dans notre échantillon de répondants, la moyenne d'age calculée en prenant le centre des classes est de 21,4 années. Nous pouvons donc constaster que notre échantillon est plus expérimenté que la population générale étudiée.

Discipline principale

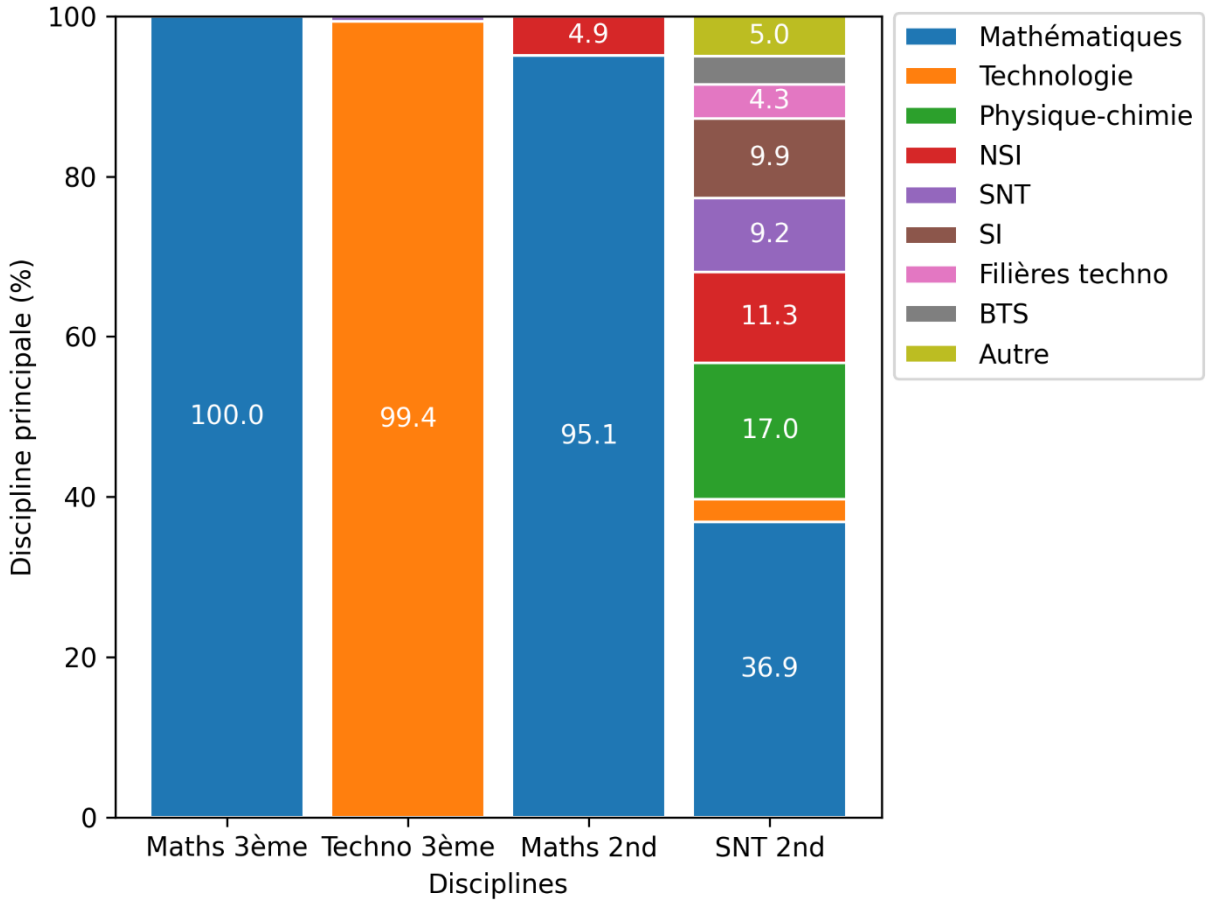


Figure 4-5 – Discipline principale des enseignants en proportion par discipline de réponse à l'enquête/ QA2 : Quelle est votre discipline d'enseignement principale en nombre d'heure ?

Ensuite, la Figure 4-5 indique que, contrairement aux enseignants de mathématiques et de technologie, les enseignants de SNT sont issus de disciplines très variées. Ce sont principalement des enseignants de mathématiques (36,9%), de physique-chimie (17%) ou de NSI (11,3%). Seuls 9% d'entre eux ont la SNT comme discipline principale. Les autres enseignants sont globalement issus de matières scientifiques (SI, filières technologiques, BTS). Nous trouvons, à la marge dans la catégorie « Autre », quelques profils plus originaux (langues, philosophie, SES).

Selon un rapport de la DEPP (2021a), à la rentrée 2020 au niveau national, les enseignants de SNT avaient les « disciplines de poste » suivantes : mathématiques (43,5%), technologie[28] (26,4%), physique-Chimie (16,7%), autres (12,6%) et NSI (0,8%). En opérant le même classement, c'est-à-dire en considérant que nos répondants en SI, filières technologiques et BTS font partie de la méta-catégorie « technologie », nous atteignons 20,6% pour cette catégorie. Nous pouvons donc constater une sous-représentation des enseignants de mathématiques (36,9% contre 43,5%) et de technologie (20,6% contre 26,4%) dans notre échantillon SNT au profit

[28] Dans ce rapport, le terme technologie ne désigne pas la discipline du collège mais les matières technologiques enseignées en lycée général ou technologique ainsi qu'en BTS : Sciences de l'Ingénieur (SI), Ingénierie et développement durable (I2D), Innovation technologique (IT), Méthodes et Techniques Informatiques (MTI), etc.

des enseignants de NSI qui sont largement sur-représentés (11,3% contre 0,8%). Cela est sous doute lié au biais de sélection provoqué par la diffusion du questionnaire sur la liste RENATER NSI.

Discipline secondaire

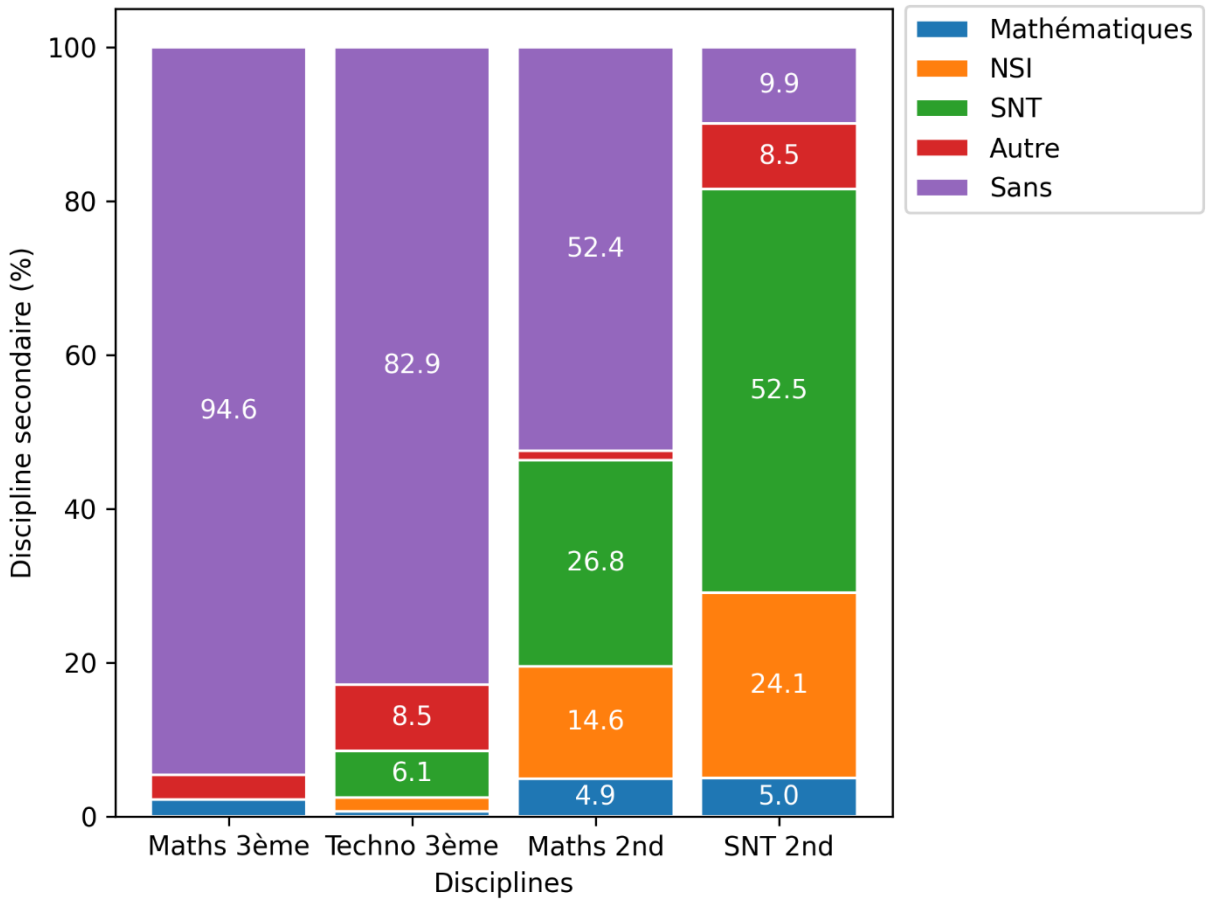


Figure 4-6 – Discipline secondaire des enseignants en proportion par discipline de réponse à l'enquête/ QA5 : Quelle est votre discipline d'enseignement secondaire ?

Comme l'indique la Figure 4-6, les enseignants de troisième de notre enquête n'ont en général pas de discipline secondaire. En revanche en seconde, et de façon très marquée en SNT, ils ont couramment la charge d'une seconde matière d'enseignement. La SNT est d'ailleurs la discipline secondaire la plus répandue parmi nos répondants.

Formation initiale

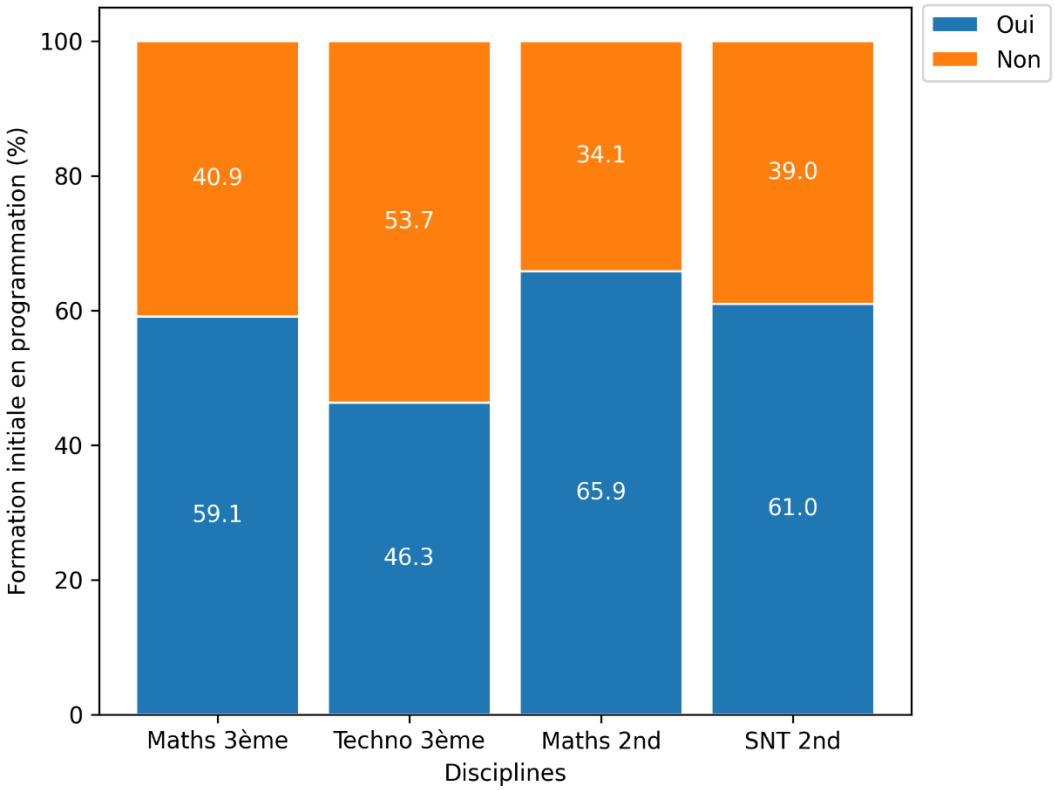


Figure 4-7 – Formation initiale en programmation des enseignants en proportion par discipline / QA7 : Concernant votre formation initiale, possédez-vous un ou plusieurs diplômes en lien avec la programmation informatique (au moins un module d'enseignement y était consacré) ?

En consultant la Figure 4-7, nous pouvons avancer de façon schématique qu'environ la moitié des enseignants interrogés ont pu bénéficier d'un module d'enseignement de programmation au cours de leur formation initiale. Cette proportion est plus importante en mathématiques (d'autant plus en seconde). Les enseignants de technologie ont proportionnellement été moins formés à la programmation lors de leurs études.

Formation continue

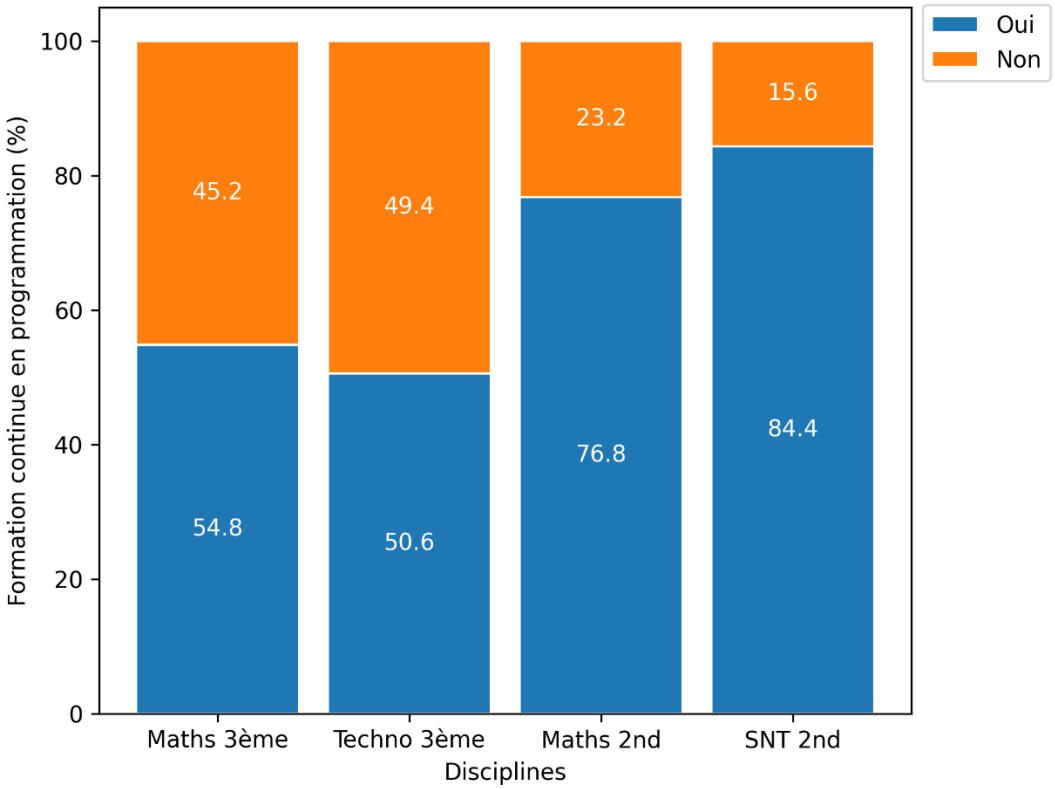


Figure 4-8 – Formation continue en programmation des enseignants en proportion par discipline / QA8 : Avez-vous suivi une ou plusieurs formations continues relatives à la programmation informatique ?

D'après la Figure 4-8, environ la moitié des enseignants de troisième ont pu suivre une formation continue en lien avec la programmation (54,8% et 50,6%). Cette proportion est plus élevée au lycée. En effet, en seconde les trois quarts des enseignants de mathématiques et environ 84% des enseignants de SNT ont eu accès à une telle formation. Comme nous venons de le voir, c'est paradoxalement les enseignants de seconde qui avaient déjà la meilleure formation initiale en programmation. Nous pouvons faire l'hypothèse que l'offre de formation continue a été plus importante suite à l'introduction des nouvelles disciplines au lycée (SNT, NSI) en septembre 2019.

Selon la DEPP (2022a, p. 232), durant l'année scolaire 2020-2021, 65% des enseignants du second degré se sont inscrits à au moins un module de formation continue (quelle que soit la thématique). Ce taux d'accès avoisinant généralement les 75% sur la période 2015-2019 hors crise sanitaire (DEPP, 2022a, p. 232). Il est difficile de comparer nos données accumulées sur la carrière et spécifiques à la programmation informatique avec ces données annuelles et générales. Ces données nationales permettent tout de même de confirmer cette forte appétence des enseignants pour les formations continues.

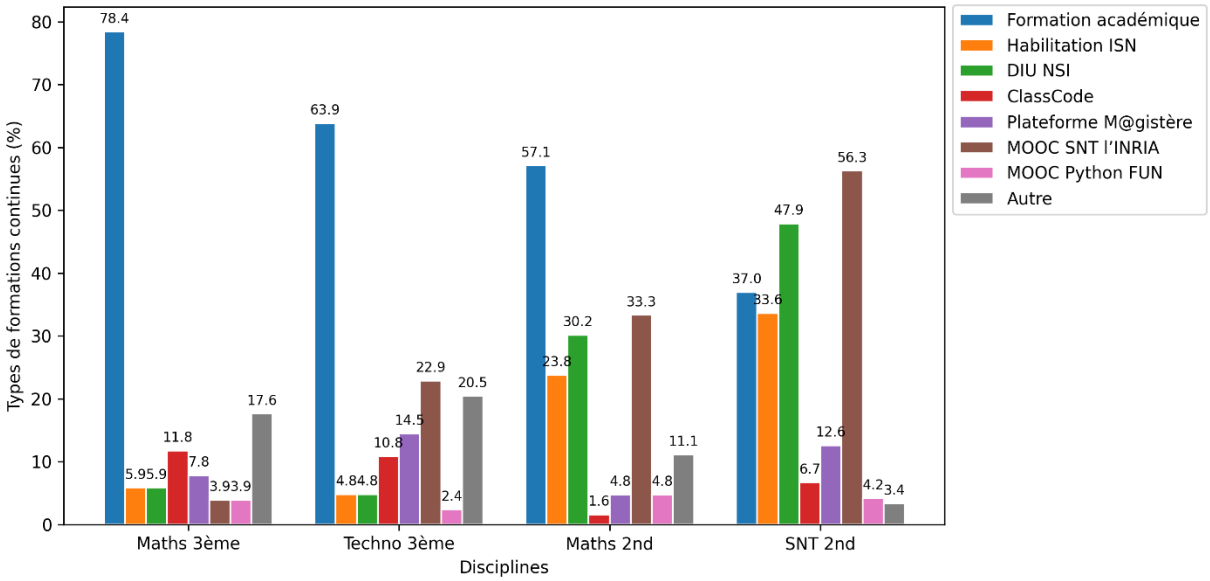


Figure 4-9 – Détails des formations continues suivies par les enseignants en proportion et par discipline / QA8 : Avez-vous suivi une ou plusieurs formations continues relatives à la programmation informatique ?

La Figure 4-9 donne le détail des formations continues suivies par les enseignants. Elle indique que les formations académiques sont les plus demandées, en particulier en troisième. On retrouve en seconde une forte présence des formations universitaires, habilitation ISN et DIU NSI, requises pour enseigner respectivement les spécialités ISN puis NSI. En effet, nous avons vu précédemment que les enseignants de mathématiques et de SNT en seconde sont souvent également des enseignants de NSI (respectivement 15% et 25%). Notons également la présence importante des formations en ligne (ClassCode, M@gistère et MOOCs) en complément. Ce phénomène semble particulièrement marqué pour la SNT où le MOOC de l'INRIA a par exemple été suivi par plus de la moitié des répondants.

À titre de comparaison, selon un rapport du Cnesco[29] (A. Paris, 2021, p. 14), en 2020, 47% des enseignants du second degré avaient eu recours pendant leur carrière à au moins un action de formation proposée par une structure hors de l'éducation nationale (université, association, mouvement pédagogique, etc.). Cette statistique explique en partie la part élevée des formations extérieures à l'Éducation nationale dans notre échantillon, en particulier en seconde.

[29] Le Centre national d'étude des systèmes scolaires (Cnesco) fait partie du Conservatoire National des Arts et Métiers (CNAM). C'est une instance indépendante chargée d'évaluer le système scolaire français en publiant des recommandations, des études et des rapports scientifiques.

Rapport à l'informatique

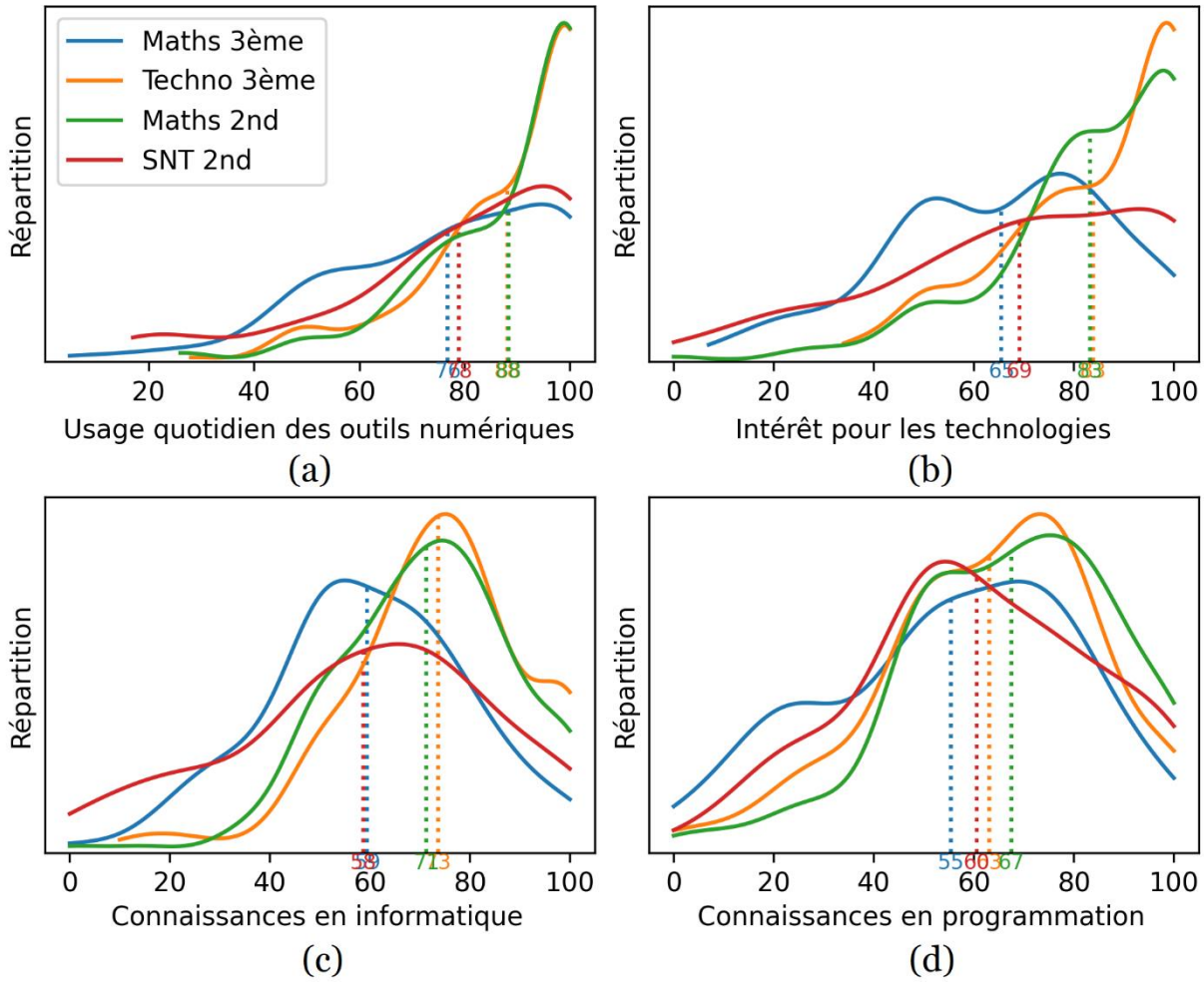


Figure 4-10 – Fonctions de répartition et moyennes des enseignants concernant leurs rapports au numérique par discipline / QA9 : Comment qualifieriez-vous les éléments suivants ? Donner pour chaque élément une estimation allant de «Faible» à «Important» en déplaçant le curseur

Lors de l'enquête, les enseignants étaient amenés à se positionner au sujet de leur rapport à l'informatique en évaluant plusieurs aspects à l'aide de curseurs allant de « Faible » à « Important ». Ces curseurs génèraient, en arrière-plan, des scores de 0 à 100. Concernant les usages quotidiens des outils numériques (Figure 4-10-a), deux groupes semblent se distinguer. Les enseignants de mathématiques en troisième et ceux de SNT déclarent un usage moindre des outils numériques (en moyenne respectivement 76 et 78) en comparaison des enseignants de technologie et de mathématiques en seconde (en moyenne 88). On retrouve cette même dichotomie à l'égard de l'intérêt pour le « numérique et les technologies en général » (Figure 4-10-b). Cette division en deux groupes est également visible à propos des connaissances en informatique (Figure 4-10-c). Nous pouvons tenter d'expliquer ces différences par le fait que les enseignants de SNT ont des profils assez hétérogènes et peuvent venir de disciplines non directement liées à l'informatique. À l'inverse, les enseignants de mathématiques en seconde enseignent également pour moitié en SNT ou en NSI. Ils ont donc à priori plus d'attaches avec l'informatique et le numérique. En ce qui concerne le dernier élément évalué, les connaissances en programmation (Figure 4-10-

d), l'ordre des disciplines est sensiblement le même, cependant la répartition est un peu différente. De manière cohérente avec les formations initiales déclarées, les enseignants de mathématiques en seconde évaluent le plus favorablement leurs connaissances (en moyenne 67). Paradoxalement, les enseignants de technologie ont les taux de formation initiale et continue en programmation les plus faibles (46,3% et 50,6%) mais évaluent plutôt favorablement leurs connaissances dans ce domaine (en moyenne 63).

Finissons par remarquer que les enseignants de SNT, discipline la plus directement en lien avec l'informatique, sont ceux qui déclarent le moins de connaissance dans ce domaine (en moyenne 58). C'est logiquement eux qui ont le plus recours aux formations continues (84,4%). Il est probable que pour cette question, le terme « informatique » soit entendu par les enseignants du SNT comme la science informatique (théorie et techniques informatiques) alors que les enseignants des autres disciplines le comprennent comme l'ensemble des outils et usages de l'informatique (voir section 1.2.3). Cela pourrait expliquer ces différences dans l'autoévaluation des connaissances.

4.4.3 Contenus et connaissances

Nous exposons dans cette sous-section les analyses des réponses aux questions du groupe C relatives aux contenus et connaissances.

Distinction algorithme-programme

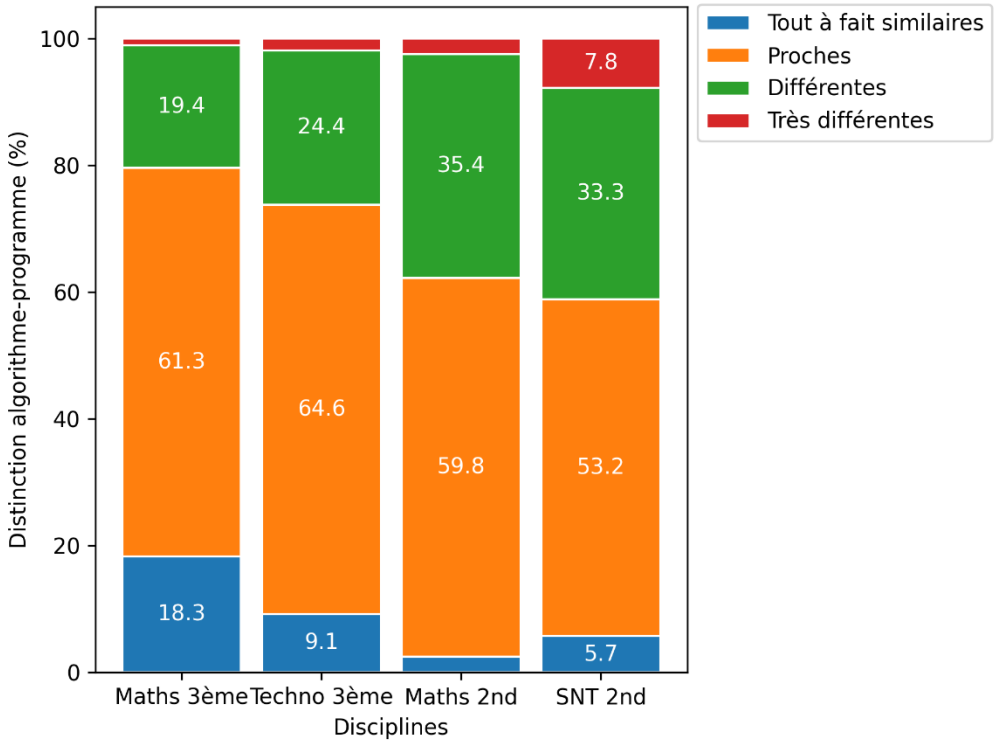


Figure 4-11 – Distinction entre algorithme et programme en proportion par discipline / QC1 : Quelle distinction faites-vous <u>avec vos élèves</u> entre la notion d'algorithme et la notion de programme ? Je présente ces notions comme étant :

Comme le montre la Figure 4-11, la distinction des notions d'algorithme et de programme parait suivre un gradient des mathématiques en troisième jusqu'à la SNT

en seconde. Cette évolution est cohérente avec les indications émanant des textes officiels (voir section 2.8.5), même si, en nous basant sur ces textes, nous pouvions nous attendre à une différenciation moins forte en mathématiques en troisième et plus forte en seconde.

Concertation entre enseignants au sujet des concepts

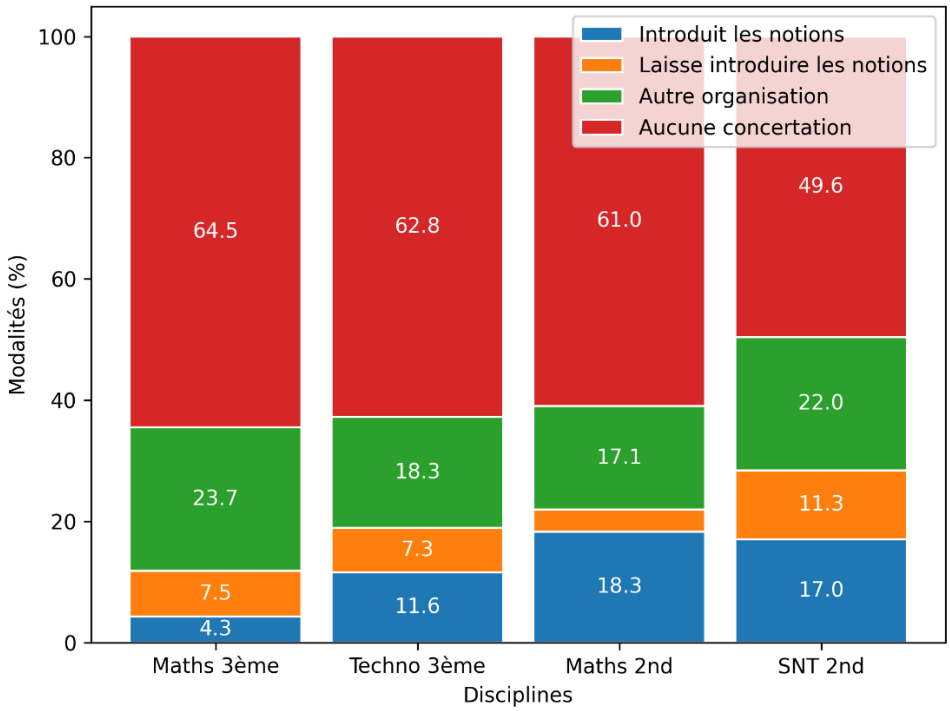


Figure 4-12 – Concertation avec des enseignants de l'autre discipline au sujet de la programmation en proportion et par discipline / QC2-QC5 : Vous concertez-vous avec vos collègues professeurs de… concernant l'introduction ou l'utilisation des notions algorithmiques ?

L'analyse des textes officiels (voir section 2.5.5) montre que les enseignants sont encouragés à enseigner la programmation « conjointement » par couple de disciplines pour un même niveau (maths-techno et maths-SNT). Ces textes désignent plus ou moins explicitement les mathématiques comme responsables de l'introduction des notions, et la technologie et les SNT comme disciplines de « réinvestissement ».

La réalité esquissée par cette enquête semble bien différente. D'abord, la Figure 4-12 indique que plus de la moitié des enseignants ne se concertent pas avec les professeurs de leur binôme disciplinaire. Ensuite, lorsqu'une coordination est mise en place, l'introduction des notions n'apparait pas comme étant particulièrement portée par les enseignants de mathématiques. Aucune organisation évidente ne sort de ces résultats, même si la concertation entre les enseignants semble aller croissante de la troisième à la seconde.

Synthèse de la sous-section

Q4.1 : Quelles sont les pratiques déclarées des enseignants concernant les contenus d'enseignement de la programmation informatique ?

L'analyse des réponses à notre enquête fait ressortir les résultats suivants concernant les contenus d'enseignement :

- la **distinction** entre les notions d'**algorithme** et de **programme** est davantage faite auprès des **élèves de seconde** que de ceux de troisième ;
- plus de **la moitié des enseignants ne se concertent pas** avec les professeurs de leur binôme disciplinaire au sujet de l'enseignement de la programmation, lorsque qu'une coordination est mise en place, les enseignants de **mathématiques n'endossent pas particulièrement la charge de l'introduction** conceptuelle.

4.4.4 Activité

Cette sous-section contient les analyses des réponses aux questions ayant trait aux activités d'enseignement de la programmation (groupe D).

Nombre de séances par an

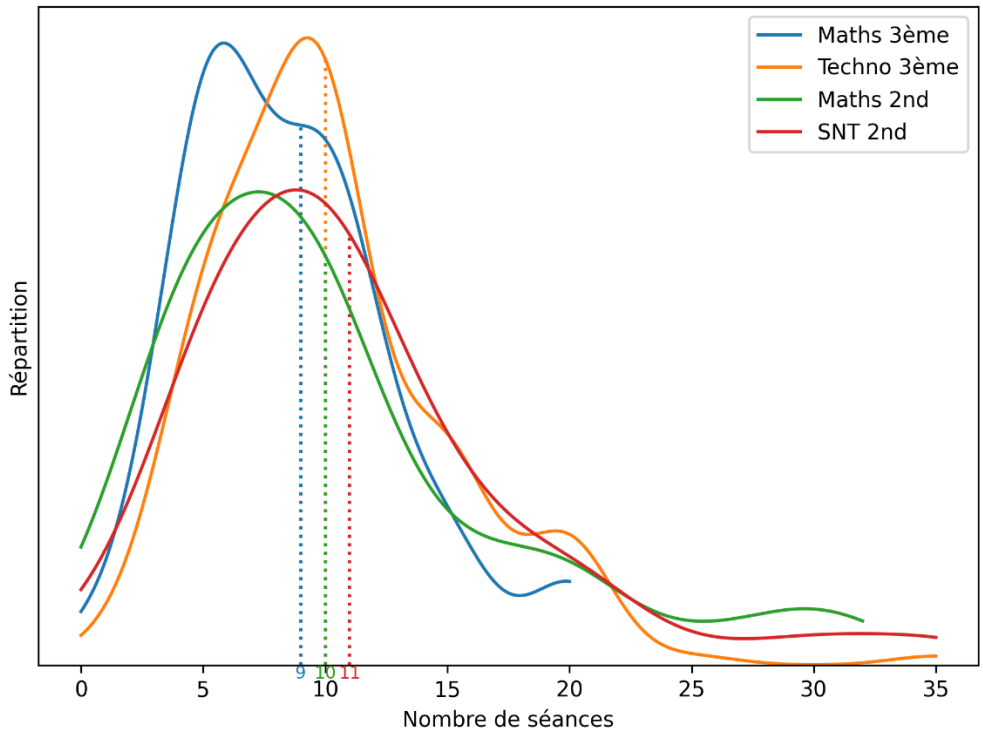


Figure 4-13 – Fonctions de répartition et moyennes du nombre de séances en lien avec la programmation informatique par discipline / QD1 : Combien de séances d'enseignement consacrez-vous à la programmation informatique sur l'année scolaire pour la discipline et le niveau concernés par cette enquête ?

Pour cette question, il s'agissait pour les enseignants d'estimer le nombre de séances pendant lesquelles les élèves sont amenés à utiliser la programmation informatique. Nous avions précisé dans l'aide de la question : « compter les séances pendant lesquelles les élèves sont amenés à utiliser la programmation informatique même si cela ne constitue pas l'objectif principal de la séance ». Les résultats représentés dans la Figure 4-13 attestent que, malgré des volumes horaires

hebdomadaires disciplinaires disparates (4h30 pour les mathématiques en troisième, 4h en seconde, 1h30 pour la technologie et la SNT), la moyenne du nombre de séances de programmation est étonnamment similaire pour les quatre disciplines : autour de 10 séances par an.

Ressources utilisées pour préparer les séances

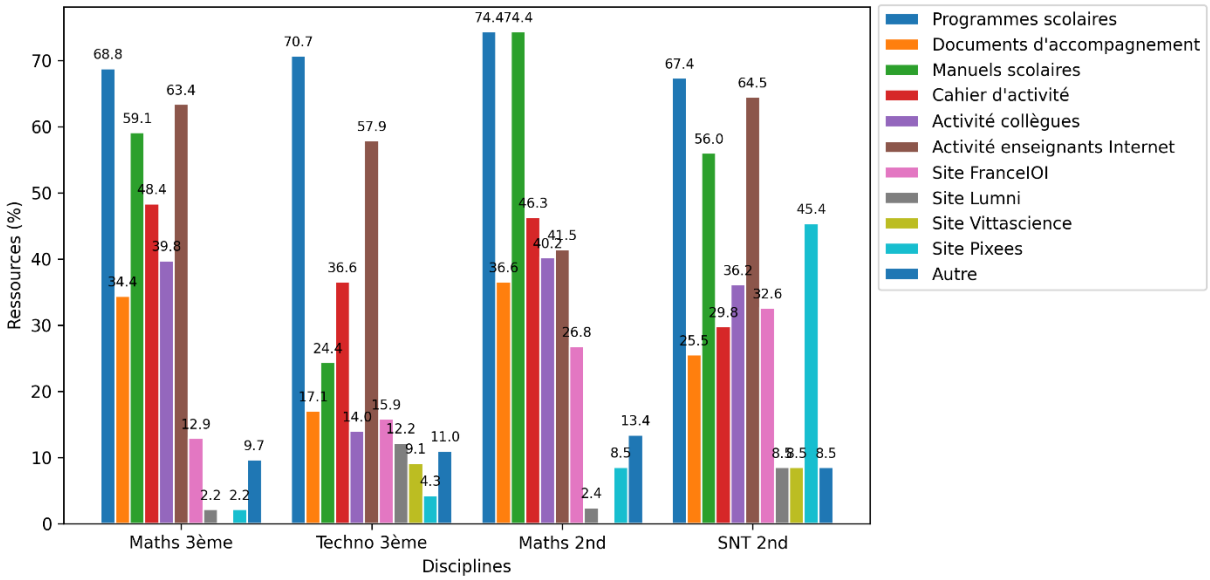


Figure 4-14 – Types de ressources utilisées lors de la préparation des séances en proportion et par discipline / QD2 : Sur quelles ressources vous appuyez-vous pour préparer vos activités de programmation informatique ?

Nous interrogeons dans cette question le « travail documentaire » des enseignants (Gueudet & Trouche, 2010) qui les amène à sélectionner, adapter et combiner des ressources afin d'organiser leurs séances d'enseignement de la programmation informatique.

En nous appuyant sur la Figure 4-14, nous pouvons d'abord constater que deux tiers à trois quarts des enseignants de notre échantillon utilisent les programmes d'enseignement pour guider la préparation de leurs activités.

Les ressources d'accompagnement sont moins exploitées. Ainsi, un peu plus d'un tiers des enseignants de mathématiques (34,4% en troisième et 36,6% en seconde), un quart des enseignants de SNT (25,5%) et 17% des enseignants de technologie les mobilisent dans leurs préparations.

Les manuels scolaires sont beaucoup mis à profit en mathématiques (59,1% en troisième et 74,4% en seconde) et sont complétés par des cahiers d'activités pour la moitié des enseignants (respectivement 48,4% et 46,3%). Les professeurs de technologie utilisent peu les manuels scolaires (24,4%) et les cahiers d'activités (36,6%). Enfin, en SNT, l'utilisation des manuels scolaires et des cahier d'activité est modérée (respectivement 56% et 29,8%).

Dans les quatre disciplines étudiées, les activités mises à disposition par leurs collègues sont des sources d'inspiration pour les enseignants (de façon moindre en technologie). Nous pouvons de plus constater un recours privilégié aux ressources partagées en ligne par les pairs dans toutes les disciplines (entre 41,5% et 64,5%).

Enfin, les sites Internet offrant des activités en ligne « clés en main », tels que FranceIOI, Lumni, VittaScience et Pixees, sont également employés de façon assez importante, et d'autant plus en seconde (pourcentages cumulés de 37,7% en mathématiques et de 94,5% en SNT).

Liens avec les autres domaines du programme

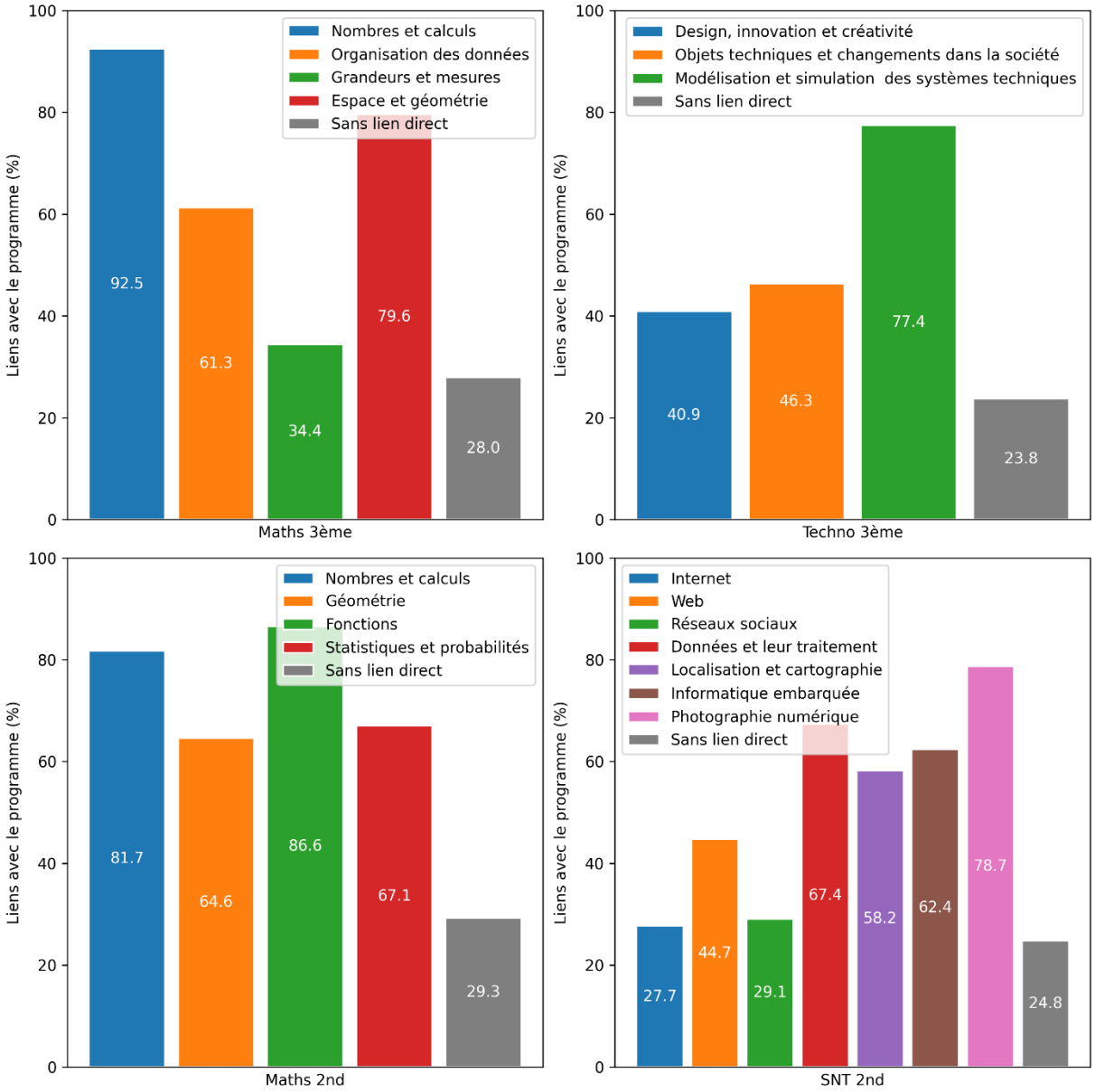


Figure 4-15 – Liens des activités de programmation avec les parties des programmes scolaires en proportion et par discipline / QD3-6 : Les activités de programmation informatique que vous proposez à vos élèves sont-elles en lien avec les autres thèmes du programme de mathématiques ?

Rappelons que notre analyse des textes officiels, et en particulier l'étude de l'habitat de la programmation informatique (voir section 2.5.5), montre qu'en troisième, le domaine de la programmation devrait être autonome (mathématiques), ou très peu en interaction avec les autres domaines du programme (technologie). À l'inverse, au lycée la programmation devrait être en interaction forte avec les autres domaines du programme.

Concernant le niveau troisième, les données issues de l'enquête représentées dans la Figure 4-15 semblent indiquer que, sur le terrain, la programmation informatique est amplement utilisée en lien avec tous les domaines du programme. Nous pouvons essayer d'expliquer cette situation par l'identité professionnelle (Zimmermann et al., 2012) des enseignants de mathématiques qui ne se voient pas comme des professeurs d'informatique. Ils auraient donc des difficultés à enseigner la programmation informatique en dehors de leur contexte disciplinaire. En revanche, en seconde, nous rencontrons une réalité plus en phase avec les indications officielles. À savoir, une programmation informatique au service des autres domaines du programme. Les domaines d'interactions semblent plutôt équilibrés en technologie, là où les trois premiers domaines sont moins impliqués par rapport aux quatre autres en SNT.

Notons une part commune à toutes les disciplines d'environ un quart des enseignants qui proposent des activités de programmation sans lien direct avec les autres parties du programme.

Type de démarches

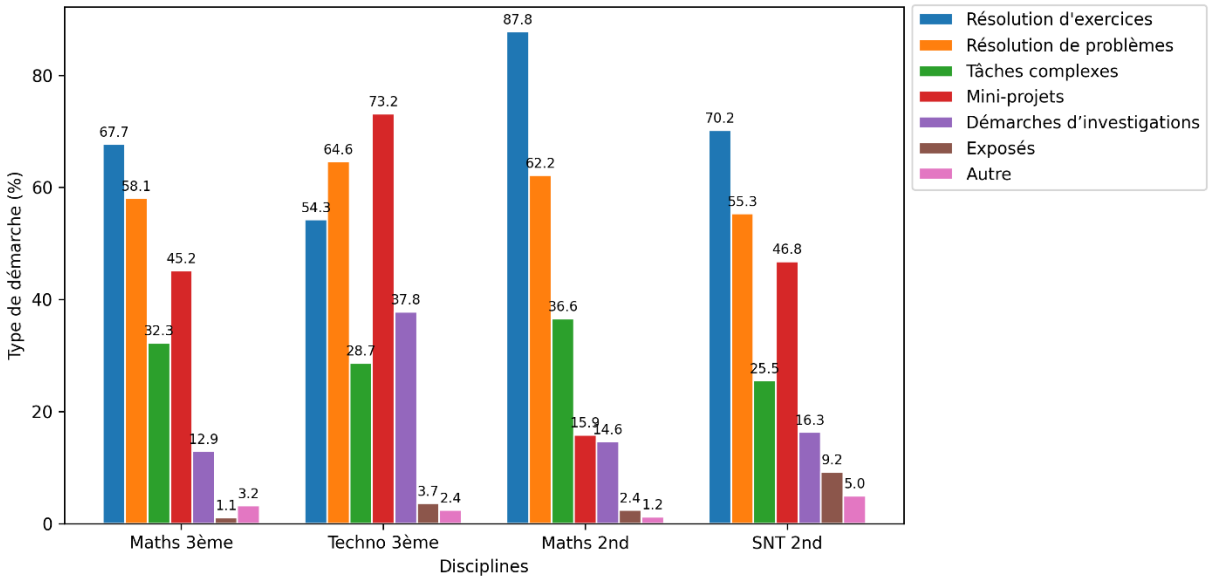


Figure 4-16 – Types de démarches proposées aux élèves par les enseignants en proportion et par discipline / QD7 : Quel type de démarche proposez-vous à vos élèves lors des activités de programmation ?

Nos analyses des textes officiels font apparaitre une scission entre la discipline mathématique qui se voit prescrire des activités de résolution d'exercices et de problèmes accompagnées de « mini-projets » d'un côté, et les disciplines technologie et SNT tournées vers des démarches plus variées : investigations, projets plus conséquents, exposés, etc. (voir section 2.8.5).

Les résultats de notre enquête, visibles dans la Figure 4-16, tendent à montrer que les activités classiques de résolution d'exercices et de problèmes restent largement majoritaires, en particulier en mathématiques en seconde. Les démarches de type mini-projets, investigations ou exposés sont, quant à elles, très présentes en technologie (quasi majoritaires) et apparaissent ponctuellement en SNT et en mathématiques en troisième. On retrouve ici aussi les cultures disciplinaires (Becher

& Trowler, 1989) propres aux mathématiques et à la technologie dans la forme des activités proposées aux élèves.

Type d'activités

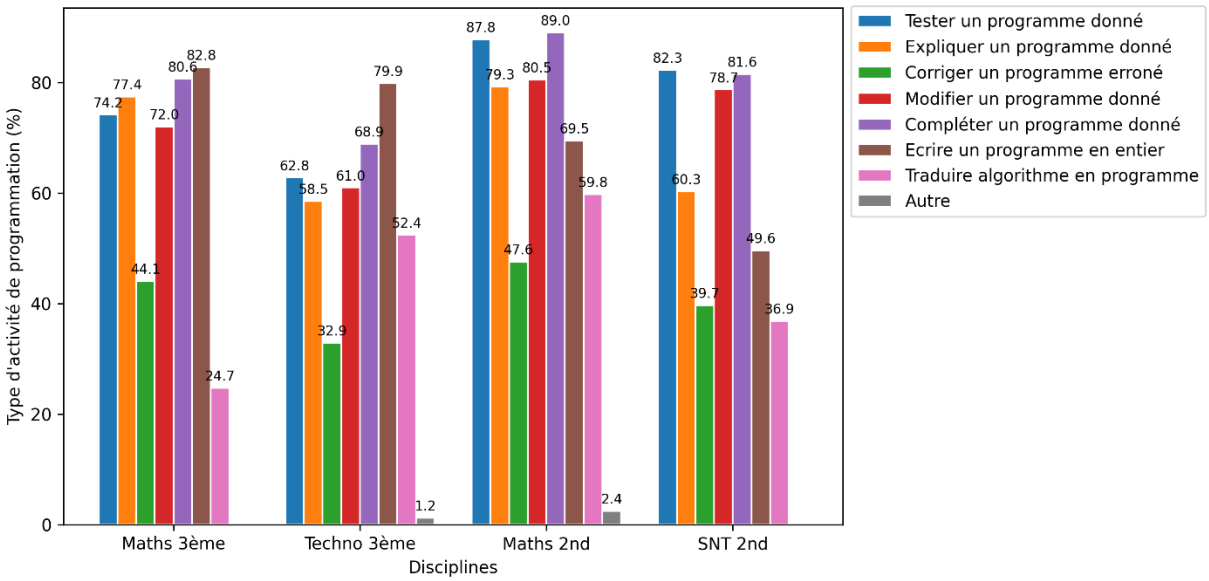


Figure 4-17 – Types d'activités de programmation proposées aux élèves par les enseignants en proportion et par discipline / QD8 : Quel type d'activité de programmation proposez-vous à vos élèves ?

Nos résultats concernant les types d'activités de programmation recommandées dans les textes officiels (voir section 2.8.5) indiquent qu'en mathématiques, les variations devraient être plus riches. Il s'agit en effet de compléter, d'améliorer, de corriger, de modifier, d'expliquer, d'écrire ou de tester un programme. Là où la technologie et les SNT disposent d'une palette d'activité plus restreinte : écrire, modifier, identifier et expliciter.

Force est de constater en analysant la Figure 4-17 que nous retrouvons dans toutes les disciplines une répartition variée et assez similaire des activités : il s'agit beaucoup d'agir sur des programmes existants en les testant, expliquant, modifiant ou complétant et, dans une moindre mesure, de les corriger ou de les traduire. L'activité consistant à écrire un programme en entier est également très présente en troisième et moins en seconde (en particulier en SNT). Il est probable que la modalité de programmation textuelle employée au lycée rende plus difficile la rédaction intégrale d'un programme.

Collaboration entre élèves

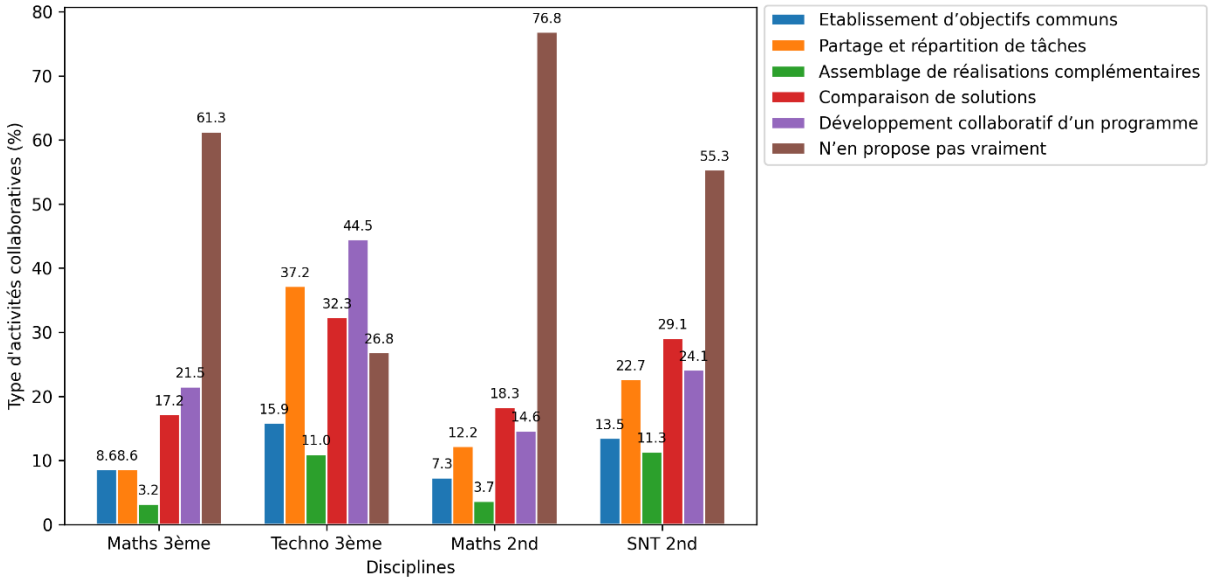


Figure 4-18 – Types d'activités collaboratives proposées aux élèves par les enseignants en proportion et par discipline / QD9 : Quel type d'activités collaboratives proposez-vous à vos élèves dans le cadre de la programmation informatique ?

Bien que la collaboration entre élèves soit citée dans les textes officiels de toutes les disciplines (voir section 2.8.5), nous retrouvons dans cette enquête une large majorité d'enseignants qui « ne proposent pas vraiment » d'activités collaboratives à leurs élèves (voir Figure 4-18). Exception faite de la technologie, et dans une moindre mesure, de la SNT pour lesquelles nous retrouvons quelques usages collaboratifs. Ces usages se concentrent sur le partage des tâches et la comparaison de solutions.

Mode de programmation

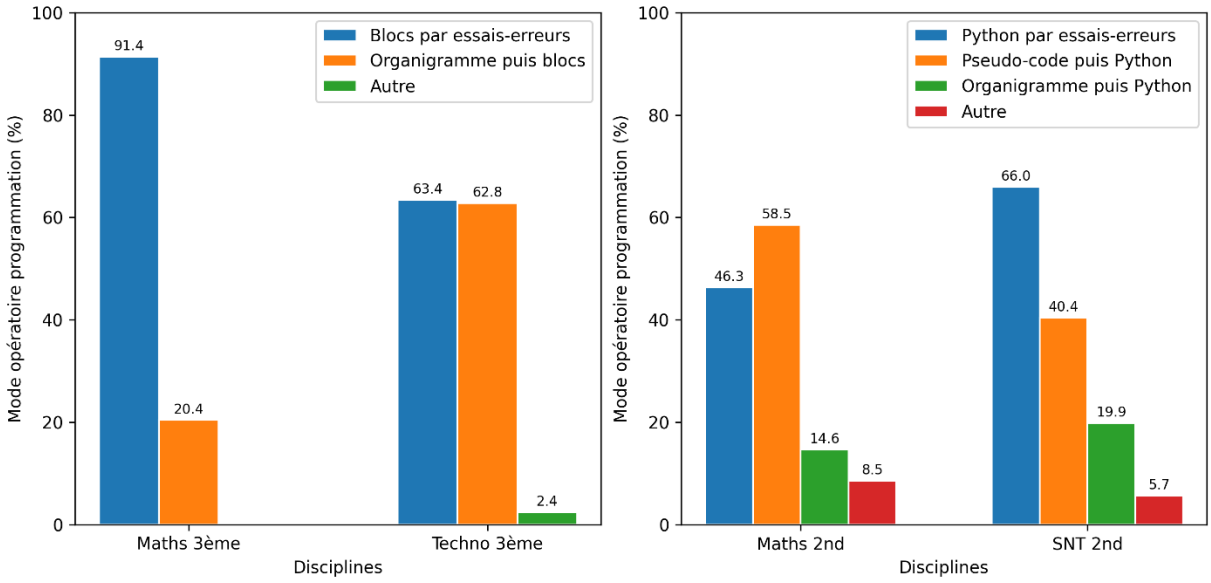


Figure 4-19 – Modes opératoires attendus par les enseignant lors du développement de programmes en proportion et par discipline / QD10-11 : Quel mode opératoire attendez-vous des élèves lorsqu'ils développent un programme informatique ?

Rappelons que les textes officiels indiquent explicitement pour les mathématiques en troisième un mode de programmation en blocs basé sur le tâtonnement et les essais-erreurs. En technologie et en mathématiques en seconde, un mode opératoire plus réfléchi est conseillé. Il consiste en une expression algorithmique de la solution du problème à résoudre (algorigramme en technologie et pseudo-langage en mathématiques en seconde) suivi d'une traduction dans un langage de programmation. Aucune consigne n'étant donnée pour la SNT.

La Figure 4-19 montre que concernant les trois premières disciplines, ces instructions semblent plutôt suivies, même si le mode opératoire par essais-erreurs s'avère bien implanté dans toutes les disciplines. En SNT, en l'absence de toute indication officielle, nous constatons que les trois modes opératoires sont mis en œuvre, avec une prédominance de la modalité par essais-erreurs.

Informatique débranchée

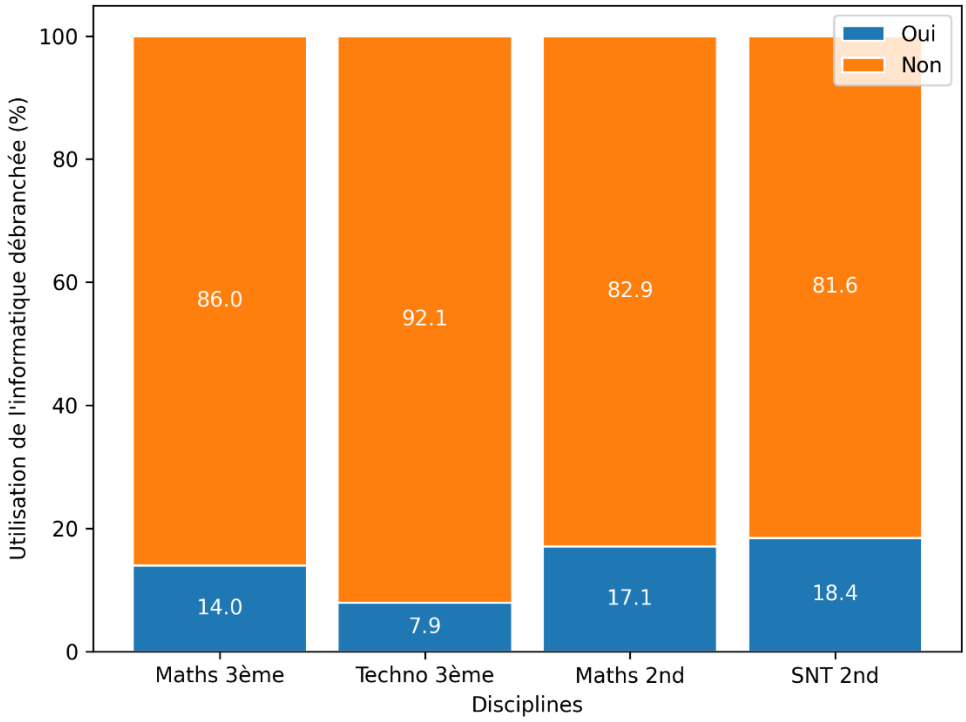


Figure 4-20 – Utilisation de l'informatique débranchée en proportion et par discipline / QD12 : Proposez-vous à vos élèves des activités de manipulation ludiques n'ayant pas recours à l'ordinateur (informatique débranchée) ? Si oui, décrire brièvement ces activités

Notre analyse des programmes d'enseignement a relevé des directives concernant la programmation débranchée exclusivement en mathématiques en troisième. Quelques exemples illustrent ces prescriptions, comme des déplacements guidés sur un quadrillage dans la cour de récréation ou un jeu d'instructions à suivre à l'aide d'un stylo (voir section 2.8.1). Dans les faits, les données dont nous disposons qui sont représentées dans la Figure 4-20 tendent à démontrer que l'informatique débranchée est une pratique peu courante dans toutes les disciplines (rare en technologie). Néanmoins, les activités les plus citées par les enseignants sont : le jeu du « Robot idiot », le jeu de Nim, le « Crêpier psychorigide », la formalisation algorithmique d'une activité du quotidien, le jeu « La guerre des Pythons », le suivi d'un programme de

dessin, ou la mise en œuvre incarnée d'algorithmes du programme de SNT : codage binaire, tri, graphe, routage, PageRank.

Synthèse de la sous-section

Q4.2 : Quelles sont les pratiques déclarées des enseignants concernant les activités d'enseignement de la programmation informatique ?

L'analyse des réponses des enquêtés met en évidence les résultats suivants concernant les activités déclarées de programmation :

- quelle que soit leur discipline, les enseignants consacrent en moyenne environ **dix séances par an** à la programmation informatique ;
- lors de la préparation de leurs séances, les enseignants disent se référer beaucoup aux **programmes d'enseignement** mais peu aux ressources d'accompagnement, ils mettent à profit les **manuels scolaires et les cahiers d'activités** (surtout en mathématiques), les **ressources mises à disposition en ligne** par leurs pairs, et des sites Internet offrant des **activités « clés en main »** (surtout en seconde) ;
- les activités de programmation déclarées sont **fortement en lien avec les autres domaines** du programme d'enseignement dans toutes les disciplines et ceci ne semble pas dépendre des préconisations officielles ;
- les démarches classiques de **résolutions d'exercices et de problèmes** sont **majoritaires** (surtout en mathématiques), les démarches plus **actives de type projets** sont **très présentes en technologie**, modérément en SNT et peu en mathématiques ;
- dans toutes les disciplines, les **types d'activités** de programmation sont **très variés**, il s'agit de tester, expliquer, compléter ou modifier un programme existant, la création complète d'un programme est également très présente (moins en seconde) ;
- les **activités collaboratives** sont **rares en mathématiques** et **plus courantes en technologie et en SNT**. Ces usages se concentrent sur le partage des tâches et la comparaison de solutions ;
- le mode de programmation mis en œuvre par les élèves de troisième en mathématiques est très majoritairement la **programmation directe par essais-erreurs**, ce mode est partagé dans les autres discipline avec une **formalisation préalable des algorithmes** (organigramme, pseudo-code) suivie d'une **traduction en langage** de programmation ;
- les enseignants de toutes les disciplines disent proposer **peu d'activités débranchées**.

4.4.5 Dispositif

Nous exposons dans cette sous-section les analyses des réponses aux questions du groupe E, en rapport avec les dispositifs d'enseignement.

Lieux

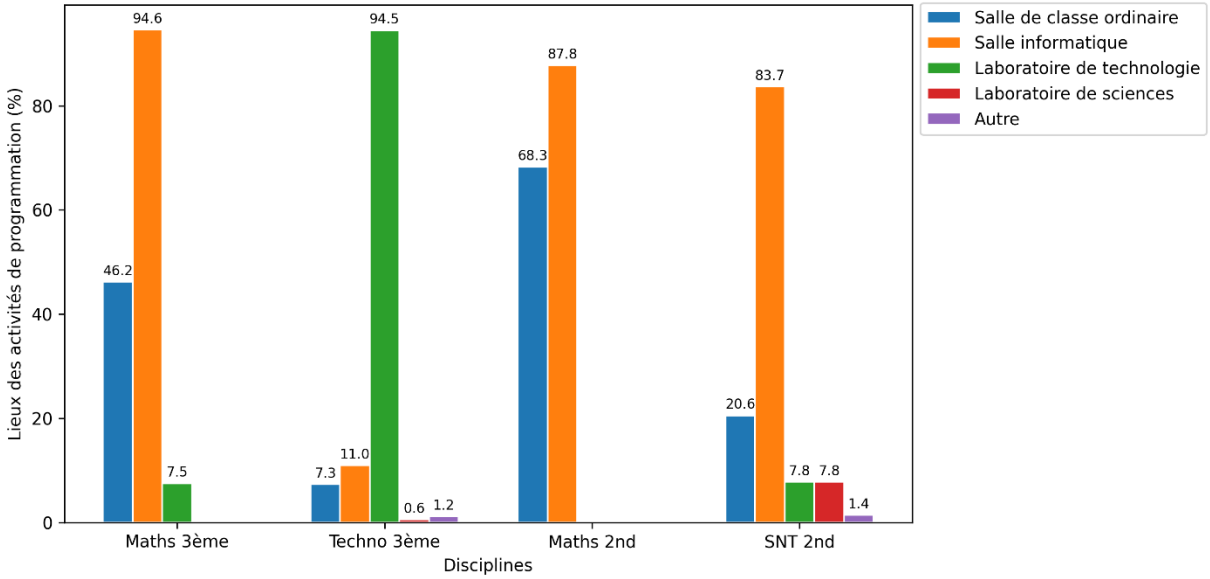


Figure 4-21 – Lieux où se déroulent les activités de programmation en proportion et par discipline / QE1 : Dans quels lieux se déroulent les activités de programmation informatique ?

Les réponses représentées dans la Figure 4-21 sont globalement en accord avec les éléments d'analyse des textes officiels concernant les lieux de programmation (voir section 2.8.5). Ainsi, les activités de programmation prennent place en mathématiques dans des salles de classe et des salles informatiques. Il est probable que la pratique de la programmation en salle de classe ait pour support la calculatrice, des ordinateurs portables ou des tablettes dans la mesure où la programmation débranchée est déclarée peu utilisée (voir plus loin les questions liées aux artefacts-supports). Les enseignants de technologie disposent, pour la grande majorité d'entre eux, de laboratoires dédiés. Les activités de programmation en SNT semblent, quant à elles, se dérouler dans des salles informatiques ou dans les laboratoires destinés aux autres disciplines scientifiques.

Notons que la forte utilisation des salles informatiques peut être expliquée par le très important taux d'équipement des établissements du second degré en France. Ainsi, d'après une étude de la DEPP (2022b, p. 45), en 2021 en France, 97% des collèges publics et 99% des lycées publics étaient dotés d'au moins une salle informatique.

Nombre d'élèves par classe

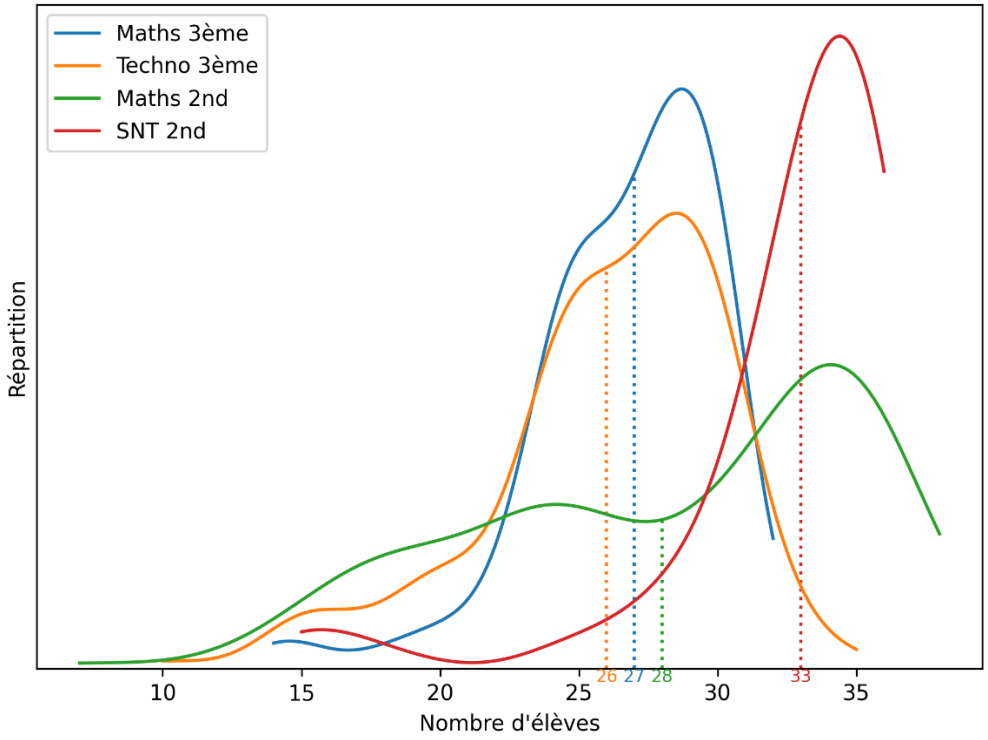


Figure 4-22 – Fonctions de répartition et moyennes du nombre d'élèves par classe par discipline / QE2 : De combien d'élèves sont constituées vos classes pour le niveau concerné par cette enquête ?

La Figure 4-22 indique la répartition du nombre d'élèves par classe selon les disciplines de l'enquête. Les classes de troisième de nos répondants sont en moyenne moins chargées (26 et 27) que les classes de seconde qui, malgré des moyennes de 28 et de 33, comptent régulièrement 35 élèves ou plus. Ces effectifs mettent en lumière le besoin d'heures en classe dédoublés (voir question suivante) afin de pouvoir mener à bien des activités de programmation dans les meilleures conditions (un élève par machine, plus de temps disponible pour chacun, etc.).

La moyenne des effectifs par classe pour notre échantillon est de 26,2 pour la troisième et de 29,9 pour la seconde. D'après un rapport de la DEPP (2021b, p. 35), en 2021 en France, les classes de troisième comportaient en moyenne 25,3 élèves et les classes de seconde 31,7 élèves. Ainsi, de ce point de vue notre échantillon est plutôt représentatif de la population.

Classes en demi-groupes

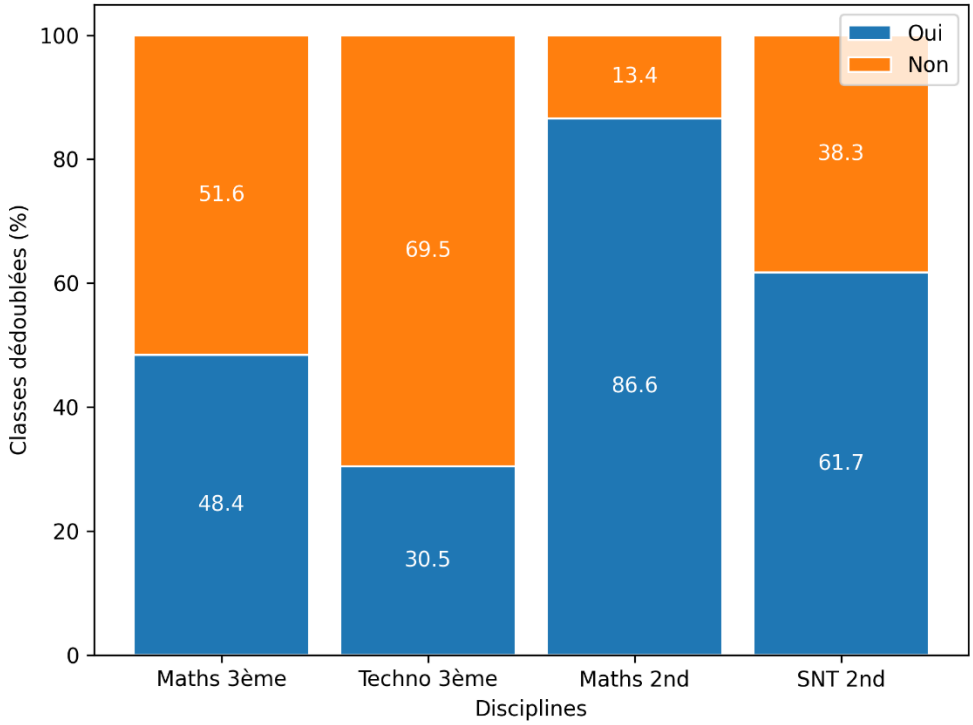


Figure 4-23 – Possibilités d'heures en classe dédoublée en proportion et par discipline / QE3 : Disposez-vous d'heures dédoublées pour le niveau concerné par cette enquête ? Ces heures pouvant être utilisées pour effectuer vos « travaux pratiques » en programmation informatique

En lien avec la question précédente, la Figure 4-23 montre que la grande majorité des enseignants de seconde (de façon plus systématique en mathématiques) disposent d'heures dédoublées en demi-groupe afin de mettre en œuvre des activités de programmation informatique. Cette proportion est moindre en troisième, en particulier en technologie.

Organisation de l'espace et rapport entre individus

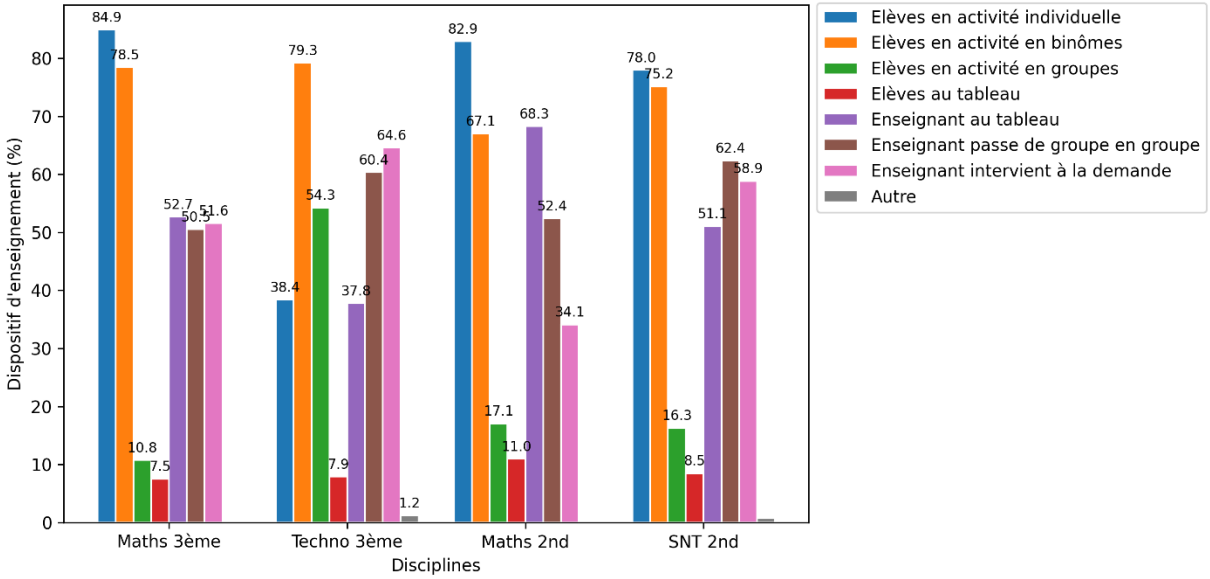


Figure 4-24 – Dispositifs d'enseignement mis en œuvre par les enseignants en proportion et par disciplines / QE4 : Quel dispositif d'enseignement mettez-vous en œuvre concernant les activités de programmation ?

La Figure 4-24 représente les réponses en lien avec l'organisation de l'espace et les interactions entre individus dans les dispositifs d'enseignement. En résonnance avec la question sur les activités collaboratives, nous retrouvons pour les mathématiques et la SNT sensiblement la même répartition des dispositifs d'enseignement avec une forte présence des dispositifs individuels et en binôme, et une présence faible des travaux de groupe. À l'inverse, les enseignants de technologie font la part belle aux travaux de groupe. Rappelons que les indications issues des textes officiels préconisent, en plus des travaux individuels, des travaux de groupe dans toutes les matières.

Concernant le rôle de l'enseignant, le dispositif majoritaire est celui de l'enseignant au tableau devant le groupe classe en mathématiques. En SNT et de façon plus prégnante en technologie, les enseignants sont plus souvent au contact des groupes d'élèves.

Synthèse de la sous-section

Q4.3 : Quelles sont les pratiques déclarées des enseignants concernant les dispositifs d'enseignement de la programmation informatique ?

En analysant les réponses à notre enquête concernant les dispositifs d'enseignement de la programmation informatique, nous avons obtenu les résultats suivants :

- les enseignements de programmation ont lieu majoritairement **en salle informatique** et en **salle de classe ordinaire** pour les **mathématiques**, quasi exclusivement dans le **laboratoire de**

technologie pour la **technologie**, et en **salle informatique** ou dans des **laboratoires de sciences** pour les **SNT** ;
- les **effectifs par classe** pour nos répondants sont en moyenne de **26,2 en troisième** et **29,9 en seconde** ;
- la grande majorité des enseignants de **seconde** (surtout en mathématiques) ont la possibilité de donner des **heures de cours dédoublés en demi-groupe**, ce qui est plus **rare en troisième** (en particulier en technologie) ;
- concernant l'organisation de l'espace et les interactions, les **élèves** de **mathématiques** et de **SNT** travaillent majoritairement de manière **individuelle ou en binôme**, et les **enseignants** se trouvent **au tableau** devant le groupe classe, en **technologie**, les activités de **groupe ou en binômes** sont **majoritaires** pour les élèves, et les **enseignants** sont plus souvent à leur **contact direct**.

4.4.6 Artefacts

Dans cette dernière sous-section nous analysons les réponses aux questions du groupe F en lien avec les artefacts mis à disposition des élèves. Commençons par les artefacts-supports, ceux qui permettent de formaliser des algorithmes et de fabriquer des programmes : machines, environnements d'édition des programmes, langages de programmation, etc.

Machine de développement

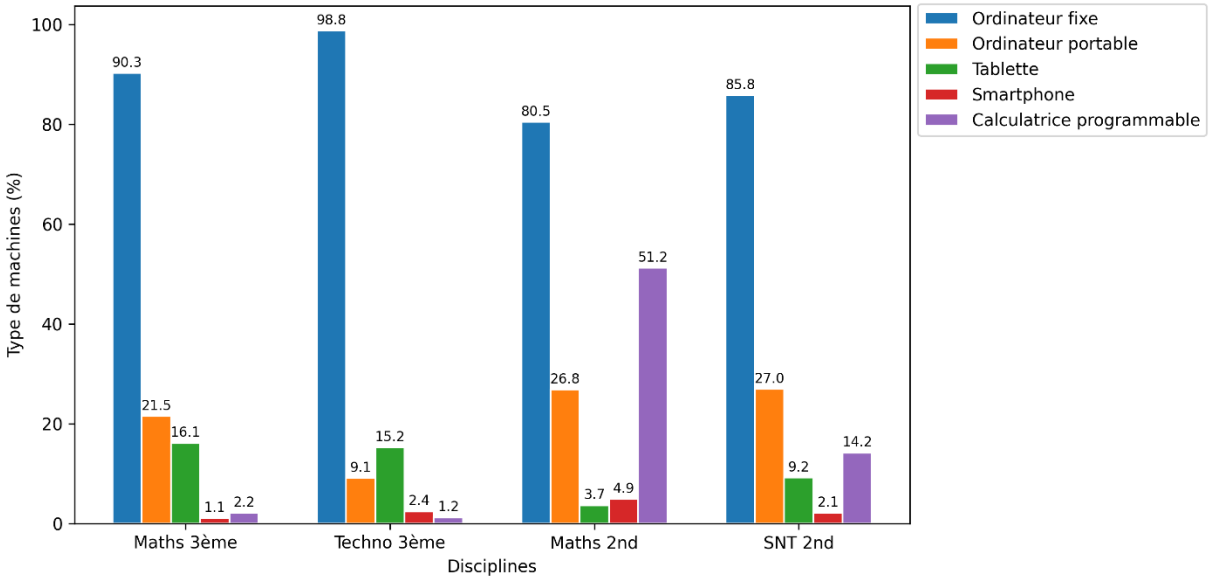


Figure 4-25 – Types de machines utilisées par les élèves pour programmer en proportion et par disciplines / QF3 : Quel type de machine vos élèves utilisent-ils pour développer leurs programmes informatiques ?

Selon la Figure 4-25, dans les quatre disciplines, le développement des programmes se fait dans plus de 80% des cas sur des ordinateurs de bureau. En seconde, environ 30% des enseignants utilisent des classes mobiles ayant pour support

des ordinateurs portables, cette proportion est plus faible en troisième (21% à 9%). En revanche en troisième, dans environ 15% des cas, ces classes mobiles sont constituées de tablettes, contre 4% à 9% en seconde. Les langages utilisés (Scratch ou Python) ont vraisemblablement une influence sur le type de terminaux mobiles utilisés. Remarquons enfin que les calculatrices programmables sont exploitées au lycée, et de façon très courante en mathématiques (51,2%). Nous avançons l'hypothèse que la possibilité nouvelle de programmer directement en Python sur les calculatrices encourage l'adoption de cette modalité de programmation.

	Nombre de terminaux fixes pour 100 élèves	Nombre de terminaux mobiles pour 100 élèves	Nombre d'outils de vidéoprojection pour 100 élèves
Écoles maternelles	3	3	1
Écoles élémentaires	5	9	4
Écoles primaires	5	10	4
Ensemble 1er degré	**5**	**8**	**3**
Collèges	23	17	6
LEGT	43	23	7
LP	71	26	11
Ensemble 2nd degré	**33**	**20**	**7**

Tableau 4-3 – Indicateurs de l'équipement du numérique du secteur public en France pour l'année scolaire 2021-2022 (DEPP, 2022b, p. 45)

Le Tableau 4-3 est tiré d'une étude de la DEPP (2022b). Il concerne l'équipement numérique des collèges et des lycées en 2021. Il est intéressant de constater que les terminaux fixes (« ordinateur de bureau type PC ») sont quasiment deux fois plus disponibles au lycée qu'au collège (43 contre 23 pour 100 élèves), bien que moins utilisés dans notre étude. Nous ne retrouvons pas ce déséquilibre au niveau des terminaux mobiles (« un ordinateur portable, un net book, un ultrabook, une tablette, un terminal de classe mobile ou tout objet mobile ») qui ont des disponibilités presque équivalentes au collège et au lycée (17 contre 23 pour 100 élèves).

Langage de programmation

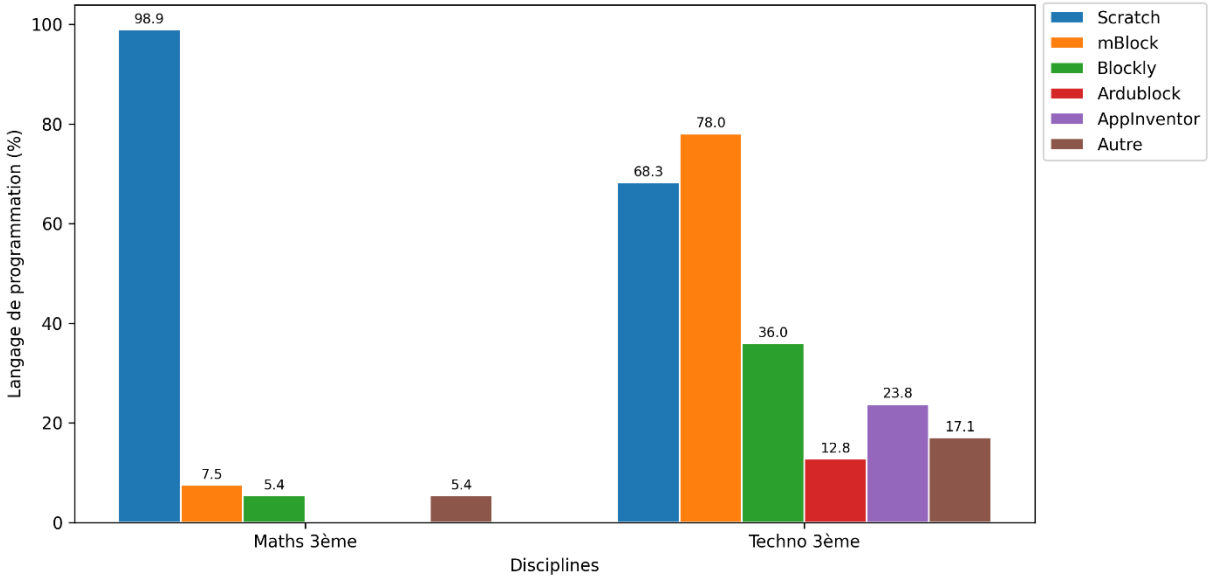


Figure 4-26 – Langages de programmation utilisés par les élèves en proportion par discipline du collège / QF8 : Quel(s) langage (s) de programmation graphique vos élèves utilisent-ils ?

Comme le montre la Figure 4-26 qui porte sur les langages de programmation utilisés en troisième, les élèves se servent quasi exclusivement du langage Scratch en mathématiques (98,9%). Cette très forte utilisation de Scratch est conforme aux prescriptions des programmes d'enseignement (voir section 2.8.5). En technologie, les usages sont plus diversifiés (programmation de robots, cartes, objets domotiques, etc.), par conséquent les langages employés sont plus variés, chacun d'entre eux permettant de piloter des artefacts-cibles différents. Cette diversité des langages est également en phase avec les indications officielles (voir section 2.8.5).

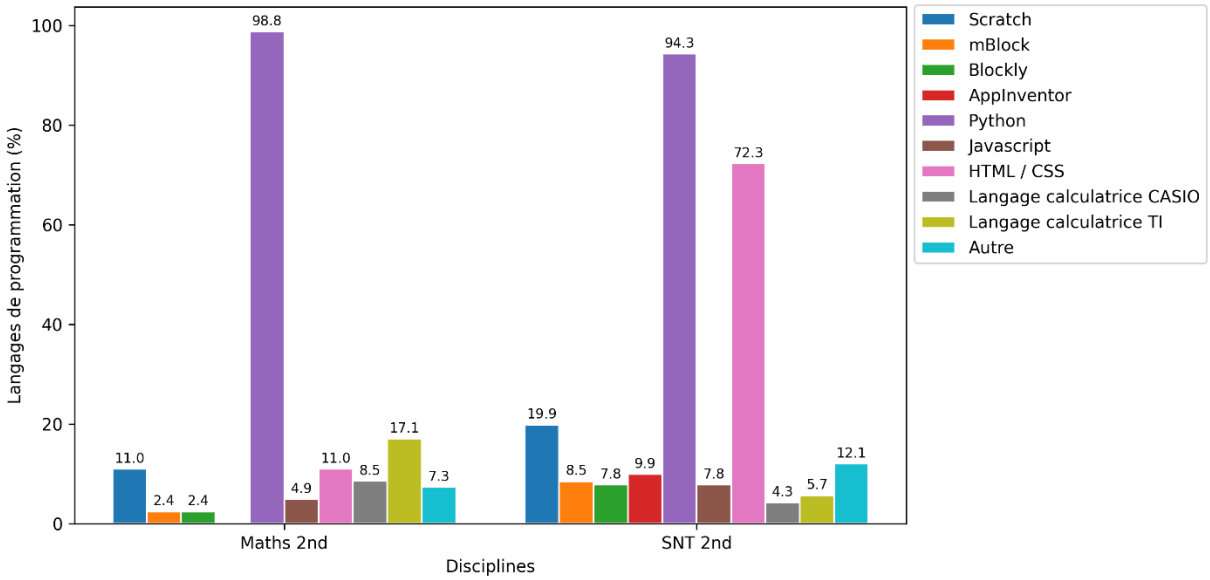


Figure 4-27 – Langages de programmation utilisés par les élèves en proportion par discipline du lycée / QF9 : Quel(s) langage(s) de programmation vos élèves utilisent-ils ?

Comme nous pouvons le voir sur la Figure 4-27, en seconde, en mathématiques comme en SNT, Python est massivement employé dans les classes (respectivement 98,8% et 94,3%). En mathématiques, nous pouvons également remarquer l'utilisation des langages de programmation propres aux calculatrices CASIO et TI (respectivement 8,5% et 17,1%). Il faut sans doute ajouter à cela l'usage des calculatrices qui permettent de coder directement en Python (voir ci-dessous). En SNT, nous retrouvons une forte présence du couple HTML/CSS certainement utilisé dans la partie du programme dédiée au Web. Notons également la pratique marginale de programmation par blocs (Scratch, mBlock, Blockly et AppInventor) en particulier en SNT. Nous avançons l'idée que cette programmation est utilisée dans un premier temps afin de faciliter la transition vers Python, ou en SNT pour piloter des artefacts qui ne sont pas adressables en Python.

Bibliothèque

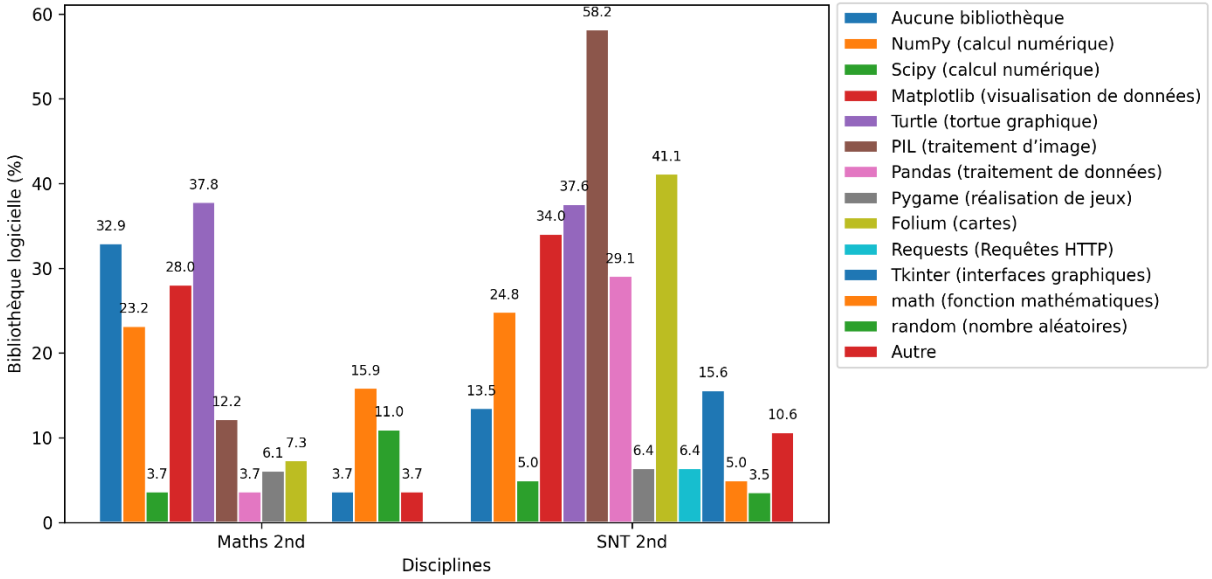


Figure 4-28 – Utilisation des bibliothèques Python par les élèves en proportion et par discipline au lycée / QF2 : Vos élèves utilisent-ils des bibliothèques Python? Si oui, lesquelles?

La Figure 4-28 porte sur les bibliothèques Python utilisées en seconde. Nous observons qu'en mathématiques, les librairies logicielles dont les élèves se servent le plus sont celles en rapport avec les calculs (NumPy, math, random), la géométrie (Turtle), et la visualisation des données (Matplotlib). Cette utilisation est totalement cohérente avec les indications officielles en termes d'interaction avec les autres domaines du programmes (voir section 2.5.5). Les déclarations de certains enseignants de mathématiques concernant des bibliothèques de traitement d'image, de jeu ou des cartographies sont pour le moins surprenantes.

En SNT, nous retrouvons les bibliothèques de traitement d'images (PIL), de cartographie (Folium), de visualisation et de traitement de données (Matplotlib, Pandas), d'interface graphique (Tkinter), mais également quelques bibliothèques liées aux mathématiques (NumPy et Turtle). Ceci est en phase avec, à la fois les prescriptions d'activités enchevêtrées dans les autres domaines du programme (voir section 2.5.5), mais également avec les réponses données précédemment (section 4.4.4) concernant la forte utilisation de la programmation dans les activités de traitement d'images, de traitement des données et de cartographie.

Environnement de développement

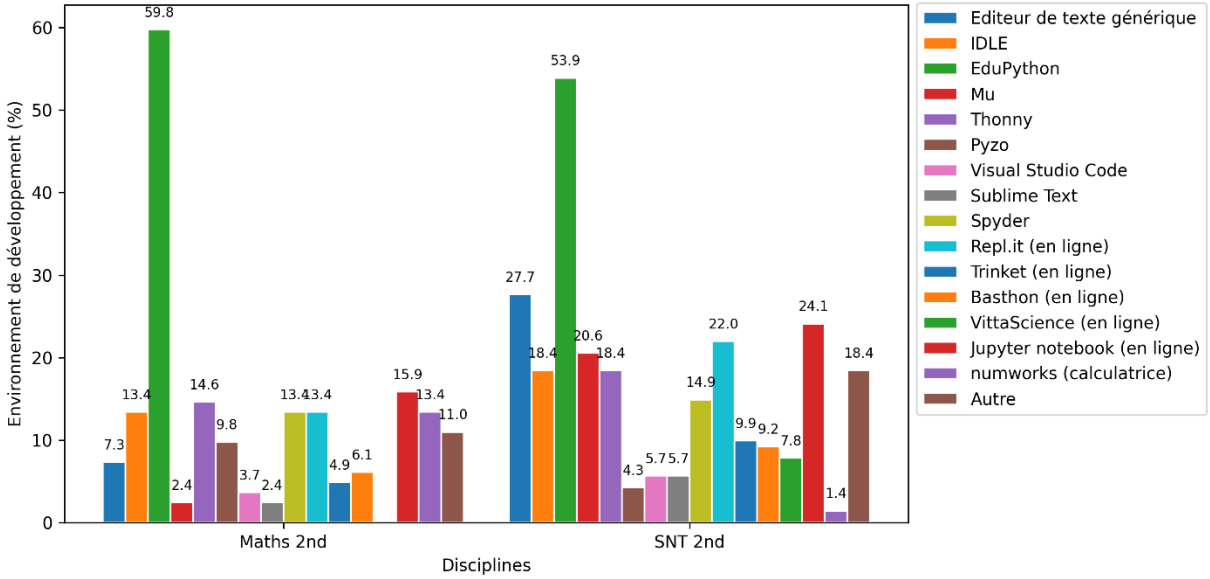


Figure 4-29 – Environnements de programmation utilisés par les élèves en proportion et par discipline en seconde / QF1 : Quel environnement de développement est utilisé par vos élèves ?

Comme le montre la Figure 4-29, au niveau seconde, en mathématiques comme en SNT, l'environnement de développement le plus courant est EduPython. Il s'agit d'un logiciel conçu par un groupe d'enseignants qui vise l'introduction de la programmation Python au lycée. Il est tout-en-un et simple à installer. Il offre la coloration syntaxique, la complétion automatique ainsi que la vérification syntaxique en direct du code. Il inclut de plus de façon native les bibliothèques Python les plus utilisées au lycée (PIL, Folium, Matplotblib, Pandas, etc.).

L'éditeur de texte générique (sans coloration syntaxique) est ensuite utilisé par 27% de enseignants de SNT et 7% des enseignants de mathématiques. Cette pratique est étonnante au regard de l'offre pléthorique disponible pour les éditeurs de code. Nous avançons l'hypothèse que ces enseignants souhaitent que les élèves puissent se concentrer sur la syntaxe du langage sans l'intervention d'artefacts d'assistance tels que la coloration syntaxique ou la complétion automatique du code.

Viennent ensuite les éditeurs pédagogiques qui ciblent les débutants (IDLE, MU, Thonny, Pizo), puis les éditeurs en ligne qui ne nécessitent pas d'installation (Repl.it, Trinket, Basthon, VittaScience). Remarquons l'utilisation de la forme particulière du notebook (Jupyter, Basthon, Colaboratory), venu du monde de la science des données, par 24% des enseignants de SNT et 16% de mathématiques. Enfin, les éditeurs les moins utilisés sont logiquement ceux, plus complexes, issus du monde professionnel (Visual Studio Code, Sublime Text, Spyder).

Intéressons-nous désormais aux questions portant sur les artefacts-cibles utilisés par les élèves, c'est-à-dire les artefacts contrôlés par les programmes : cartes, robots, objets domotiques, etc. Au regard des programmes d'enseignement, cette question n'était posée qu'aux enseignants de technologie et se SNT.

Cartes programmables

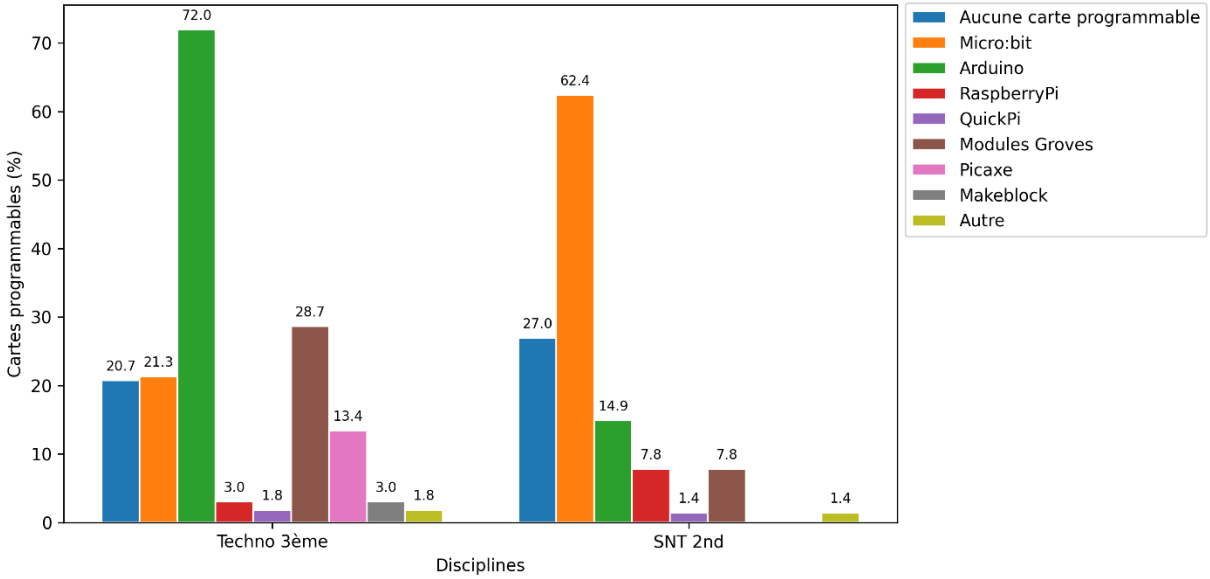


Figure 4-30 – Cartes programmables à disposition des élèves en proportion en technologie et en SNT / QF4 : Mettez-vous à disposition des élèves des cartes programmables ? Si oui, lesquelles ?

Commençons par remarquer dans la Figure 4-30 qu'environ trois quarts des enseignants de technologie et de SNT proposent la programmation de cartes à leurs élèves. En technologie, c'est la carte Arduino qui domine, associée aux modules Groves. La possibilité de programmer ces cartes à l'aide de blocs (Ardublock) et leur faible coût d'achat sont des hypothèses pouvant expliquer cette prépondérance. En SNT, la carte Micro:bit est la plus utilisée. Le fait qu'elle soit facilement programmable en Python, qu'elle dispose de nombreux capteurs et actionneurs adaptables, et que son prix soit faible sont autant d'atouts qui peuvent éclairer ce choix.

Robots programmables

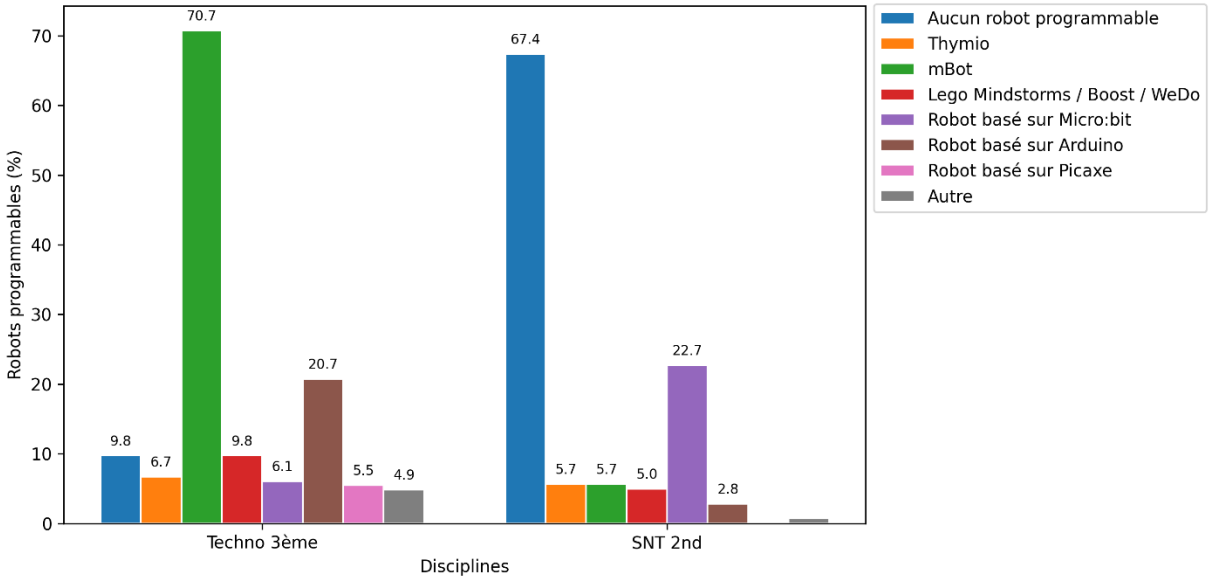


Figure 4-31 – Robots programmables mis à disposition des élèves en proportion en technologie et en SNT / QF5 : Mettez-vous à disposition des élèves des robots programmables ? Si oui, lesquels ?

La Figure 4-31 représente les réponses concernant les robots. En technologie, la grande majorité des élèves programment des robots, les modèles mBot dominant très largement les usages. En SNT, les robots sont peu employés, sauf en complément de la carte Micro:bit qui est massivement adoptée. Ces pratiques sont conformes aux indications données par les textes officiels (voir section 2.8.5), avec des prescriptions visant explicitement la programmation de robots en technologie et plus largement les objets connectés en SNT.

Systèmes domotiques programmables

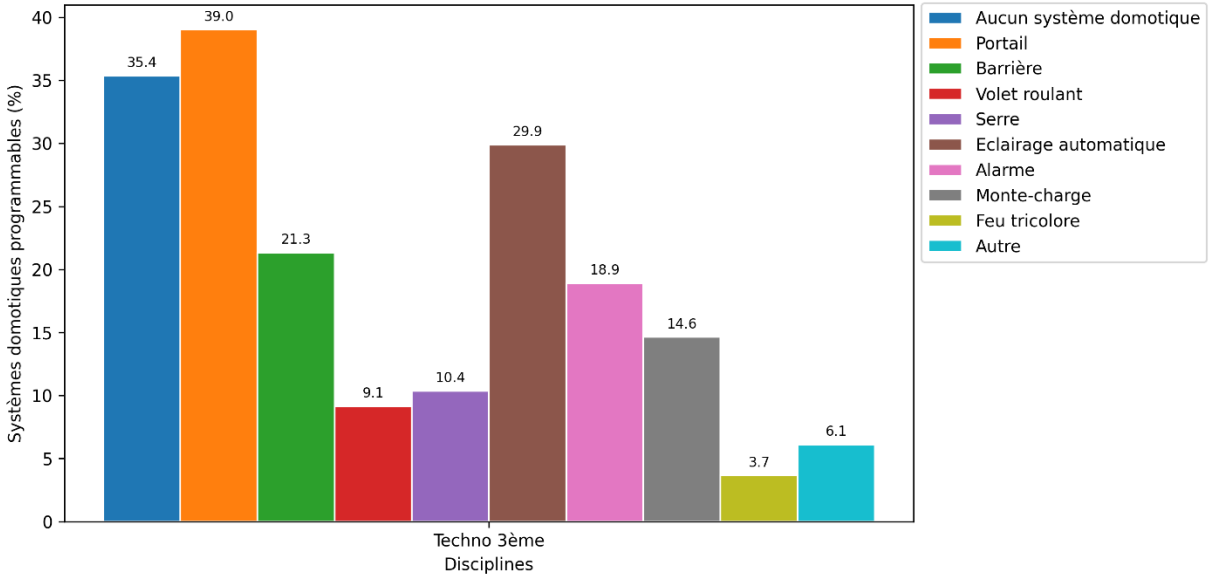


Figure 4-32 – Systèmes domotiques programmables à disposition des élèves en proportion en technologie / QF6 : Mettez-vous à disposition des élèves des systèmes domotiques programmables (maquette ou réels)? Si oui, lesquels ?

Comme le montre la Figure 4-32, en technologie, la majorité des élèves sont amenés à programmer des systèmes domotiques. Les portails et les éclairages automatiques étant les plus cités. Ces systèmes technologiques sont également explicitement nommés dans les programmes d'enseignement (voir section 2.8.5).

Navigateurs Internet

Pour terminer, nous avions posé une question sur les navigateurs Internet disponibles dans les établissements pour les élèves. Ceci dans le but d'orienter d'éventuels développements informatiques pour la conception d'une ressource d'enseignement.

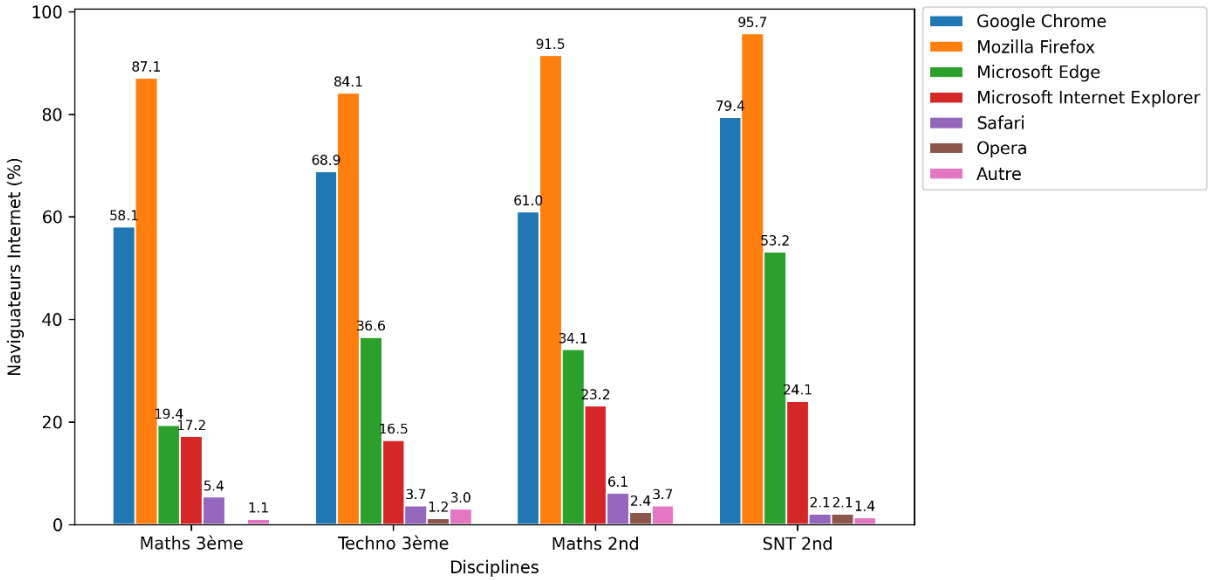


Figure 4-33 – Navigateurs Internet disponibles en proportion par discipline / QF10 : Lesquels de ces navigateurs Internet sont à disposition de vos élèves dans votre établissement ?

La Figure 4-33 montre que les élèves ont à leur disposition dans leur grande majorité les navigateurs Internet Mozilla Firefox et Google Chrome, suivis par les deux navigateurs de Microsoft : Edge et Internet Explorer. Notons une utilisation marginale de Safari, qui semble indiquer un faible emploi du matériel de marque Apple.

Synthèse de la sous-section

Q4.4 : Quelles sont les pratiques déclarées des enseignants concernant les artefacts utilisés pour l'enseignement de la programmation informatique ?

Suite à l'analyse des réponses portant sur les artefacts de notre questionnaire, nous retenons les éléments suivants :

- les **machines** utilisées pour développer les programmes sont très majoritairement des **ordinateurs de bureau** dans toutes les disciplines, on retrouve également des classes mobiles constituées plutôt de **tablettes** en **troisième** et d'**ordinateurs portables en seconde**, la **calculatrice** est également un support de programmation répandu en seconde, en particulier en mathématiques ;
- le **langage de programmation** utilisé en mathématiques au **collège** est presque exclusivement **Scratch**, on retrouve néanmoins une forte diversité de langages de blocs en technologie, en **seconde**, **Python** domine largement les usages et est complété en **SNT** par le couple **HTML/CSS** ;
- en seconde, en **mathématiques**, les élèves utilisent des **librairies** logicielles de **calcul**, de **géométrie** et de **visualisation de données**, et en **SNT** des bibliothèques de **traitement d'images**, de **cartographie** et de **traitement de données** ;

- les **environnements de programmation** les plus employés par les élèves de seconde sont les **éditeurs pédagogiques simplifiés** tels que EduPython, les **éditeurs de texte basiques** sont également utilisés (surtout en SNT), ainsi que les **éditeurs en ligne** comme les notebooks Jupiter ;
- concernant les **artefacts-cibles**, deux tiers des enseignants de technologie et de SNT font utiliser à leurs élèves des **cartes programmables** (majoritairement Arduino en troisième et Micro:bit en seconde), les **robots** ainsi que les **systèmes domotiques** sont également largement programmés en **technologie**.

Nous venons de présenter les analyses des résultats de notre enquête en fonction des différents aspects de l'enseignement de la programmation. Voyons dans la section suivante les conséquences que nous pouvons en tirer pour la conception de ressources d'enseignement.

4.5 Conséquence pour la conception de ressources

La deuxième partie de cette thèse porte sur la création d'une ressource d'enseignement de la programmation informatique à la transition collège-lycée. Nous ciblons pour cette ressource les élèves de seconde en début d'année scolaire, pour une première approche de la programmation en Python. Nous décrivons ici les enseignements que nous pouvons tirer des analyses que nous venons d'exposer afin de nourrir cette conception.

4.5.1 Discipline

D'abord, comme précisé dans la section 4.4.3, contrairement aux indications des textes officiels, aucune discipline de seconde ne prend prioritairement en charge l'introduction des concepts de programmation en seconde. Par conséquent, notre ressource devra pouvoir être utilisée indistinctement par les enseignants de mathématiques et de SNT.

4.5.2 Durée

Ensuite, les enseignants de seconde consacrent en moyenne dix séances annuelles à la programmation informatique (section 4.4.4). Dans ce contexte, il semble réaliste de mettre à leur disposition une ressource d'enseignement visant spécifiquement une première approche de Python sur une durée de deux séances de cours. Le reste des séances pouvant être utilisé, comme l'indiquent les textes officiels, par des activités en lien avec les autres domaines des programmes.

4.5.3 Type de ressource

Les sites Internet offrant des activités en ligne « clés en main » tels que FranceIOI, Lumni, VittaScience ou Pixees sont largement employés par les enseignants de seconde de notre échantillon pour mettre leurs élèves en activité (section 4.4.4). En se basant sur ce constat, notre ressource d'enseignement pourra également prendre la forme d'une application en ligne.

4.5.4 Type d'activité

Comme nous venons de le montrer dans la section 4.4.4, la création complète d'un programme n'est demandée aux élèves que par 50% à 70% des enseignants de seconde interrogés. Il s'agit pourtant ici d'un objectif prescrit dans les programmes d'enseignement de mathématiques et de SNT : « Une consolidation des acquis du cycle 4 est proposée autour de deux idées essentielles la notion de fonction et la programmation comme production d'un texte dans un langage informatique » (MEN, 2019c, p. 14) et « Capacité attendues : écrire et développer des programmes pour répondre à des problèmes » (MEN, 2019d, p. 3). Ainsi, en accord avec les programmes, nous considérons qu'il est important que les élèves pratiquent l'écriture de programmes entiers. Par conséquent la ressource que nous allons concevoir devra privilégier ce type d'activité.

4.5.5 Autonomie

Malgré la possibilité de séances en demi-groupe dédiées à la programmation, les classes sont très chargées au niveau seconde (voir section 4.4.5). La recherche a montré que la maîtrise de la programmation informatique était difficile (Bennedsen & Caspersen, 2007; McCracken et al., 2001). De plus, au cours de ce processus d'apprentissage, le support et les retours des enseignants sont essentiels (Hattie & Timperley, 2007). Compte tenu du fait que les enseignants ne disposent que d'un temps limité pour chaque élève, notre ressource devra être en mesure de favoriser le travail des élèves en autonomie, c'est-à-dire sans appeler le professeur.

4.5.6 Environnement cible

Au regard du fort taux d'équipement en salles informatiques et autres classes mobiles des lycées français (voir section 4.4.6), notre ressource d'enseignement sera facilement accessible à l'ensemble des élèves si elle prend la forme d'une application disponible sur Internet. Ajoutons qu'en seconde, les machines de développement utilisées sont essentiellement des ordinateurs fixes et portables (voir section 4.4.6), ainsi les tablettes et les smartphones ne seront pas des cibles prioritaires pour cette application. Enfin, les navigateurs Internet Mozilla Firefox et Google chrome étant très largement disponibles dans les lycées, nous pourrons porter nos efforts de développement sur ces deux logiciels.

4.5.7 Environnement d'édition

Pour finir, les éditeurs de programme employés au lycée comportent pour la plupart des aides à l'édition telles que la coloration syntaxique ou la complétion automatique du code. Comme nous l'avons vu dans la section 3.4.2, en comparaison avec l'assemblage des blocs de Scratch, la création et le maintien de la structure globale des programmes Python est source d'erreurs, et le processus mécanique de saisie au clavier peut représenter un défi cognitif et moteur pour les débutants. Il faudra par conséquent intégrer dans notre application des éléments accompagnant la rédaction des programmes afin de soutenir la transition de Scratch vers Python.

4.5.8 Synthèse de la section

Q4.5 : Quels enseignements peut-on tirer des réponses aux questions précédentes concernant la création d'une ressource d'apprentissage ?

En partant des analyses des résultats de l'enquête que nous avons produites, nous avons établi des recommandations pour la conception d'une ressources d'introduction à programmation en Python en classe de seconde. Cette ressource :

- devra pouvoir être mise en œuvre **indistinctement** par les enseignants de **mathématiques** et de **SNT** ;
- devra être conçue pour une **durée de deux séances** d'une heure maximum ;
- pourra prendre la forme d'une **application en ligne** ;
- devra privilégier l'**écriture complète de programmes** ;
- devra favoriser l'**autonomie** des élèves dans un contexte de classes chargées ;
- devra être développée prioritairement pour les **ordinateurs fixes et portables** et les navigateurs Internet Mozilla **Firefox** et Google **Chrome** ;
- devra proposer un **dispositif d'édition** à même de **soutenir** la **transition Scratch-Python**.

Dans ce chapitre, nous avons mené une enquête par questionnaire en ligne auprès de 480 enseignants impliqués dans l'enseignement de la programmation informatique à la transition collège-lycée. En analysant les réponses à ce questionnaire, nous avons donné à voir une certaine réalité des pratiques enseignantes selon les quatre indicateurs de contenus, d'activités, de dispositifs et d'artefacts. Nous avons ensuite tiré les enseignements de ces résultats en vue de concevoir une ressource d'enseignement destinée à l'introduction du langage Python en classe de seconde.

Dans le chapitre suivant, nous détaillons la conception et l'évaluation de cette ressource lors d'expérimentations sur le terrain, puis sa diffusion plus large dans les classes.

Chapitre 5

Conception, évaluation et diffusion de l'application Pyrates

Selon la méthodologie de l'ingénierie didactique qui structure cette thèse, après les analyses préalables qui occupent les quatre premiers chapitres, viennent les étapes de la réalisation didactique puis de son évaluation. Ainsi, nous exposons dans ce chapitre la conception d'une ressource d'enseignement de la programmation informatique qui vise à accompagner la transition du collège au lycée. Cette ressource prend la forme d'une application en ligne proposant un serious game d'introduction au langage Python à destination des élèves de seconde. Nous présentons ensuite l'évaluation de cette ressource à partir des traces d'activités des élèves collectées lors d'expérimentations que nous avons encadrées sur le terrain, ainsi que des programmes exécutés par les utilisateurs finaux. Enfin, nous évoquons la large diffusion de l'application auprès des élèves dans les classes en France et à l'étranger. Les aspects de la conception que nous allons détailler et évaluer dans ce chapitre sont la conception didactique des différents niveaux du jeu ainsi que l'aménagement de l'environnement dans l'objectif d'accompagner la transition Scratch-Python. Nous étudions également de façon complémentaire la perception globale de l'application par les élèves.

Nous commençons par présenter les choix généraux effectués pour notre ressource (section 5.1), notre revue de littérature portant sur les serious games d'apprentissage de la programmation et sur les environnements de développements conçus pour faciliter la transition blocs-texte (section 5.2), puis le cadre théorique que nous allons mobiliser autour de la Théorie des situations didactiques (section 5.3). Suite à quoi, nous détaillons nos questions de recherche (section 5.4) et la méthodologie mise en œuvre pour y répondre (section 5.5). Ensuite, nous exposons le fonctionnement général de l'application Pyrates (section 5.6), sa mise en œuvre technique (section 5.7), la conception didactique et l'évaluation des huit niveaux du jeu (section 5.8), la conception et l'évaluation de l'environnement pour faciliter la transition Scratch-Python (section 5.9), puis la perception globale de l'application par les élèves (section 5.10). Nous terminons en évoquant la diffusion de l'application dans les classes (section 5.11).

5.1 Choix généraux

Nous présentons ici les choix généraux que nous avons fait concernant notre ressource d'enseignement en nous appuyant sur les résultats issus de nos analyses préalables.

Pour commencer, nous ciblons les élèves de seconde en début d'année scolaire dans les cours de mathématiques ou de SNT (voir section 4.5.1). L'objectif est de leur proposer une première approche de la programmation en Python et de viser de manière prioritaire les concepts qu'ils ont en principe déjà abordés au collège dans la

modalité Scratch (variable, conditionnelle, boucles for et while). Ceci dans le but d'accompagner le changement de registres sémiotiques (passage des blocs au texte), qui, comme nous l'avons vu dans la section 3.3, n'est pas immédiat pour tous les concepts.

Comme nous l'avons indiqué dans le chapitre précédent (voir section 4.5.3), nous avons décidé de donner à cette ressource la forme d'une application en ligne librement accessible depuis un navigateur Internet. Cela doit permettre de faciliter sa mise en œuvre dans les classes et de favoriser sa large diffusion.

Une autre caractéristique de la transition collège-lycée est le passage d'activités ludiques et concrètes au collège à des activités plus scolaires et abstraites au lycée en mesure d'amoindrir l'engagement et la motivation des élèves (voir section 2.5.5). Afin d'atténuer cet écart, nous avons choisi de mettre en œuvre un *serious game* dans l'application. Dans le contexte d'une application informatique, ce type de jeu est défini par Djaouti et ses collègues (2011) comme un logiciel qui associe un objectif non ludique à une structure de jeu vidéo : « any piece of software that merges a non-entertaining purpose (serious) with a video game structure (game) » (2011, p. 119). La recherche a montré l'impact positif de la ludification et en particulier des serious games sur l'enseignement de la programmation informatique (Lindberg et al., 2019; Vahldick et al., 2014; Yoon & Kim, 2015). Une méta-analyse réalisée par Zhan et ses collègues (2022) indique que ces effets portent surtout sur la motivation des élèves et la réussite scolaire (academic achievement). Ces auteurs montrent également que, dans le cadre d'un serious game, les bénéfices sont plus importants pour la programmation textuelle que pour la programmation par blocs. Ils avancent également que parmi les différents types de jeux (role-playing game, reasoning strategy game, puzzle game, robot game), les jeux de rôle se révèlent particulièrement efficaces pour améliorer les résultats scolaires des élèves. Même si ces résultats généraux concernant le numérique éducatif et en particulier les serious games doivent être nuancés (Amadieu & Tricot, 2014), nous avons pris le parti de mettre en œuvre un jeu de rôle, et en particulier un jeu de plateforme de type aventure.

Nous avons rappelé dans la section 4.5.4 que les textes officiels recommandent de favoriser l'écriture de programmes complets par contraste avec les types de tâches consistant à expliquer, corriger, ou compléter des programmes existants. Ainsi, dans notre jeu, le personnage est contrôlé à l'aide d'un programme écrit en Python. Les joueurs sont donc amenés à rédiger des programmes en entier afin d'atteindre les différents objectifs fixés selon les niveaux.

Enfin, la transition de Scratch à Python implique le passage du paradigme de programmation orientée objet au paradigme de programmation procédurale (voir section 3.3.1). En permettant aux joueurs de diriger leur personnage à l'aide de fonctions de contrôle à la manière de Scratch, nous leur donnons la possibilité de continuer à programmer des objets, tout en se confrontant aux autres aspects de la transition (composition des programmes, erreurs, exécution, etc.).

Pour résumer, notre ressource d'enseignement vise donc principalement les concepts de programmation introduits au collège afin d'accompagner le changement de registre sémiotique. Elle prend la forme d'une application en ligne que nous avons

nommée *Pyrates* (*Page d'accueil Pyrates*, 2023). Elle consiste en un jeu de plateforme dans lequel le joueur contrôle un avatar à l'aide d'un programme Python.

5.2 Revue de littérature

Des travaux ont déjà été menés au sujet des serious games visant l'apprentissage de Python ainsi que sur les environnements de développement facilitant le passage des blocs au texte. Voici un aperçu de ces solutions.

5.2.1 Serious games d'apprentissage de la programmation

Nous présentons d'abord des logiciels ludiques qui ciblent la programmation de façon générale, puis, dans un deuxième temps, un ensemble de serious games d'apprentissage du langage Python.

Serious games visant la programmation

La littérature propose plusieurs méta-analyses portant sur les jeux sérieux visant l'enseignement de la programmation informatique (Miljanovic & Bradbury, 2018; Vahldick et al., 2014), ou plus largement sur la ludification au service de l'apprentissage des fondements de la programmation (Shahid et al., 2019). Il ressort de ces études que la grande majorité des serious games disponibles ciblent d'abord l'enseignement supérieur (18 ans et plus), ou les enfants à l'école primaire et au début du collège (5-13 ans). Peu de jeux sont destinés aux élèves à la fin de collège et au début du lycée (14-17 ans). Notre contribution qui vise la transition collège-lycée doit participer à combler ce manque. De plus, les jeux ne sont pas toujours en accès libre sur Internet ce qui est problématique tant pour les chercheurs qui les étudient que pour les enseignants à la recherche de ressources éducatives. Notre logiciel s'exécute dans un navigateur Internet sans installation supplémentaire et est à disposition gratuitement sur Internet ce qui devrait faciliter sa diffusion. À propos des langages ciblés par les jeux, on retrouve majoritairement les langages de blocs puis les langages industriels courants : Java, C++, Javascript et Ruby. Ainsi, Python est faiblement représenté parmi les langages proposés dans les logiciels existants. Au regard de l'adoption croissante du Python comme langage d'introduction à la programmation textuelle (Sobral, 2021), Pyrates se positionne comme une ressource d'enseignement à même de soutenir cette montée en puissance.

Afin d'illustrer ces méta-analyses, nous présentons quelques jeux sérieux représentatifs. Par exemple, *PROG&PLAY* (Muratet, 2010) est un jeu de stratégie permettant de programmer différentes entités dans plusieurs langages de programmation (C, Java, Python, Scratch, OCaml). Ce jeu améliore la motivation des étudiants, en particulier dans des dispositifs de type atelier ou projet dans lesquels ils ont le temps de découvrir l'univers ainsi que les règles du jeu (Muratet et al., 2012). Djelil (2016) propose le jeu *PrOgO* basé sur la construction et l'animation en 3D. Il vise l'introduction de la programmation orientée objet dans l'enseignement supérieur. L'évaluation de l'application montre qu'elle est en mesure d'aider les étudiants à acquérir de nouvelles connaissances dans ce domaine. Enfin, *Ruby Warrior* est un jeu interactif conçu pour enseigner le langage de programmation Ruby ainsi que certains concepts d'intelligence artificielle. Une étude montre que son utilisation augmente la

motivation des élèves et leur intérêt pour la programmation (Watson & Richter Lipford, 2019).

Serious games d'apprentissage de Python

D'autres logiciels visent particulièrement l'apprentissage du langage Python. Nous analysons certains d'entre eux qui ont été sélectionnés en fonction de deux critères : i) le serious game est un jeu de rôle ; ii) les personnages du jeu sont contrôlables par des programmes Python.

Ainsi, Reeborg's World (*Reeborg's World home page*, 2023) est un jeu sérieux développé par un physicien canadien dans un objectif d'initiation à la programmation en Python. Son interface utilisateur est présentée dans la Figure 5-1-a.

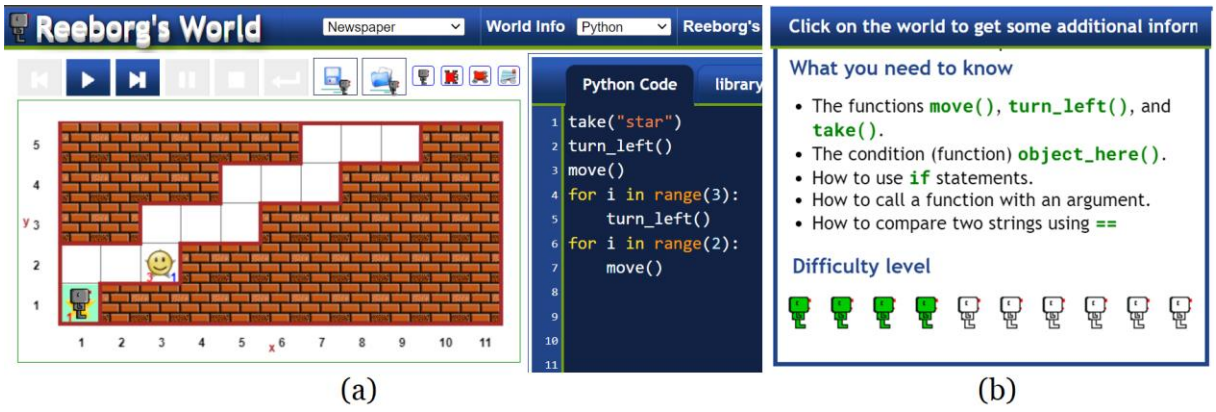


Figure 5-1 – Reeborg's World : interface utilisateur (a) et information sur le niveau courant (b)

Algorea (Léonard et al., 2022) est le jeu support d'un concours national de programmation organisé en France, ses fonctionnalités et son univers sont proches de ceux proposés par *Reeborg's World*.

D'autres serious games sont des applications commerciales qui ciblent les adolescents. Citons par exemple *Code Combat* (Kroustalli & Xinogalos, 2021) dont l'interface est visible dans la Figure 5-2-a.

Figure 5-2 – Code Combat : interface utilisateur (a) et présentation du niveau courant (b)

Code Monkey (*CodeMonkey home page*, 2023) et *Coding Park* (*CodingPark home page*, 2023) dont l'interface est reproduite dans la Figure 5-3-a sont des applications également payantes mais destinées à des enfants plus jeunes.

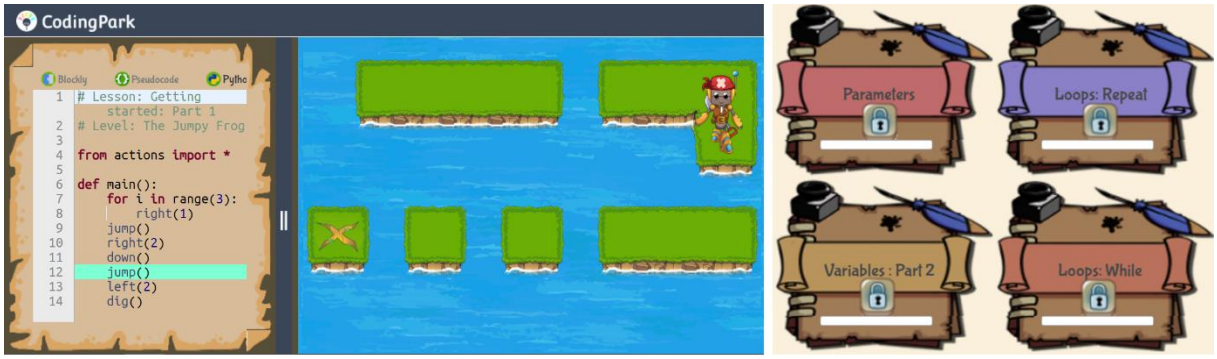


Figure 5-3 – Coding Park : interface utilisateur (a) et présentation des différentes quêtes (b)

Dans tous ces jeux sérieux, les concepts impliqués dans les différents niveaux sont clairement explicités. Par exemple, *Reeborg's World* propose un bouton « World info » qui donne plusieurs informations telles que : l'objectif du « monde », le niveau de difficulté et, dans la partie « What you need to know », les fonctions de contrôle à utiliser et les concepts à mettre en œuvre (voir Figure 5-1-b). Dans *Code Combat*, avant de commencer une nouvelle étape du jeu, les concepts à appliquer sont listés après l'objectif du niveau donné (voir Figure 5-2-b). Enfin, *Coding Park* a nommé ses différentes quêtes par le nom du concept qu'elles conduisent à découvrir (voir Figure 5-3-b). Il en va de même pour *Algorea* et *Code Monkey* qui font apparaitre explicitement dans les objectifs de chaque niveau une liste de concepts à mettre en œuvre. Nous prenons le parti de proposer dans Pyrates une approche didactique différente. Ainsi, notre objectif est également de mettre en jeu des concepts de programmation dans les différents niveaux du jeu, mais ces concepts ne seront pas explicités. Ils seront rendus nécessaires par la situation ludique proposée dans chaque niveau, ceci dans le but de leur donner plus de sens. Les élèves auront ainsi accès à tout moment à tous les concepts du langage Python sans indications particulières, et c'est uniquement les contraintes ludiques proposées dans chaque niveau qui devront les amener à implémenter les concepts visés afin de parvenir à leurs fins (voir la notion de situation adidactique définie dans la section 5.3).

5.2.2 Environnements pour la transition blocs-texte

Nous présentons dans cette section une vue d'ensemble des environnements de développement spécifiquement conçus pour soutenir la transition entre les langages de blocs et les langages textuels. Conformément à la taxonomie proposée par Lin et Weintrop (2021), nous en distinguons trois types : composés unidirectionnels (one-way transition), composés bidirectionnels (dual modality) et hybrides (hybrid).

Environnements composés unidirectionnels

Les environnements composés unidirectionnels comportent deux vues. Les programmes sont édités en assemblant des blocs dans la première vue, et sont ensuite automatiquement traduits dans un langage textuel cible dans la deuxième vue. Les utilisateurs peuvent uniquement consulter le programme traduit et, dans certains cas, l'exécuter, mais il ne peut pas être modifié directement. À titre d'exemple, l'environnement *EduBlocks* (*Edublocks home page*, 2023), visible dans la Figure 5-4,

convertit instantanément les programmes édités à l'aide de blocs en scripts Python. *Patch* (Robinson, 2016) offre une solution comparable basée sur Scratch.

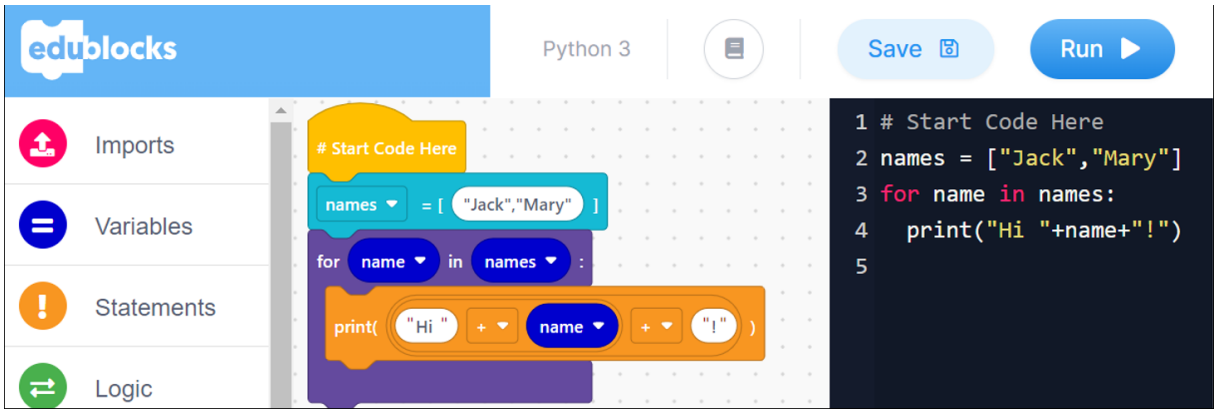


Figure 5-4 – Un exemple d'environnement composé unidirectionnel : Edublocks

Environnements composés bidirectionnels

Les environnements composés bidirectionnels sont également constitués de deux vues. Mais ici la vue textuelle permet également de créer et de modifier les programmes. Cela entraîne automatiquement la traduction ou la mise à jour du programme dans la vue blocs. Parmi les exemples d'implémentations existants, on peut citer *PencilCode* (Bau et al., 2015), qui gère le Javascript et plus récemment le Python (Andrews et al., 2021). Pour Java, *BlocEditor* (Matsuzawa et al., 2015) fournit un service analogue. *BlockPy* (Bart et al., 2017), dont l'interface est reproduite dans la Figure 5-5, offre un autre environnement composé bidirectionnel spécialement conçu pour Python.

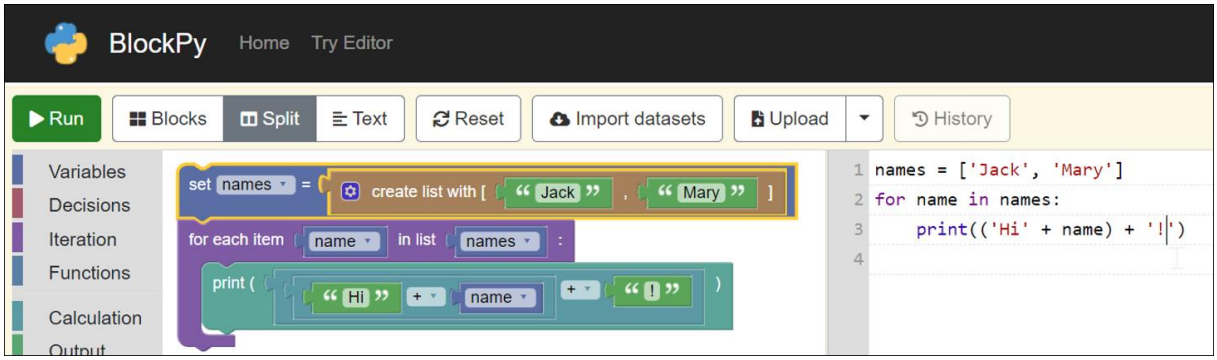


Figure 5-5 – Un exemple d'environnement composé bidirectionnel : BlockPy

Environnement hybrides

Enfin, les environnements hybrides combinent les blocs et le texte dans une unique vue. Dans ces environnements, le glisser-déposer, le pointer-cliquer ou les raccourcis clavier sont utilisés pour insérer les structures de haut niveau (telles que des boucles et des conditionnelles). L'édition de texte traditionnel soutenue par la complétion automatique permet de saisir le code de niveau expression. *Stride* offre aux apprenants une implémentation opérationnelle pour Java (Kölling et al., 2015). *Strype* (Kyfonidis et al., 2021; Weill-Tessier et al., 2022), présenté dans la Figure 5-6, fournit un environnement basé sur des cadres (framebased) permettant d'éditer du Python. Outre les caractéristiques décrites ci-dessus, il rend possible la manipulation de

groupes de lignes comme s'il s'agissait de blocs. *CodeStruct* (Kazemitabaar et al., 2022) est un autre environnement hybride récent qui a la particularité de fournir du contenu explicatif au sujet des concepts de programmation et des exemples exécutables dans des infobulles contextuelles.

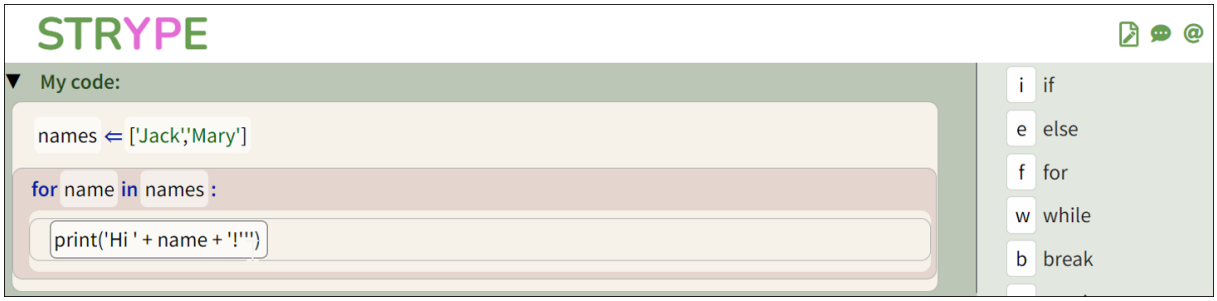


Figure 5-6 – Un exemple d'environnement hybride : Strype

Au-delà de cette taxonomie, il est possible de classer ces environnements dans deux catégories. Ceux basés sur la traduction (composés unidirectionnels et bidirectionnels) dont l'intérêt est de soutenir la transition bloc-texte sur les aspects intrinsèques des langages (paradigme de programmation, concepts implémentables et registres sémiotiques). Ainsi, la traduction instantanée du code d'un registre sémiotique à l'autre peut permettre aux étudiants d'assimiler la syntaxe des langages textuels en observant (unidirectionnels) ou en éditant (bidirectionnels) le code généré. Les autres éléments (paradigmes et concepts) étant, par construction, constants. La deuxième catégorie d'environnements est celle basée sur la fusion des modalités (hybride). Dans ce cas, l'accompagnement porte sur certaines différences liées aux environnements d'édition (catalogue de commandes et composition des programmes). Ainsi, ces applications hybrides sont en mesure de fournir des environnements de programmation textuels tout en continuant de bénéficier de certains des avantages des environnements de blocs (présence d'un catalogue de commandes, composition par assemblage). Lors de la conception de l'application Pyrates, notre objectif est de soutenir la transition sur les deux aspects à la fois : les propriétés intrinsèques des langages et les environnements d'édition des programmes.

Après avoir présenté notre revue de travaux, nous exposons dans la section suivante les différents concepts qui constituent notre cadre théorique.

5.3 Cadre théorique : la théorie des situations didactiques

Les travaux présentés dans ce chapitre s'appuient sur la Théorie des Situations Didactiques (TSD) élaborée par Brousseau (1998). Nous exposons ci-dessous quelques concepts fondamentaux de cette théorie qui seront mobilisés dans ce chapitre.

D'abord, Brousseau définit une *situation* comme : « une situation problème qui nécessite une adaptation, une réponse de l'élève » (1981, p. 112). Il établit le *milieu didactique* comme étant « constitué des objets (physique, culturels, sociaux, humains) avec lesquels le sujet interagit dans une situation [...]. C'est le système antagoniste de l'actant, [...] tout ce qui agit sur l'élève et ce sur quoi l'élève agit » (2010, p. 2).

Ensuite, selon Brousseau, les activités proposées aux élèves doivent tendre vers « une sorte d'idéal vers lequel il s'agit de converger » : la *situation adidactique* (1986a, p. 50). Ce modèle de situations a pour objectif de « provoquer chez l'élève les adaptations souhaitées, par un choix judicieux des "problèmes" [posés] » (1986, p. 49). Le professeur doit ainsi se demander : « À quel jeu[30] le sujet doit-il jouer pour avoir besoin de telle connaissance ? Quelle aventure - succession de jeux - peut l'amener à la concevoir, ou à l'adopter ? » (Brousseau, 2012, p. 105). Il s'agit ensuite de faire la *dévolution* de cette situation à l'élève, c'est-à-dire de lui déléguer la responsabilité de la construction de son savoir. Le professeur cherche alors à ce que l'« action de l'élève ne soit produite et justifiée que par les nécessités du milieu et par ses connaissances, et non par l'interprétation des procédés didactiques du professeur » (Brousseau, 2010, p. 5). De plus, « le maître se refuse à intervenir comme "proposeur" de connaissances qu'il veut voir apparaître [...] cette connaissance est entièrement justifiée par la logique interne de la situation » (Brousseau, 1986a, p. 49). Pour résumer, une situation didactique permet à l'élève s'il résout la tâche proposée d'accéder à une nouvelle connaissance, sans que sa réponse ne relève d'une injonction didactique. L'élève agit pour des raisons strictement de l'ordre du savoir, et pas pour faire plaisir à son enseignant, ni pour lui obéir (Dorier, 2010). Précisons que dans cette thèse nous nous limitons aux situations adidactiques d'*action* caractérisées par Bessot (2003) comme des situations dans lesquelles « un sujet (un élève) élabore des connaissances implicites comme moyen d'action sur un milieu : ce milieu lui apporte informations et rétroactions en retour de ses actions » (2003, p. 15). Bessot propose également un ensemble de conditions opérationnelles assurant qu'une situation « puisse être vécue comme adidactique » par un apprenant, nous en retenons deux (2003, p. 8) :

- (C1) l'élève doit pouvoir envisager une réponse initiale au problème posé sous la forme d'une *procédure de base* peu efficace s'appuyant sur ses connaissances antérieures ;
- (C2) le savoir ciblé doit permettre, lorsqu'il est mis en œuvre, de passer à une *procédure gagnante* résolvant le problème posé ;

Dorier (2010) distingue les notions de procédure et de stratégie. Ainsi une *procédure* se situe au niveau des observables et comporte donc de la contingence, c'est ce que fait un « vrai élève » avec ses contradictions et ses hésitations lorsqu'il est confronté à une situation. Une *stratégie* est imaginée par le concepteur d'une situation au niveau d'un élève théorique et générique.

Enfin, un chercheur qui conçoit une situation adidactique se doit de déterminer les *variables didactiques* de cette situation dont il fixe ensuite les valeurs adéquates afin de contrôler les stratégies et les savoirs en jeu. La définition que donne Brousseau (1982) de ces variables est la suivante:

> Un champ de problèmes peut être engendré à partir d'une situation par la modification de certaines variables qui, à leur tour, font changer les caractéristiques des stratégies de solution (coût, validité, complexité, etc.) [...]. Seules les modifications qui affectent la hiérarchie des stratégies sont à considérer comme des variables pertinentes et parmi

[30] Le mot « jeu » n'est pas ici à comprendre au sens ludique, c'est une façon alternative de désigner une situation, Brousseau ayant été fortement influencé par la théorie des jeux (Fudenberg & Tirole, 1991) lors de la constitution de la TSD (Brousseau, 2002)

> les variables pertinentes, celles que peut manipuler un professeur sont particulièrement intéressante : ce sont les variables didactiques. (Brousseau, 1982)

Ainsi ces variables, lorsqu'elles sont affectées de manière judicieuse, permettent de contrôler la « stratégie optimale » de la situation et par là même le savoir visé et la signification que l'on souhaite lui donner (Bessot, 2003). Ce qui fait dire à Dorier : « ce n'est que par ce choix de valeur de la variable [didactique] que l'on s'assure d'un milieu suffisamment antagoniste pour que la stratégie gagnante soit à coup sûr porteuse de la connaissance visée » (Dorier, 2010, p. 4).

Après avoir présenté les notions de notre cadre théorique, nous énonçons les questions de recherche qui nous occupent dans ce chapitre.

5.4 Questions de recherche

Dans ce chapitre, nous étudions la question de recherche Q5 : Comment concevoir une ressource d'enseignement de la programmation informatique sous la forme d'une application en ligne basée sur un jeu sérieux ? De quelle manière les élèves s'emparent-ils de cette application sur le terrain ?

Nous la déclinons selon les sous-questions énoncées ci-dessous :

- Q5.1 : Comment concevoir les niveaux du jeu Pyrates sur le modèle des situations adidactiques en ciblant les concepts fondamentaux de la programmation étudiés au collège ?
- Q5.2 : Quel est le niveau d'adidacticité effectif des situations conçues lorsqu'elles sont soumises à des élèves ?
- Q5.3 : De quelles fonctionnalités doter l'environnement de l'application Pyrates afin de soutenir la transition Scratch-Python ?
- Q5.4 : Sur le terrain, comment les élèves interagissent-ils avec les fonctionnalités de l'environnement ? Quel usage en font-ils ? Comment les jugent-ils en termes de clarté et d'utilité ?
- Q5.5 : Quelle est la perception globale de l'application par les élèves concernant la prise en main, la difficulté, l'aspect ludique et la motivation qu'elle engendre ?

Nous exposons dans la section suivante le détail de la méthodologie utilisée pour répondre à ces cinq sous-questions.

5.5 Méthodologie : conception et évaluation

La méthodologie mise en œuvre dans ce chapitre s'insère dans la méthodologie générale de la thèse. Ainsi, après avoir présenté la première étape des analyses préalables de l'ingénierie didactique dans les quatre premiers chapitres (analyse épistémologique, analyse curriculaire, analyse du champ de contrainte, et analyse des enseignements usuels), nous faisons état dans ce chapitre de la conception puis de l'évaluation de notre réalisation didactique. Nous détaillons donc dans les sous-sections suivantes la méthodologie de conception puis d'évaluation du logiciel Pyrates.

5.5.1 Conception : gameplay, analyse à priori, et fonctionnalités de l'environnement

La conception de notre application comporte deux versants, la constitution adidactique des niveaux du jeu sérieux sur le modèle des situations adidactiques (Q5.1) et l'aménagement de l'environnement afin d'accompagner la transition Scratch-Python (Q5.3).

Création des niveaux

Nous concevons les différents niveaux du jeu Pyrates sur le modèle des situations adidactiques. Il s'agit donc, dans notre cas, de mettre en jeu les concepts fondamentaux de la programmation dans des situations ludiques. Rappelons que seule la logique interne du jeu doit amener les joueurs à utiliser ces concepts qui ne seront pas explicités, mais qui devront être nécessairement implémentés dans les procédures gagnantes.

La première étape de cette conception consiste à choisir les concepts ciblés. Pour cela, nous nous appuyons sur les analyses des textes officiels réalisées dans la section 2.6.1 afin de couvrir les concepts qui ont été déjà introduits au collège.

Ensuite, nous choisissons un univers pour notre jeu de plateforme ainsi que les personnages qui pourront être contrôlés par les joueurs. Plusieurs études (Cassell, 2002; Yücel & Rızvanoğlu, 2019) suggèrent de choisir un thème neutre du point de vue du genre lors de la conception de serious games. Ceci a la vertu de dépasser les stéréotypes liés au genre : « my own design philosophy, which I call "underdetermined design" encourages both boys and girls to express aspects of self-identity that transcend stereotyped gender categories » (Cassell, 2002, p. 401). D'autres recherches conseillent de limiter les scènes de violence et de combats perçues comme un domaine de réussite masculine qui diminuent d'autant le niveau de perception de compétence et de confort des joueuses (Yücel & Rızvanoğlu, 2019), et qui sont globalement peu appréciées par les filles (Akre-Aas et al., 2022). Ensuite, l'incarnation et l'identification aux personnages sont des éléments importants dans la motivation et l'engagement dans les jeux, en particulier pour les filles (van Reijmersdal et al., 2013). Akre-Aas et ses collègues (2022) préconisent de mettre à disposition des joueurs des avatars réalistes auxquels ils peuvent s'identifier, en évitant la sexualisation des personnages féminins : « M1 : Realistic avatars in games, which the players can relate to or identify with, and M2 : Sexualisation of female characters should be avoided » (2022, p. 32).

Par la suite, afin d'établir les différentes situations ludiques, nous définissons également le *gameplay* de notre jeu. Ce gameplay est défini par Djaouti et ses collègues (2008) comme la combinaison de cinq éléments : les règles, les méthodes d'entrée, la configuration de l'espace, la configuration du temps et la configuration de la dramaturgie. Après avoir analysé plus de 600 jeux vidéo, ces auteurs (2008, 2011) ont pu mettre en évidence un ensemble de règles ludiques qui peuvent être combinées pour décrire ou concevoir le gameplay d'un jeu. Ces règles prennent la forme de dix « briques de gameplay » que nous reproduisons dans la Figure 5-7.

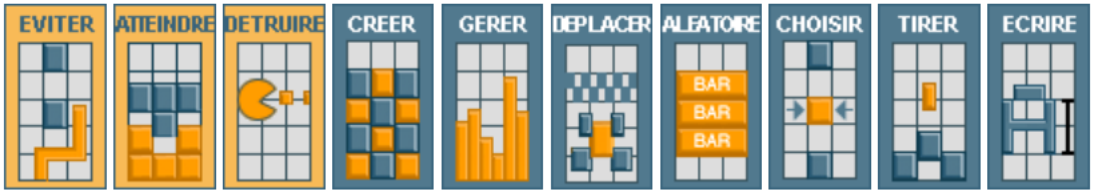


Figure 5-7 – Différentes briques de gameplay : objectifs du jeu et moyens mis à disposition des joueurs (Djaouti et al., 2011)

Les trois premières briques (oranges) permettent de caractériser les objectifs du jeu, les sept suivantes (bleues) décrivent les moyens mis à disposition des joueurs pour les atteindre. Nous ajoutons à ces dix modules la brique de moyen « collecter » qui avait été initialement introduite dans les travaux d'un des auteurs (Alvarez, 2007). Ces briques vont nous aider à concevoir les différentes situations ludiques qui constituent les niveaux de notre jeu.

Après quoi, afin de s'assurer que ces situations mettent bien en jeu les concepts visés de façon adidactique selon les deux conditions énoncées par Bessot (voir section 5.3), nous procédons à leur *analyse à priori*. Dorier (2010) présente cette analyse comme la réponse à une question :

> La question qui est au cœur de l'analyse *a priori* est : avec tout ce que mes élèves savent et ce qu'ils ont à disposition – le milieu – comment la question que je leur pose ou le problème que je leur soumets peut-il prendre sens – problème de dévolution - et que doivent-ils apprendre de nouveau pour arriver à le résoudre ? (Dorier, 2010, p. 2)

Artigue (1988) avance que cette méthodologie a pour but d'analyser les caractéristiques d'une situation adidactique en cours de conception ou d'évaluation. Ainsi, cette analyse comporte deux aspects, une partie descriptive et une partie prédictive. Il s'agit plus précisément de :

- motiver, pour chaque situation, les choix de conception, en particulier les valeurs données aux différentes variables didactiques et les savoirs en jeu ;
- imaginer et anticiper les différentes stratégies de résolution possibles et vérifier que celles comportant le meilleur coût résultent bien de la mise en œuvre de la connaissance ciblée.

Nous procédons donc à l'analyse à priori de chaque niveau du jeu en décrivant de façon méthodique pour chacun d'eux (voir annexe G) :

- les savoirs en jeu (parmi les concepts fondamentaux de la programmation) ;
- les valeurs données aux différentes variables didactiques ainsi que leur influence sur les procédures des élèves ;
- la listes des stratégies gagnantes, avec pour chacune d'elles, un nom, une description, la liste des concepts implémentés (utilisée pour la détection automatique de stratégies) et un exemple de programme.

Précisons que ce processus d'analyse à priori n'est pas linéaire et que son résultat relève de boucles de conceptions-analyses. De plus, comme le précise Dorier (2010), toutes les stratégies ne peuvent pas être anticipées en amont de l'expérimentation :

> L'analyse *a priori* vise à donner des explications rationnelles aux comportements des élèves en termes de choix et de stratégies. Dans ce sens, tout ne peut pas toujours être anticipé, c'est pourquoi une première réalisation avec de « vrais » élèves amène souvent à rectifier certains points. (Dorier, 2010, p. 2)

Aussi, comme nous le relatons plus loin, avant de passer à l'évaluation proprement dite de l'application dans les classes, nous avons procédé à plusieurs séances de bêta-test avec une classe. Ces séances nous ont conduit à remodeler certains niveaux. Ajoutons également que nous avons complété l'analyse à priori, en particulier l'inventaire des stratégies gagnantes, dans un second temps lors de l'analyse des données issues de l'expérimentation. Il s'agissait d'y ajouter les procédures mises en œuvre par les élèves que nous n'avions pas prévues. Margolinas (1992) explique à ce sujet que l'analyse à priori peut également être effectuée après l'expérimentation et qu'elle prend dans ces conditions une dimension plus explicative que prédictive :

> Dans l'analyse *a priori*, le terme de « *a priori* » ne signifie pas première temporellement. Il signifie que l'analyse *a priori* n'est pas dépendante des faits d'expérience [...]. L'analyse *a priori* peut donc avoir lieu après l'observation; elle perd son sens prédictif pour prendre un sens causal. (Margolinas, 1992, p. 132)

Aménagement de l'environnement pour la transition

Notre deuxième objectif de conception est d'implémenter dans l'environnement de Pyrates des fonctionnalités à même de soutenir la transition de Scratch vers Python. Nous nous appuyons pour cela sur les résultats établis dans le Chapitre 4 qui portent sur les différences intrinsèques entre ces langages, mais également entre les environnements d'édition des programmes. Pour ce faire, nous incorporons dans l'environnement de Pyrates certaines caractéristiques des langages de programmation par blocs en espérant pouvoir profiter de leurs avantages.

Suite à cette présentation de la méthodologie relative à la conception de notre logiciel, nous poursuivons par une description de la méthodologie d'évaluation.

5.5.2 Évaluation : terrain, traitement et analyse des données

Afin d'évaluer l'application conçue, nous avons procédé à une expérimentation dans plusieurs classes de seconde auprès d'élèves débutants en Python. Les données que nous avons recueillies sont à la fois quantitatives car issues des traces d'utilisations des élèves mais aussi qualitatives suite à la passation par les élèves d'un questionnaire en ligne en fin d'expérimentation. Enfin, nous avons traité ce corpus de données au moyen de programmes Python. Nous détaillons ces différentes étapes dans les lignes qui suivent.

Terrain

Notre évaluation s'appuie sur des expérimentations de terrain dans les classes. Nous avons donc commencé par chercher des enseignants susceptibles d'y participer avec leurs élèves. Lors de notre enquête auprès des enseignants, nous avions reçu 30 retours de personnes souhaitant obtenir les résultats de l'enquête ou désirant être informées de la création de ressources numériques (voir section 4.4.1). Nous les avons donc recontactées par courriel afin de leur proposer de participer à la phase de test de Pyrates (voir Annexe J.2). Au-delà des contraintes matérielles, les conditions à remplir

pour pouvoir participer à l'étude étaient d'enseigner en mathématiques ou SNT dans une classe de seconde dont les élèves n'ont pas encore été exposés au langage Python.

Suite à cette sollicitation, nous avons reçu quatre retours positifs d'enseignants situés en Bretagne[31]. Nous donnons les informations biographiques les concernant dans le Tableau 5-1 (les prénoms sont pseudonymisés).

Pseudo	Diplôme	Form. cont.	Concours.	Anc.	Disciplines
Yvan	Licence de maths	DIU EIL	CAPES Maths	17 ans	Maths, NSI
Mickael	Maitrise de maths pures	∅	Agreg. Maths	20 ans	Maths
Rémi	Ingénieur électronique	DIU EIL	CAPES Maths	7 ans	Maths, NSI, SNT
Carole	Maîtrise de maths pures	Hab. ISN	Agreg. Maths	27 ans	Maths, SNT

Tableau 5-1 – Données biographiques concernant les enseignants impliqués dans l'évaluation de Pyrates

Nous pouvons constater que ces enseignants ont une formation initiale principalement axée sur les mathématiques (à l'exception de Rémi) et qu'ils sont tous titulaires d'un concours d'enseignement dans la discipline mathématiques. Ils sont expérimentés voire très expérimentés. Bien qu'ayant une formation initiale en mathématiques, ces enseignants ont tous acquis des compétences informatiques au cours de leur carrière. Ainsi, Yvan et Rémi ont récemment suivi le DIU « Enseigner l'informatique au lycée » leur donnant le droit d'enseigner la discipline NSI, Carole est titulaire d'une habilitation à enseigner la spécialité ISN et Mickael développe de façon autodidacte des logiciels dans le cadre de projets personnels. Ajoutons qu'Yvan est également formateur INSPE en master MEEF NSI.

Ces enseignants travaillent dans trois lycées bretons que nous décrivons dans le Tableau 5-2.

Pseudo	Localisation	Type / taille	Nb élèves	Statut
Lycée E	Petite ville	Lycée polyvalent	1500	Public
Lycée D	Grande ville	Lycée général et technologique	2000	Public
Lycée S	Grande ville	Lycée général et technologique	1000	Privé sous contrat

Tableau 5-2 – Informations concernant les lycées impliqués dans l'évaluation de Pyrates

Nous retrouvons une certaine diversité dans ces lycées, avec un lycée public polyvalent situé dans une petite ville, un lycée public général et technologique localisé dans une grande ville, et un lycée privé sous contrat de centre-ville.

L'application a donc pu être testée dans neuf classes de seconde. Nous donnons le détail des séances dans le Tableau 5-3.

Enseignants	Lycée	Groupe / modalité	Séances / écart	Phase
Yvan, Chercheur	Lycée E	2nd J (33) / demi grp	2 x 1h / 15 jours	Bêta-test
Yvan, Chercheur	Lycée E	2nd H (31) / demi grp	2 x 1h / 15 jours	Expérimentation

[31] Pour des raisons pratiques de temps de transport, nous avons limité notre terrain d'expérimentation à la région Bretagne

Mickael, Chercheur	Lycée E	2nd B (28) / demi grp	2 x 1h / 15 jours	Expérimentation
Mickael, Chercheur	Lycée E	2nd E (30) / demi grp	2 x 1h / 15 jours	Expérimentation
Carole, Chercheur	Lycée D	2nd 10 (35) / demi grp	2 x 1h /15 jours	Expérimentation
Carole, Chercheur	Lycée D	2nd 12 (34) / demi grp	2 x 1h / 15 jours	Expérimentation
Carole, Chercheur	Lycée D	2nd 13 (30) / demi grp	2 x 1h / 15 jours	Expérimentation
Rémi, Chercheur	Lycée S	2nd 7 (16) / demi grp	2 x 1h / 7 jours	Expérimentation
Rémi, Chercheur	Lycée S	2nd 3 (36) / demi grp	3 x 1h / 7 jours	Expérimentation

Tableau 5-3 – Détails des séances d'évaluation de Pyrates

Les élèves ont ainsi pu utiliser Pyrates par demi-groupes pendant deux ou trois séances de 55 minutes espacées d'une ou deux semaines. Notons que suite à un problème technique, seul un demi-groupe de la 2nd 7 a pu utiliser l'application dans des conditions normales, les données issues de l'autre groupe ont donc été retirées du corpus. La modalité demi-groupe était une contrainte que nous avons imposée afin de pouvoir disposer d'un taux d'encadrement d'un enseignant (enseignant de la classe ou chercheur) pour sept à huit élèves.

Notons également que les 33 élèves de la classe de 2nd 7 ont été les premiers à utiliser l'application, les quatre premières séances ont servi de bêta-test pour le logiciel et la méthodologie dans la mesure où des dysfonctionnements ont pu être mis au jour et corrigés. Ainsi, suite à cette première phase nous avons : remodelé certains niveaux, modifié des contenus, ajouté un chronomètre pour les enseignants, et corrigé quelques problèmes techniques. Les traces d'utilisation générées dans cette phase ont donc été retirées du corpus. Toutes les autres séances ont donc été prises en compte dans l'expérimentation qui comporte donc un total de 240 élèves.

Caractérisation des élèves

Nous caractérisons ici ces 240 élèves concernant leur genre ainsi que leurs connaissances préalables au sujet de Scratch et Python.

D'abord, à la rentrée 2018, les classes de lycées généraux et technologiques comportaient 53,9% de filles (MEN, 2020a). Plus précisément, les classes de seconde générale et technologique françaises sont généralement constituées de façon équilibrées en terme de genre. Il est donc raisonnable de considérer que la parité est pratiquement respectée dans notre échantillon d'élèves.

Ensuite, nous faisons dans ce chapitre l'hypothèse implicite que les élèves disposent de connaissances préalables en programmation Scratch. Afin d'évaluer cette conjecture, nous leur avons demandé dans le questionnaire clôturant l'expérimentation d'estimer leurs connaissances générales en Scratch, puis plus précisément concernant les concepts de variable, conditionnelle, boucle for et boucle while : « Comment estimez-vous vos connaissances en Scratch ? Donner pour chaque élément une estimation allant de "Faibles" à "Importantes" en déplaçant le curseur. » (voir Annexe C.1). Les réponses à cette question sont compilées dans plusieurs courbes de répartition visibles sur la Figure 5-8.

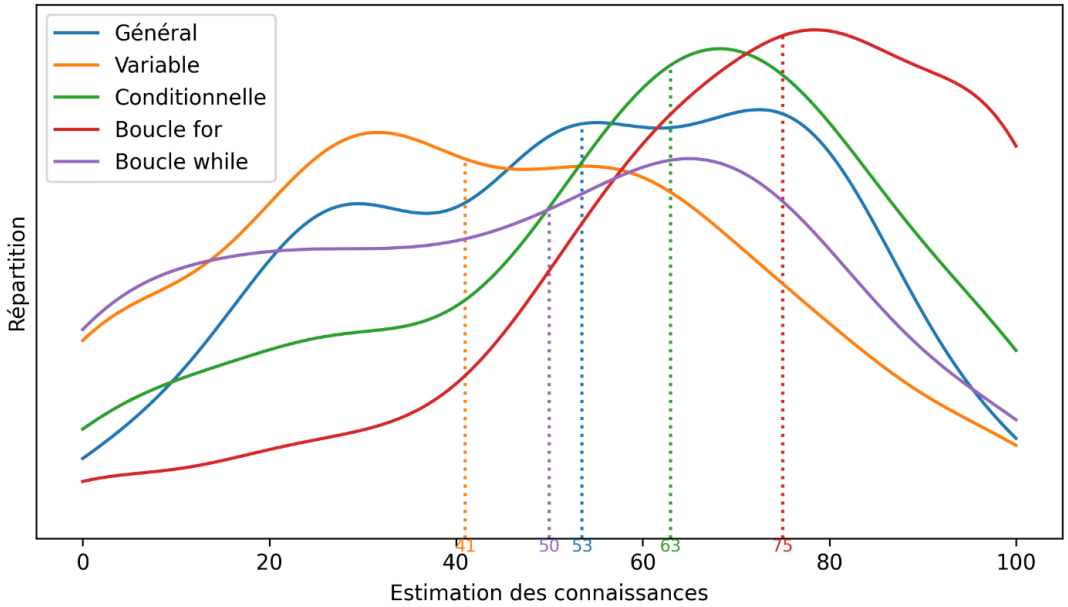


Figure 5-8 – Estimations déclarées des connaissances en Scratch (faibles à importantes) par concepts des élèves de l'étude (répartition et médiane)

Concernant la connaissance de « Scratch en général », les élèves déclarent des connaissances plutôt moyennes (53), même si nous pouvons voir trois sous-groupes se distinguer. Le concept de « boucle for (Répéter … fois) » est le mieux évalué (75), suivi par le concept de « conditionnelle (Si…alors…sinon) » (63), et celui de « boucle while (Répéter jusqu'à ce que…) » (50). Notons que le concept de variable est le moins bien valorisé avec un score médian de 41. À l'aune de ces éléments déclaratifs, nous pouvons considérer que les élèves de notre expérimentation disposent globalement de connaissances préalables en Scratch, même si le concept de variable semble moins bien intégré.

Enfin, les élèves sont supposés être débutants en Python. Afin de vérifier cette hypothèse, nous leur avons demandé dans l'enquête finale s'ils avaient déjà utilisé ce langage ou un autre langage textuel : « Avant de jouer à Pyrates, aviez-vous déjà utilisé le langage de programmation Python ou un langage similaire (sous la forme d'un texte) ? » (voir Annexe C.1). Les réponses des élèves sont regroupées dans la Figure 5-9 sous la forme d'un diagramme circulaire.

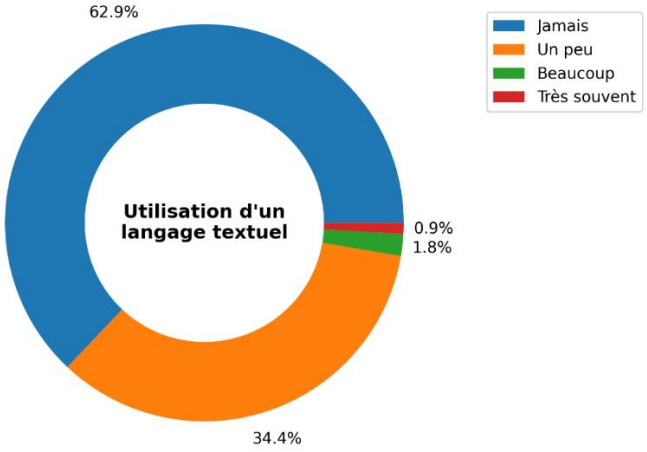


Figure 5-9 – Utilisation déclarée du langage de programmation Python ou similaire (langage textuel) avant l'expérimentation

Nous pouvons constater qu'un peu moins de deux tiers des élèves (62,9%) déclarent n'avoir jamais utilisé de langage textuel alors qu'un tiers d'entre eux (34,4%) affirment en avoir un peu utilisé. Enfin, une faible minorité d'élèves (2,7%) évoquent une expérience plus importante en programmation textuelle (beaucoup ou très souvent). Nous pouvons donc globalement estimer que les élèves de notre expérimentation sont débutants en Python.

Mode opératoire

Pour chaque demi-groupe, l'application et son fonctionnement ont été brièvement présentés par le chercheur pendant les cinq premières minutes de la première séance. Durant le reste de l'expérimentation, il était attendu des élèves qu'ils utilisent le jeu de manière autonome en évitant les communications entre pairs.

Les enseignants avaient pour consigne de n'intervenir qu'à la demande des élèves, ou s'ils repéraient un temps de jeu supérieur à 15 minutes dans un niveau, seuil à partir duquel nous considérons un élève comme étant en difficulté. Pour avoir accès à cette information, l'application dispose d'un mode recherche. Ce mode est accessible grâce à une adresse particulière, il est utilisé exclusivement lors des expérimentations que nous encadrons. Dans cette configuration, l'interface de Pyrates est enrichie d'une zone « Réservé enseignant » visible en bas à droite sur la Figure 5-10.

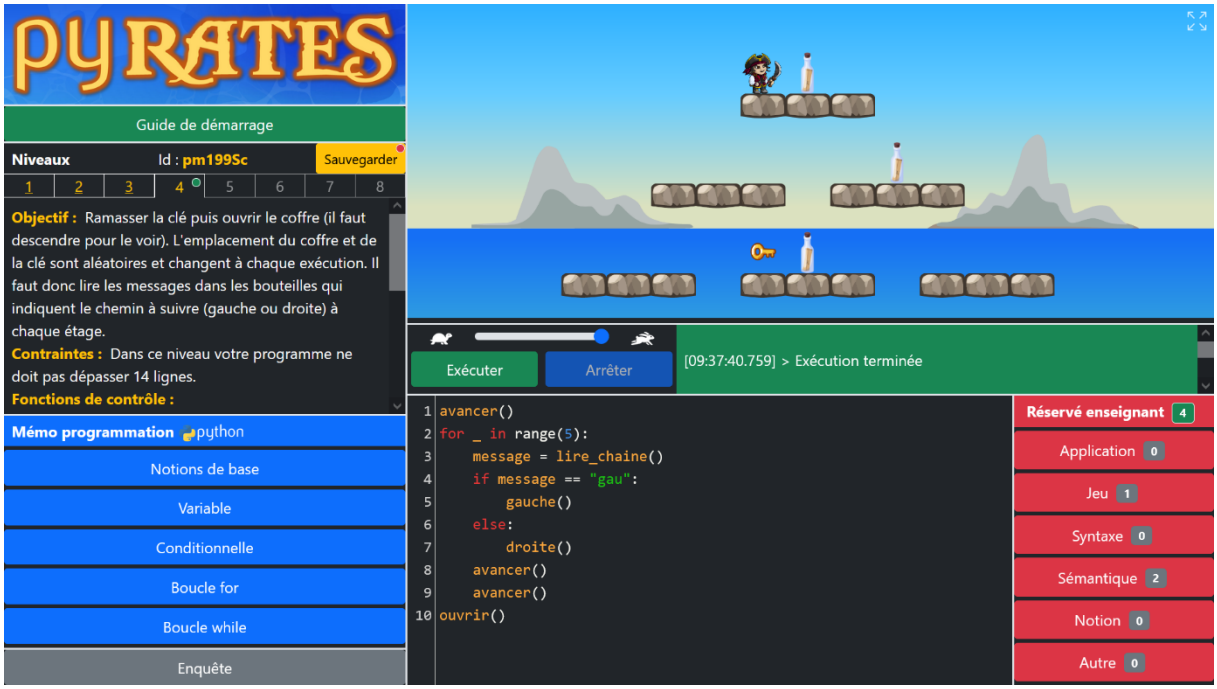


Figure 5-10 – Page principale de l'application Pyrates en mode recherche

Cette zone contient, entre autres choses, un chronomètre indiquant le temps en minutes passé par le joueur dans le niveau courant (pastille verte contenant le chiffre 4 dans l'exemple). La pastille associée au chronomètre change de couleur selon différents seuils afin d'aider l'enseignant dans sa prise de décision d'intervention : vert (0 à 10 minutes), orange (10 à 15 minutes) ou rouge (plus de 15 minutes). Ces seuils ont été établis en amont en lien avec le temps attribué à chaque niveau (voir section 5.8.1), puis affinés suite à la phase de bêta-test. Lors d'une intervention (demandée par l'élève ou proposée par l'enseignant) le rôle de l'enseignant consistait à apporter une aide aux élèves en fonction de son diagnostic. Il avait ensuite pour consigne de

renseigner le ou les sujets de l'aide fournie en cliquant sur les différents boutons prévus à cet effet dans la zone réservée : « Application », « Jeu », « Syntaxe », « Sémantique », « Notion » ou « Autre ». Ces différentes catégories d'aides sont issues d'un travail précédent (Branthôme, 2021) dans lequel nous avons élaboré une typologie les aides apportées par un enseignant lors de l'encadrement de tâches de programmation en Python destinées à des élèves débutants.

Le mode opératoire, et en particulier les différentes catégories d'aides, ont été expliqués aux enseignants en amont de l'expérimentation. Notons que le chercheur, présent lors de chaque séance, a prêté main forte à l'enseignant de la classe dans cette tâche en agissant selon le même schéma.

Traces d'utilisation et enquête

Durant les expérimentations que nous venons de décrire, nous avons récolté des données par l'intermédiaire des traces d'utilisations de l'application et via un questionnaire en ligne diffusé aux élèves.

Au cours de l'utilisation de l'application en mode recherche, les traces sont générées automatiquement en fonction de l'activité des élèves. Elles sont ensuite exportées au format standard xAPI puis stockées dans un Learning Ressources Store (LRS). xAPI est un format de données permettant de formaliser et de partager des données relatives aux activités humaines dans un environnement numérique (Kevan & Ryan, 2016). Il s'appuie sur le formalisme JSON. Chaque action unitaire est relatée à posteriori sur la forme d'un *statement* contenant les parties suivantes :

- actor : acteur de l'action ;
- verb : verbe au passé décrivant l'action entreprise par l'acteur ;
- object : objet de l'action ;
- result (optionnel) : résultat de l'action ;
- context (optionnel) : élément de contexte de l'action ;
- timestamp (optionnel) : date de début de l'action.

Un LRS permet d'enregistrer, de représenter, et dans une moindre mesure, d'analyser les données exportées au format xAPI. Le LRS utilisé est Learning Locker, il est hébergé sur les serveurs de Sorbonne Université à Paris.

Les traces collectées par l'application concernent six types d'activités. Le Tableau 5-4 donne le détail des informations contenues dans ces traces.

Code	Activité	Informations
CON_CON	Consultations de contenus	Identifiant du contenu, durée de consultation
COP_CON	Copie de contenus	Identifiant du contenu
LAN_PRO	Lancement de programmes	Code du programme, vitesse d'exécution, issue du programme (too-many-lines, syntactic-error, semantic-error, game-error, user-stopped, level-lost, level-completed, fully-executed), erreur (si survenue)
NAV_NIV	Navigation entre les niveaux	Commencé, terminé, recommencé, repris
AID_REC	Aides reçues de l'enseignant ou du chercheur	Type d'aide (application, game, syntax, semantics, notion, other)

MOD_VIT	Modification du curseur de vitesse exécution	Nouvelle valeur de la vitesse

Tableau 5-4 – Traces générées par l'application Pyrates en mode recherche en fonction de l'activité des élèves

Nous donnons dans l'Annexe F le détail complet des éléments tracés ainsi que des exemples de statements xAPI générés pour chacune de ces catégories[32].

Lors des cinq dernières minutes de la séance finale, les élèves ont été invités à répondre à une enquête en ligne accessible par un lien situé en bas à gauche de l'interface (voir Figure 5-10). L'objectif de cette enquête est de recueillir des informations complémentaires au sujet des élèves, notamment concernant les connaissances préalables et la perception globale de l'application. Les questions de l'enquête sont reproduites en intégralité en Annexe C. Afin de permettre une passation rapide, toutes les questions sont fermées et y répondre nécessite uniquement de positionner des curseurs entre deux extrémités (par exemple « faible » - « important »), ou de choisir parmi une liste de boutons radios. De façon similaire à la méthodologie utilisée pour l'enquête auprès des enseignants (voir section 4.3.7), le questionnaire a été mis en œuvre sur la plateforme LimeSurvey et les données sont hébergées sur les serveurs de l'INSPE de Bretagne.

Comme nous l'avons indiqué, toutes les données que nous récoltons dans le cadre de cette expérimentation sont hébergées en France. De plus, nous ne collectons aucune donnée personnelle permettant d'identifier directement les élèves (nom, prénom, genre, lycée, etc.). En effet, chaque partie est repérée au moyen d'un identifiant unique généré aléatoirement que les élèves peuvent noter. Cela leur permet d'enregistrer la partie en cours puis de la rependre ultérieurement (voir section 5.6.3).

Ce protocole de collecte respecte les règles éthiques concernant la protection des données personnelles dans la mesure où il ne rentre pas dans le cadre du Règlement Général sur la Protection des Données (RGPD). Afin de nous en assurer, nous avons contacté la CNIL au mois de juin 2021 pour présenter notre projet de collecte. Cet organisme nous a répondu qu'en cas d'*anonymisation* des données, le RGPD ne s'appliquait pas et il ne serait donc pas nécessaire d'obtenir le consentement des responsables légaux des élèves. Cependant, il nous est revenu de nous assurer que nous sommes bien dans le cadre d'une anonymisation (et non d'une *pseudonymisation*) en sollicitant l'avis du délégué à la protection des données (DPO) de l'organisation hébergeuse :

> Concernant l'anonymisation, il s'agit d'un traitement qui consiste à utiliser un ensemble de techniques de manière à rendre impossible, en pratique, toute identification de la personne par quelque moyen que ce soit et de manière irréversible. Dans ce cas, la législation relative à la protection des données ne s'applique plus, car la diffusion ou la réutilisation des données anonymisées n'a pas d'impact sur la vie privée des personnes concernées. Cependant, l'anonymisation ne doit pas être confondue avec la notion de pseudonymisation des données personnelles. La pseudonymisation est un traitement de données personnelles réalisé de manière à ce qu'on ne puisse plus attribuer les données relatives à une personne physique sans information

[32] Dans cette annexe, les éléments préfixés par {V2} ne doivent être pris en compte ici, ce sont des modifications utilisées pour les évolutions présentées dans le Chapitre 6.

> supplémentaire [...]. Il vous revient d'analyser le processus mis en place à travers votre application afin de déterminer s'il s'agit de données pseudonymisées ou anonymisées [...]. Je vous précise que le DPO a pour mission de veiller au respect de la protection des données au sein d'une organisation. (Extrait du courriel de réponse de la CNIL daté du 25 juin 2021)

Nous avons donc contacté le DPO de Sorbonne Université en juillet 2021 qui a considéré que nos données étaient anonymes dans la mesure où il n'existe pas de « table de concordance permettant de relier directement l'utilisateur de l'application et le code [aléatoire] généré ».

Enfin, en complément de ces traces d'activités récoltées auprès de 240 élèves lors des expérimentations que nous avons encadrées, nous utilisons de façon ponctuelle les données applicatives enregistrées dans le cadre de l'utilisation standard de l'application Pyrates par ses utilisateurs finaux. Ces données qui portent sur un nombre beaucoup plus important de personnes (75 000) permettent d'inférer des résultats plus robustes. En revanche, elles sont plus sommaires car elles ne contiennent que les informations sauvegardées dans la base de données applicative qui permettent de retrouver sa partie après avoir quitté le jeu (voir section 5.7.3). Précisons que ces informations sont également collectées hors du cadre du RGPD dans la mesure où elles ne concernent aucune donnée personnelle permettant d'identifier directement les utilisateurs et qu'elles sont anonymisées de la même manière que les traces d'utilisations du mode recherche (identifiant de partie). Parmi ces données applicatives, nous prenons en compte tous les programmes sauvegardés ayant permis de terminer les niveaux (procédures gagnantes) pour la période allant du 01/11/2021 au 25/03/2023.

Nous venons de décrire la collecte des données que nous allons ensuite traiter et analyser. Cette collecte a permis de constituer un corpus composé de trois jeux de données dont nous donnons le récapitulatif dans le Tableau 5-5

Code	Description	Cardinalité
EX_TA	Traces d'activités provenant des 240 élèves de notre expérimentation	69 701 traces d'activités
EX_RQ	Réponses au questionnaire en ligne proposé en fin d'expérimentation	224 réponses complètes[33]
UT_PG	Programmes constituant les procédures gagnantes exécutées par 75 059 utilisateurs finaux du 01/11/21 ou 25/03/23	261 767 programmes Python

Tableau 5-5 – Description des trois jeux de données constituant le corpus

Traitement et analyses des données

Notre traitement des données consiste d'abord à les exporter depuis les différentes sources (LRS, LimeSurvey et base de données applicative) au format Excel (.xlsx). Ces données ont ensuite été traitées au moyen de Script Python. Les bibliothèques *Pandas*, *NumPy* et *Ast* ont été utilisées pour traiter les données, et les bibliothèques *Matplotlib* et *Seaborn* ont permis de créer les graphiques présentés dans

[33] Suite à un problème technique de blocage d'adresse dans un établissement, un demi-groupe n'a pas pu répondre au questionnaire.

ce chapitre. Nous détaillons maintenant les traitements réalisés en fonction des questions de recherche.

D'abord, afin d'évaluer la conception des niveaux (Q5.2), nous nous appuyons sur les programmes exécutés par les utilisateurs. Plus précisément, nous nous intéressons aux concepts de programmation mis en œuvre dans le code des programmes en nous basant sur leur arbre de syntaxe abstraite (AST). Un AST est une représentation arborescente de la structure générale d'un programme. Pour illustrer, nous représentons dans la Figure 5-11 un programme simple issu du jeu contenant une boucle et l'AST qui lui est associé.

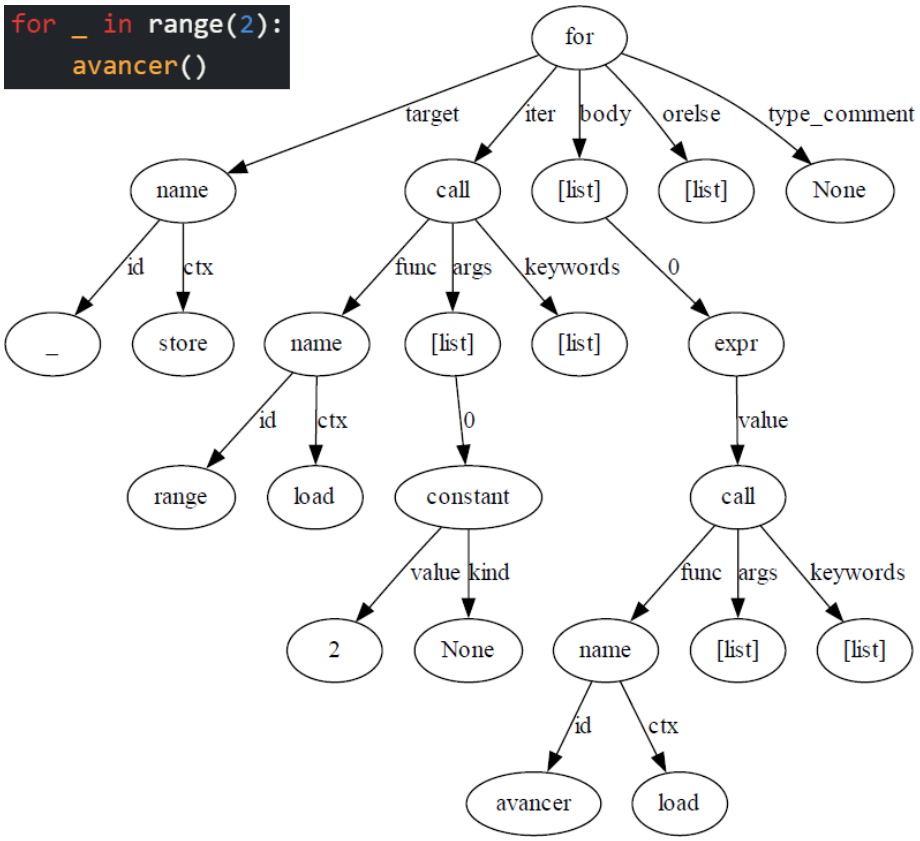


Figure 5-11 – Exemple d'un programme Python et de son AST

L'identification des concepts mis en œuvre dans un programme est réalisé par un parcours en profondeur de l'AST (racine puis parcours récursif des sous-arbres). Durant ce parcours, nous détectons les différents concepts de programmation à l'aide de conditions sur les nœuds. Nous détaillons dans le Tableau 5-6 les huit concepts recherchés ainsi que les critères permettant leur identification.

Code	Description	Condition de détection
for-re	Boucle for utilisée pour des répétitions sans usage de la variable de boucle	Présence d'un nœud « for » dans l'AST sans utilisation de la variable de boucle dans le sous-arbre « iter »
for-va-0	Boucle for avec utilisation de la variable de boucle qui commence à zéro (ex : « for i in range(4) »)	Présence d'un nœud « for » dans l'AST avec utilisation de la variable de boucle dans le sous-arbre « iter » et utilisation de la fonction range à un argument ou un deux arguments avec le premier valant « 0 »

for-va-n	Boucle for avec utilisation de la variable de boucle qui ne commence pas à zéro (ex : « for i in range(2,5) »)	Présence d'un nœud « for » dans l'AST avec utilisation de la variable de boucle dans le sous-arbre « iter » et utilisation de la fonction range à deux arguments avec le premier différent de « 0 »
while	Boucle while	Présence d'un nœud « while » dans l'AST
var-af	Affection d'une variable par le biais des instructions = ou +=	Présence d'un nœud « assign » ou « augAssign » dans l'AST
if	Utilisation de la branche if dans une conditionnelle	Présence d'un nœud « if » dans l'AST dont le parent n'est pas un nœud « if »
elif	Utilisation de la branche elif dans une conditionnelle	Présence d'un nœud « if » dans l'AST dont le parent est également un nœud « if »
else	Utilisation de la branche else dans une conditionnelle	Présence d'un sous-arbre « orelse » qui ne soit pas de type « if » dans un nœud « if »

Tableau 5-6 – Liste des concepts identifiés dans les programmes et critères de détection

Ces huit concepts que nous identifions dans le code des joueurs ont été choisis en cohérence avec les concepts visés lors de la conception des différents niveaux du jeu qui sont issus de notre analyse didactique (voir plus loin section 5.8.1).

Ayant à disposition les concepts implémentés dans les programmes, nous procédons ensuite à la détection automatique des stratégies pour les procédures gagnantes (programmes exécutés menant à la fin du niveau). Pour mettre en œuvre cette détection, nous nous appuyons sur notre analyse à priori. En effet, nous avons catalogué les stratégies gagnantes pour chaque niveau et nous disposons de la liste des concepts mis en œuvre pour chacune d'elle (voir Annexe G). La détection s'effectue donc en comparant la liste des concepts implémentés dans un programme donné avec les listes des concepts constituant les différentes stratégies de l'analyse à priori jusqu'à trouver une correspondance. Si aucune correspondance n'est trouvée, la procédure de l'élève est classée comme « Other ». Précisons que pour les procédures gagnantes issues de nos expérimentations dans les classes (jeu de données EX_TA), nous avons mis à jour l'analyse à priori en ajoutant de nouvelles stratégies jusqu'à ne plus avoir de procédures catégorisées « Other ».

La suite des traitements ainsi que les diagrammes générés afin d'évaluer la conception des niveaux (Q5.2) sont décrits dans le Tableau 5-7.

Jeu données	Données	Traitements	Type diagramme
EX_TA	LAN_PRO (code du programme)	Filtration des programmes: fully-executed, level-lost, user-stopped Détection des concepts (AST) Calcul des fréquences d'apparition des différents concepts dans les procédures non gagnantes exécutées par niveau	Barres juxtaposées Figure 5-21
EX_TA	LAN_PRO (code du programme)	Filtration des programmes: level-completed Détection des concepts (AST) puis des stratégies Calcul de la répartition des stratégies par niveau	Circulaire en anneau Figure 5-22

UT_PG	LAN_PRO (code du programme)	Filtration des programmes: level-completed Détection des concepts (AST) puis des stratégies Calcul de la répartition des stratégies par niveau	Circulaire en anneau Figure 5-23

Tableau 5-7 – Description des traitements et des représentations graphiques permettant de répondre à la question de recherche Q5.2

L'implémentation des différents concepts dans les procédures non gagnantes des élèves ainsi que la classification des différentes procédures gagnantes mises en œuvre nous permettent d'évaluer l'adidacticité des situations conçues selon les conditions proposées par Bessot.

Ensuite, afin d'évaluer la manière dont les élèves s'approprient et jugent les différentes fonctionnalités que nous avons intégrées à l'environnement de Pyrates afin de faciliter la transition Scratch-Python (Q5.4), nous nous appuyons sur les traitements et les diagrammes listés dans le Tableau 5-8. Nous représentons la répartition des valeurs numériques issues des différents curseurs du questionnaire en utilisant la méthode KDE décrite dans la méthodologie du Chapitre 4 (voir section 4.2.2).

Jeu données	Données	Traitements	Type diagramme
EX_TR	CON_CON (durée de consultation)	Calcul de la durée moyenne de consultations de chaque contenu par élève et par niveau	Barres juxtaposées Figure 5-27
EX_RQ	Réponses Q4	Calcul de la répartition des réponses concernant la clarté des explications du mémo (KDE)	Courbe répartition Figure 5-28-a
EX_RQ	Réponses Q5	Calcul de la répartition des réponses concernant l'utilité des comparaisons Scratch-Python (KDE)	Courbe répartition Figure 5-28-b
EX_TR	COP_CON (identifiant du contenu)	Calcul du nombre moyen de copies de chaque contenu par élève et par niveau	Barres juxtaposées Figure 5-29
EX_TR	LAN_PRO (issue du programme)	Calcul du nombre moyen de programmes comportant des erreurs syntaxiques (syntactic-error) et sémantiques (semantic-error) par élève et par niveau	Barres juxtaposées Figure 5-30-a
EX_TR	AID_REC (type d'aide)	Calcul du nombre moyen d'aides reçues par élève et par niveau concernant la syntaxe (syntax) ou la sémantique (semantics)	Barres juxtaposées Figure 5-30-b
EX_RQ	Réponses Q6	Calcul de la répartition des réponses concernant la clarté des messages d'erreur (KDE)	Courbe répartition Figure 5-31
EX_TR	LAN_PRO (issue du programme)	Calcul du nombre moyen de programmes lancés par élève et par niveau selon leur issue : - Erronés lancés : syntactic-error, semantic-error, too-many-lines	Barres juxtaposées Figure 5-32

		- Corrects lancés : fully-executed, game-error, level-completed, level-lost, user-stopped - Lancés arrêtés : user-stopped	
EX_TR	MOD_VIT	Calcul du nombre moyen de changements de vitesse d'exécution par élève et par niveau	Barres Figure 5-33-a
EX_TR	LAN_PRO (vitesse d'exécution)	Calcul de la répartition des vitesses d'exécution des programmes par niveau selon une estimation de la distribution (KDE)	Courbes répartition Figure 5-33-b

Tableau 5-8 – Description des traitements et des représentations graphiques permettant de répondre à la question de recherche Q5.4

L'étude des interactions des élèves avec les différentes fonctionnalités de l'environnement (mémo Python, panneau de commande, etc.) ainsi que les réponses au questionnaire portant sur la clarté et l'utilité de ces éléments nous fournissent des informations à même d'évaluer leur appropriation par les élèves.

Pour finir, nous estimons la perception globale de l'application par les élèves (Q5.5) en analysant les réponses au questionnaire en lien avec la prise en main, la difficulté, l'aspect ludique, et la motivation engendrée (voir Tableau 5-9).

Jeu données	Données	Traitements	Type diagramme
EX_RQ	Réponse Q8	Calcul de la répartition des réponses concernant la perception globale de l'application (KDE)	Courbes répartition Figure 5-34

Tableau 5-9 – Description des traitements et des représentations graphiques permettant de répondre à la question de recherche Q5.5

Dans cette section, nous avons détaillé la méthodologie mise en œuvre dans ce chapitre, nous présentons maintenant le fonctionnement global de notre logiciel.

5.6 Fonctionnement général de l'application Pyrates

Avant de préciser la conception des différents niveaux et l'aménagement de l'environnement pour la transition Scratch-Python, nous exposons ici les généralités concernant le fonctionnement de Pyrates.

5.6.1 Choix du personnage

En tapant l'adresse de l'application dans un navigateur Internet (*Page d'accueil Pyrates*, 2023), l'utilisateur accède à la page d'accueil reproduite dans la Figure 5-12. La première étape consiste à choisir un personnage parmi les deux disponibles (nous présentons les éléments relatifs aux choix du thème et des personnages dans la section 5.8.1).

Figure 5-12 – Page d'accueil de l'application Pyrates

5.6.2 Internationalisation

La page d'accueil propose également de choisir la langue de l'application via une liste déroulante (voir en haut à droite Figure 5-12). Au moment de la rédaction de cette thèse, seuls le français et l'anglais sont disponibles. L'ensemble des contenus consultables, des instructions et des programmes ont ainsi été traduits, et l'intégralité des messages sont externalisés dans un fichier ad-hoc. Il est ainsi possible de proposer de nouvelles langues sans développements supplémentaires.

Ces dernières années, au niveau international, le premier contact des apprenants avec la programmation a généralement lieu dans des environnements basés sur des blocs comme Scratch (Weintrop, 2019). Ensuite, il est courant que les élèves approfondissent leurs compétences en programmation en utilisant le langage Python (Sobral, 2021). Ainsi, la problématique de transition des blocs vers le texte que nous connaissons en France à la transition collège-lycée est largement partagée dans le reste du monde. Dans ce contexte, il est probable que l'application Pyrate puisse être employée dans des pays non francophones.

5.6.3 Sauvegarde

En suivant les recommandations établies dans le chapitre précédent (section 4.5.2), le jeu a été conçu pour une durée de deux séances de cours d'environ une heure chacune. Il est donc nécessaire de proposer aux joueurs l'enregistrement de leur partie afin de pouvoir la retrouver d'une séance à l'autre. Comme nous l'avons expliqué précédemment (section 5.5.2), nous ne pouvons pas sauvegarder de données nominatives relatives aux joueurs afin de respecter le RGPD. Ainsi, un clic sur le bouton « Je commence » sur la page d'accueil (voir Figure 5-12) provoque la génération d'un identifiant aléatoire unique et l'apparition d'une fenêtre contextuelle (voir Figure 5-13).

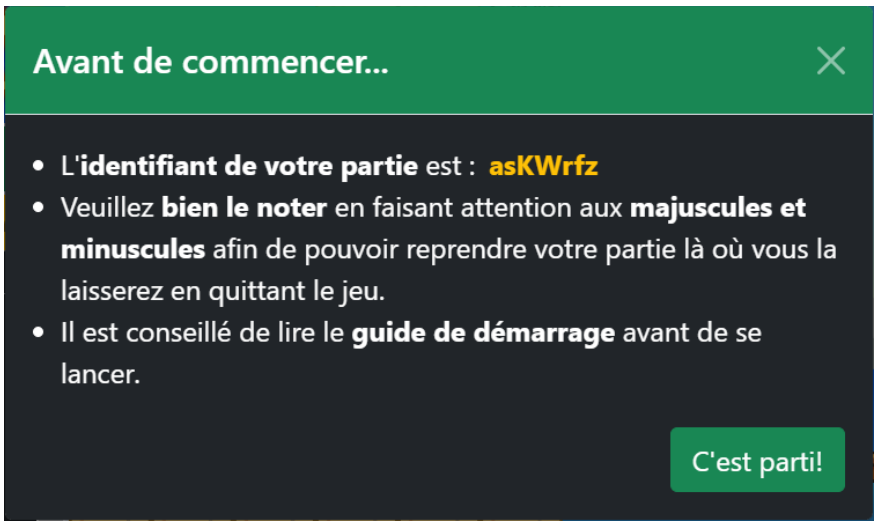


Figure 5-13 – Fenêtre contextuelle affichée lors du démarrage d'une partie

Cette fenêtre conseille aux joueurs de prendre note de leur identifiant de partie, et de consulter le guide de démarrage avant de commencer (voir plus loin).

Le lancement de la partie mène ensuite à la plage principale de l'application (voir Figure 5-14). On retrouve à nouveau dans le coin supérieur gauche de cette interface l'identifiant de la partie en cours ainsi qu'un bouton « Sauvegarder » permettant d'enregistrer l'état de la partie (le point rouge indique qu'une sauvegarde est possible). En cas de fermeture du navigateur web, la page d'accueil permet ensuite de reprendre une partie en saisissant l'identifiant dans un champ texte puis en cliquant sur le bouton « Je reprends ! » (voir partie basse sur la Figure 5-12).

5.6.4 Guide de démarrage

Lors de la première connexion à l'application, les utilisateurs sont invités à consulter le guide de démarrage reproduit intégralement en Annexe H. Ce guide, accessible depuis le bouton vert au haut à gauche de l'écran (voir Figure 5-14), explique brièvement l'objectif du jeu, le fonctionnement général de l'application, ainsi que le mécanisme de sauvegarde des parties.

5.6.5 But du jeu

Comme le montre la Figure 5-14, le serious game que nous avons développé est un jeu de plateforme permettant de contrôler un personnage à l'aide d'un programme écrit en Python. Ce personnage évolue dans un environnement en deux dimensions régi par la gravité et les collisions physiques. Du fait du pilotage par programme, les déplacements du personnage ne sont pas totalement libres mais s'appuient sur une grille de blocs partiellement visible qui transparait dans les éléments constituant les plateformes. Les déplacements horizontaux et vers le haut (sauts) se font donc en suivant cette grille de blocs. Toutefois ces déplacement sont limités par les collisions avec les plateformes et les objets du premier plan. Les déplacements vers le bas sont uniquement soumis à la gravité et aux collisions.

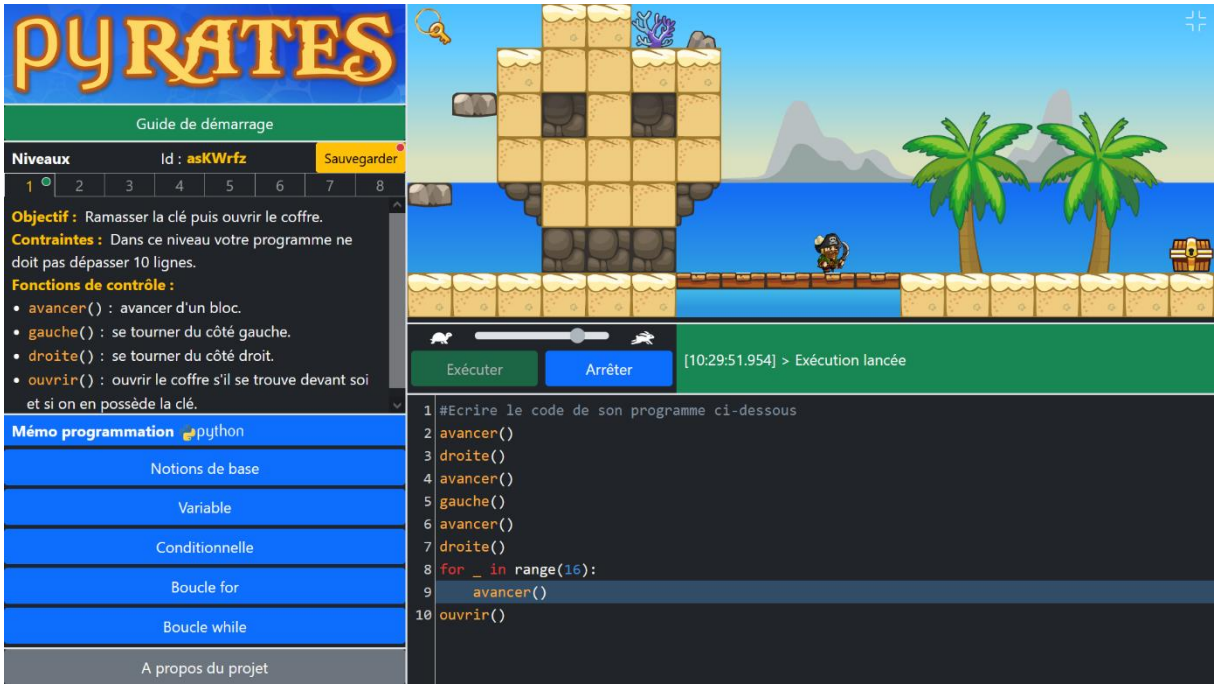


Figure 5-14 – Page principale de l'application Pyrates

Le jeu comprend huit niveaux ayant des objectifs bien établis basés sur la collecte d'une clé permettant d'ouvrir un coffre au trésor. Chaque niveau possède des *fonctions de contrôle* qui lui sont propres (avancer, ouvrir, etc.) et qui permettent de diriger le personnage dans sa quête. Ajoutons qu'il est nécessaire de terminer un niveau avant de passer au suivant. Il est cependant possible de naviguer entre les niveaux terminés en cliquant sur les numéros de niveau dans la zone dédiée (voir Figure 5-14 en haut à gauche de l'écran). Cela permet d'accéder à des niveaux antérieurs afin de consulter leur code ou de les recommencer.

5.6.6 Guide pédagogique

L'application Pyrates dispose d'un guide pédagogique en ligne destiné aux enseignants (*Guide pédagogique Pyrates*, 2023). Rédigé en français et en anglais, il présente l'application, donne des conseils de mise en œuvre dans les classes, et décrit chaque niveau à travers les concepts en jeu et une solution possible. Cette dernière partie constitue une version simplifiée de l'analyse à priori présentée dans la section 5.8.

Nous venons d'expliquer le fonctionnement général de l'application Pyrates, nous évoquons dans la section suivante les aspects techniques de son implémentation.

5.7 Mise en œuvre technique de l'application Pyrates

Nous exposons maintenant l'implémentation technique du logiciel Pyrates en commençant par son architecture générale avant de préciser les éléments relatifs au front-end (gestion de l'interface utilisateur) puis au back-end (accès aux données).

5.7.1 Architecture générale

La Figure 5-15 présente le fonctionnement général du logiciel Pyrates. Elle détaille ses différentes parties et les communications entre elles.

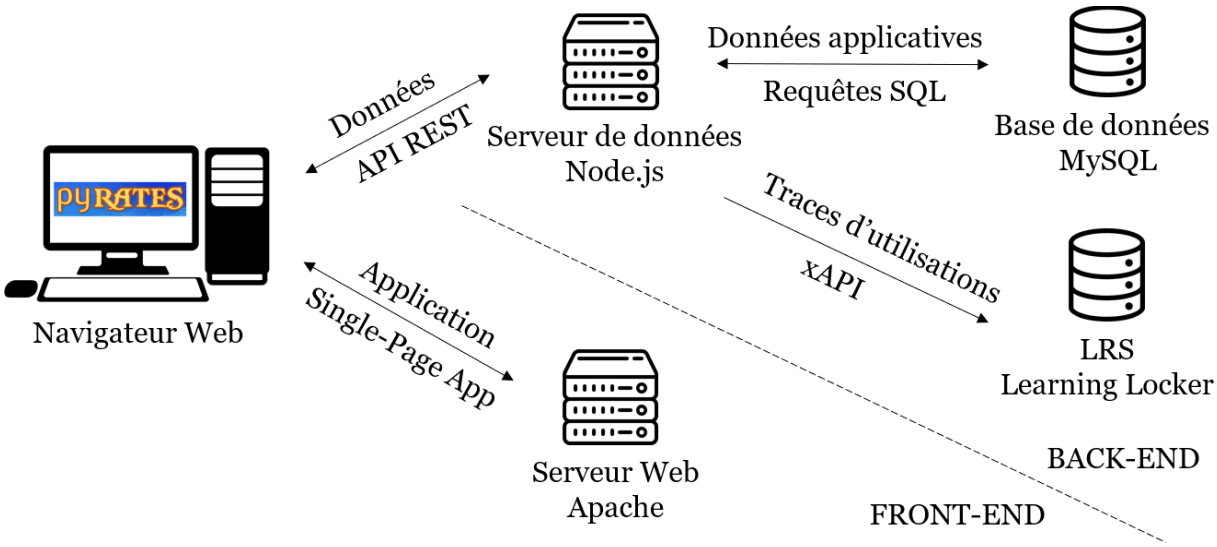


Figure 5-15 – Architecture générale de l'application Pyrates

D'abord, côté front-end, le logiciel est accessible en ligne depuis un navigateur Web. Plus précisément, l'application prend la forme d'une Single-Page App (SPA) hébergée par un serveur Web. Dans une SPA, il n'y a pas de rafraîchissement des pages par appels au serveur Web lors de la navigation comme c'est le cas pour un site classique. Au lieu de cela, tout le code HTML, JavaScript et CSS est récupéré par le navigateur en une seule fois lors du chargement initial de la page d'accueil du site. L'application interagit ensuite avec l'utilisateur en réécrivant dynamiquement la page web courante avec de nouvelles données provenant du serveur de données. L'objectif est d'accélérer les transitions pour que le site web ressemble davantage à une application de bureau (Li & Zhang, 2021).

Ensuite, côté back-end, dans l'architecture que nous avons choisie, les données de l'application sont fournies à la demande par un serveur *Node.js*. L'accès à ces données est assuré par un ensemble de service web de type REpresentational State Transfert (REST). Les services Web conformes à REST fournissent aux demandeurs des données textuelles (format JSON) en réponse à des requêtes HTTP en fonction des méthodes utilisées (POST, GET, PUT, etc.) et des paramètres qu'elles contiennent (Masse, 2011). Selon le service web appelé, les données applicatives (celles qui permettent l'enregistrement des parties) sont stockées ou récupérées dans une base de données *MySQL*. Les traces d'utilisations (mode recherche) sont, quant à elles, envoyées au format *xAPI* dans un RLS.

Nous donnons dans les sous-sections suivantes les détails d'implémentation technique du front-end et du back-end.

5.7.2 Front-end : un jeu de plateforme dans une application SPA

Les données issues de notre enquête auprès des enseignants (voir section 4.5.6) montrent que les classes de lycée sont largement équipées en ordinateurs fixes et portables. Il n'a donc pas été jugé utile de rendre l'application compatible avec les tablettes et les smartphones. De façon similaire, les résultats de cette enquête indiquent que les navigateurs Internet Mozilla Firefox et Google Chrome sont disponibles dans la grande majorité des établissements. Nous avons par conséquent axé le développement et les tests de l'application en direction de ces deux navigateurs.

Application SPA

Afin de mettre en œuvre notre application web SPA, nous avons utilisé principalement le langage Javascript en nous appuyant sur les bibliothèques répertoriées dans le Tableau 5-10.

Bibliothèques	Usage dans Pyrates
VueJS	Framework permettant de mettre en œuvre l'application SPA
Bootstrap	Gestion des styles et de la navigation (boutons, éléments interactifs, etc.)
CodeMirror	Éditeur de code (coloration syntaxique, auto-indentation, etc.)
Skulpt	Exécution du code Python dans le navigateur Internet de l'utilisateur
Phaser	Moteur de jeu 2D basé sur la simulation physique (gravité, collisions, etc.)
Axios	Appel des services web du back-end

Tableau 5-10 – Bibliothèques Javascript utilisées pour le front-end de l'application Pyrates

VueJS et *Bootstrap* fournissent le cadre général permettant de mettre en forme les différentes parties de l'application et de gérer les aspects dynamiques (panneaux latéraux, changement de niveaux, etc.).

Jeu de plateforme

Concernant le jeu lui-même, les bibliothèques *CodeMirror*, *Skulpt* et *Phaser* permettent le contrôle des personnages par des programmes Python. La Figure 5-16 schématise cette chaîne d'exécution.

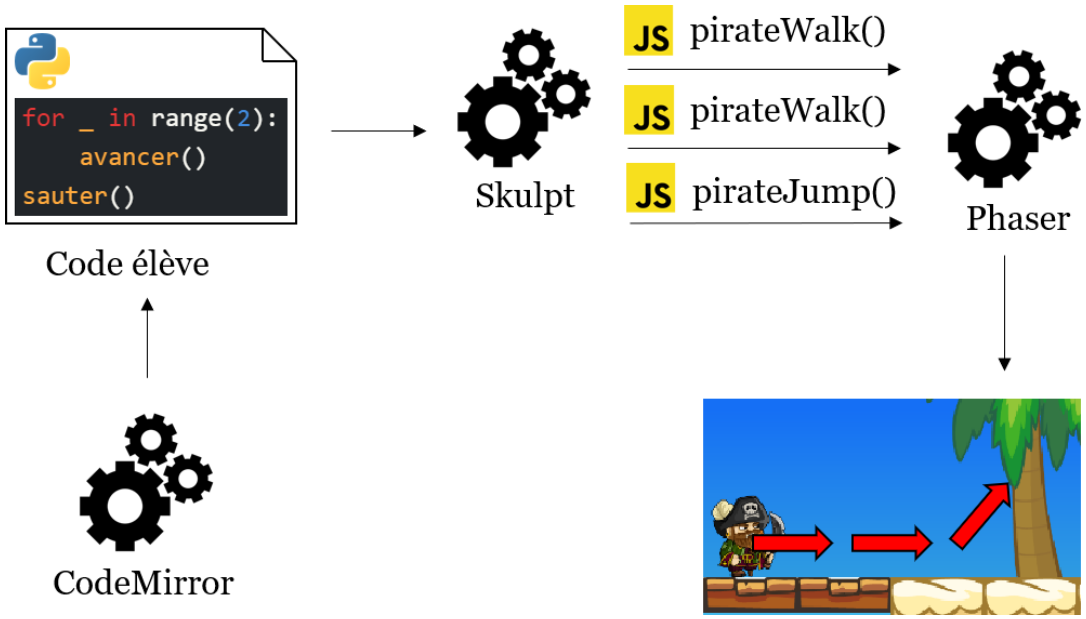


Figure 5-16 – Chaîne d'exécution d'un programme dans l'application Pyrates

Un programme Python est d'abord saisi par le joueur dans l'éditeur de code fourni par CodeMirror. Lors du lancement de ce programme (bouton « Exécuter ») le code est exécuté dans le navigateur du joueur par l'intermédiaire de Skulpt. Au cours de cette exécution, les fonctions de contrôle présentes dans le programme Python déclenchent l'appel de fonctions Javascript. Ces fonctions agissent directement sur la scène du jeu par le biais de Phaser.

La bibliothèque Phaser fournit un moteur de jeu 2D basé sur une simulation physique qui prend en charge la gravité, l'accélération et les collisions entre les différents artefacts présents dans une scène. Comme nous l'avons déjà évoqué dans la section 5.6.5, les déplacements des personnages s'appuient sur une grille de blocs semi-visible qui contraint des déplacements horizontaux et vers le haut tout en étant soumis aux collisions avec les plateformes et les objets du premier plan. Les déplacements vers le bas sont, quant à eux, uniquement régis par la gravité et les collisions. Cette contrainte de déplacement dans un grille de blocs au sein d'une simulation physique s'appuyant sur des vecteurs de vélocité et les collisions nous a obligé à mettre au point des algorithmes assez complexes afin de gérer les déplacements. Ces algorithmes contrôlent ainsi perpétuellement la position du personnage dans la boucle de rafraichissement du moteur du jeu afin qu'il s'arrête sur une « case » de la grille en fin de mouvement.

Afin de faciliter la conception des niveaux, Phaser permet de décrire leur *map* (carte), c'est-à-dire le décor dans lequel vont évoluer les différents personnages, dans des fichiers JSON. Ces fichiers détaillent les différents plans (fond, plateformes, objets, etc.) en faisant référence à des *tiles* (tuiles) qui sont les blocs élémentaires qui constituent les maps. Ces tiles sont regroupés dans un ou plusieurs fichiers de décors (*tile set*). Pour illustrer, nous reproduisons dans la Figure 5-17 une partie du tile set que nous avons utilisé pour constituer les maps des niveaux de Pyrates.

Figure 5-17 – Extrait du tile set utilisé dans les maps des différents niveaux de Pyrates

Cette ressource graphique sur le thème des pirates a été acquise sur un site de partage de créations numériques. Nous avons cependant ajouté quelques tiles comme des blocs de pierre ou un caillebotis en bois sombre. Ajoutons que nous avons utilisé l'application *Tiled* (*Tiled home page*, 2023) qui facilite la création de fichier JSON de description de maps à partir d'un tile set.

Afin d'animer les différents personnages et éléments graphiques présents dans le jeu (*sprites*), Phaser s'appuie sur des *sprite sheets*. Ces sprite sheets sont des fichiers regroupant plusieurs images décomposant les différentes animations d'un même sprite. Nous donnons par exemple dans la Figure 5-18 le sprite sheet décrivant les mouvements réalisables par Jack.

Figure 5-18 – Sprite sheet décrivant les différents mouvements de Jack

Nous avons obtenu les sprite sheets de nos personnages pirates sur un site de partage de ressources graphiques libres de droits. Il a néanmoins fallu y apporter quelques modifications. Nous avons par exemple ajouté une animation de tir au pistolet pour Jack et une animation de maniement d'épée pour Mary. Concernant les autres objets animés (coffre, clé, bouteilles, noix de coco, pots, tonneaux, bâtons de dynamite, etc.), nous avons créé nos propres sprite sheet à partir d'images libres de droits trouvées sur Internet.

Appels aux services web

Enfin, la bibliothèque *Axios* nous a servi à faire les appels aux services web exposés par le back-end de Pyrates. Nous détaillons la mise en œuvre technique de ce back-end dans la section suivante.

5.7.3 Back-end : données applicatives et traces d'utilisations

Le back-end du logiciel Pyrates permet donc d'accéder en lecture et en écriture à la base de données applicatives (mode standard) et en écriture au LRS stockant les traces d'utilisations (mode recherche). Notre serveur de données est implémenté par la plateforme javascript Node.js. Les bibliothèques utilisées sont inventoriées dans le Tableau 5-11.

Bibliothèques	Usage dans Pyrates
Express	Création de services web REST donnant accès aux données
mysql.js	Client MySQL permettant de faire des requêtes SQL
xAPI.js	Client xAPI permettant de converser avec le LRS distant

Tableau 5-11 – Bibliothèques Javascript utilisées pour le back-end de l'application Pyrates

Ainsi, *Express* permet d'exposer les différents services web d'accès aux données utilisés par le front-end. Les bibliothèques *mysql.js* et *xAPI.js* sont des clients Javascript donnant accès à la base de données de l'application et au LRS.

Données applicatives

La base de données doit permettre de sauvegarder l'état de la partie en cours, elle doit donc permettre de stocker :

- le personnage choisi (Jack ou Mary) ;
- le mode choisi (standard ou recherche) ;
- la langue choisie (FR ou EN) ;

- le plus haut niveau atteint ;
- le niveau courant ;
- la vitesse d'exécution choisie (curseur) ;
- le code saisi pour les différents niveaux commencés ou terminés ;
- la durée de jeu pour les différents niveaux commencés ou terminés (chronomètre).

Le modèle relationnel, le modèle physique ainsi que le détail des différentes tables de la base de données sont fournis en Annexe D. Cette base de données est hébergée à Clermont-Ferrand dans un datacenter de l'entreprise française *O2Switch*. Les données sont sauvegardées lors de chaque changement de niveau ou lorsque l'utilisateur clique sur le bouton « enregistrer ». Elles sont récupérées lors de la reprise du jeu ou lors de la navigation entre les différents niveaux terminés ou en cours.

Traces d'utilisations

Dans le mode recherche, l'application génère des données plus précises relatives à l'activité de l'utilisateur. Ces traces sont produites en suivant la norme xAPI et enregistrées dans un LRS (voir section 5.5.2 pour plus de détails). Comme pour le mode standard, les données sont envoyées au LRS automatiquement au début de chaque niveau ou manuellement lorsque l'utilisateur appuie sur le bouton « enregistrer ».

Cette section portait sur l'implémentation technique de l'application, nous exposons dans la suivante la conception didactique des différents niveaux du jeu.

5.8 Conception et évaluation des différents niveaux

Nous décrivons ici la conception didactique des huit niveaux du jeu en suivant le modèle des situations adidactiques (Q5.1), puis l'évaluation de ces situations (Q5.2).

5.8.1 Conception des niveaux comme situations adidactiques

Comme nous l'avons indiqué dans notre méthodologie (voir section 5.5.1), il s'agit d'abord de choisir les concepts que nous souhaitons mettre en jeu, puis de construire des situations dans lesquelles la stratégie optimale est fondée sur ces concepts.

Choix des concepts

Les concepts fondamentaux de la programmation que nous avons décidé de cibler principalement sont ceux qui ont été repérés dans les programmes comme étant communs au collège et au lycée : variable, conditionnelle, boucle for et boucle while (voir section 2.6.1). En effet, le portage de ces concepts de la modalité blocs vers la modalité texte est un des enjeux de la transition collège-lycée avant d'en introduire de nouveaux. Nous avons décidé de subdiviser ces concepts en sous-concepts afin de faciliter le changement de registres sémiotiques. Le Tableau 5-12 donne le détail des sous-concepts visés. Nous expliquons à la suite nos choix concernant ce découpage.

Code	Sous-concepts visés
for-re	Boucle for avec répétition seule (pas d'utilisation de la variable de boucle)
for-va-0	Boucle for avec utilisation de la variable de boucle qui commence à zéro

for-va-n	Boucle for avec utilisation de la variable de boucle qui ne commence pas à zéro
whi	Boucle while
var-af	Affectation de variable
if	Structure conditionnelle branche if
elif	Structure conditionnelle branche elif
else	Structure conditionnelle branche else

Tableau 5-12 – Sous-concepts de programmation mis en jeu dans Pyrates

Notre analyse didactique des différences intrinsèques entre Scratch et Python (voir section 3.3.2) montre que le concept de boucle for est implémenté en Scratch sans variable de boucle (il s'agit uniquement de répéter des instructions), contrairement à Python qui fournit une variable de boucle auto-incrémentée. Nous avons donc décidé d'introduire la boucle en Python de manière graduelle. Nous commençons donc par la répétition simple (for-re) avec la syntaxe « for _ in range(5) », avant d'intégrer progressivement l'utilisation de la variable de boucle qui commence à zéro (for-va-0) avec la syntaxe « for i in range (4) », puis qui ne commence pas forcément à zéro (for-va-n) avec la syntaxe « for i in range(1,3) ».

Nos analyses (voir section 3.3.2) indiquent également que la structure conditionnelle est limitée à deux branches en Scratch (« si-alors » ou « si-alors-sinon »). En Python, la branche « elif », en plus de celles « if » et « else », permet de créer des conditionnelles à plusieurs branches sans imbrications. Nous avons donc pris le parti de commencer par cibler la structure conditionnelle à deux branches « if-else » avant d'introduire le concept de branche « elif » dans une conditionnelle à trois branches.

Ajoutons que les niveaux de Pyrates visent également de façon secondaire les nouveaux concepts à aborder en seconde que sont les fonctions et les types primitifs des variables (voir section 2.6.1). Ces concepts sont mobilisés pour des raisons utilitaires, comme le contrôle des personnages à l'aide de fonctions ou le typage des variables qui est incontournable. Nous faisons donc le choix de diversifier les types au fil des différents niveaux : entiers, chaînes de caractères, puis booléens. Les concepts de fonction, de paramètres et de retours, sont également introduits progressivement étant donné que les fonctions ne retournent pas de valeurs en Scratch (voir section 3.3.2). Ainsi nous mettons à disposition des élèves des fonctions de contrôle d'abord sans retour et sans paramètres, puis sans retour et avec paramètres, et enfin avec retour et paramètres. À la différence des concepts principaux présentés ci-dessus, ces concepts ne font pas l'objet de contenus explicatifs dans le mémo Python (voir section 5.9.1) et leur appropriation n'est pas évaluée dans la suite.

Choix du thème

Ayant sélectionné les concepts cibles, la suite de notre méthodologie de conception consiste à choisir un univers pour notre jeu sérieux. Nous avons opté pour le thème des pirates car il est associé à un imaginaire d'aventure. Cela facilite la création de situations ludiques variées pour les différents niveaux du jeu. Cet univers a également l'avantage d'offrir un large choix de ressources graphiques disponibles sur Internet, tant au niveau des personnages que des décors (voir section 5.7.2).

D'après Rennie (2003), le thème des pirates, en particulier les personnages de fictions que l'on trouve dans la littérature et dans d'autres médias, constitue un univers genré pouvant de surcroit véhiculer des stéréotypes. Ainsi, les personnages féminins y sont pratiquement absents jusqu'au début des années 1970, et les fictions reflètent systématiquement la culture masculine. Dans les années 1990, les femmes apparaissent dans les histoires en tant que pirates, mais elles s'habillent généralement comme des hommes et jouent le rôle de héros masculins dominants et agressifs. Ce n'est que plus récemment que les héroïnes apparaissent en affichant leur identité féminine.

Toutefois, comme nous l'avons évoqué dans la méthodologie (voir partie 5.5.1), la littérature recommande de choisir des univers de jeu neutres du point de vue du genre. Par conséquent, afin de contrebalancer ce choix de thème genré, nous avons pris plusieurs dispositions qui doivent avoir un impact positif sur l'expérience de jeu féminine. Comme l'indique la méthodologie, nous avons limité les scènes de violences et de combats, nous verrons cependant dans la suite que le jeu comprend quelques actions de destruction d'obstacles (implémentation de la brique de gameplay « destruction »). Ensuite, nous proposons de choisir parmi deux personnages jouables réalistes de genres différents afin de permettre une meilleure identification des joueurs aux avatars, tout en évitant la sexualisation du personnage féminin. Ainsi, les joueurs peuvent choisir d'incarner deux personnages : Jack ou Mary (voir Figure 5-19).

Figure 5-19 – Les deux personnages jouables proposés dans Pyrates

Conception des niveaux

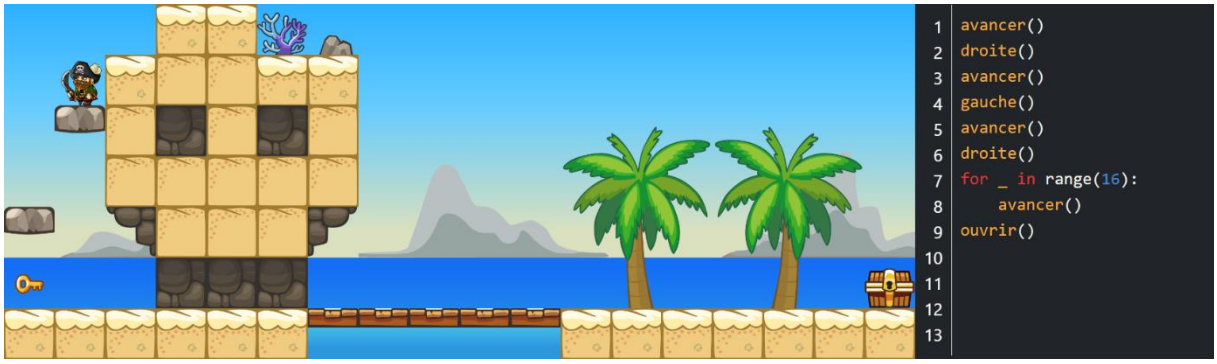
Comme nous l'indiquons dans la section 4.5.2, le jeu Pyrates a été conçu pour une durée de deux heures au regard du temps qu'accordent les enseignants à la programmation informatique au lycée. Compte tenu du nombre de concepts principaux que nous avons sélectionnés, nous avons choisi des créer huit niveaux qui devront donc être réalisables par les élèves en au plus 15 minutes chacun.

La conception adidactique de ces huit situations doit amener les élèves à mettre en œuvre les concepts sans que cela ne soit explicite. Rappelons que seul le gameplay proposé dans les niveaux doit les y conduire. Précisons que si les fonctions de contrôle utilisables dans chaque niveau sont limitées, en revanche, les concepts de programmation implémentables en Python ne sont pas contrôlés et sont tous

disponibles à tout moment. Afin d'atteindre cet objectif, rappelons également que notre but est de mettre en œuvre les conditions d'adidacticité annoncées par Bessot :

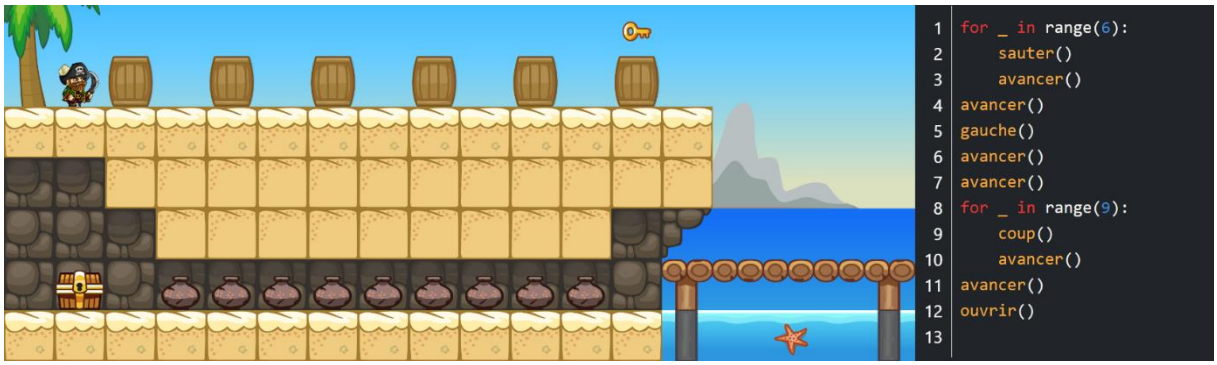
- (C1) les élèves doivent être capables d'envisager une première réponse au problème posé sous la forme d'une procédure de base peu efficace, basée sur des connaissances antérieures ;
- (C2) les concepts visés doivent permettre, lorsqu'ils sont mis en œuvre, de passer à une procédure gagnante qui résout le problème posé.

Pour ce faire, nous avons d'abord établi un objectif récurrent pour tous les niveaux du jeu. Cet objectif consiste à ramasser une clé pour ouvrir un coffre au trésor. Ensuite, la conception des maps des niveaux et le choix des briques de gameplay ont été faits de manière à rendre nécessaire l'utilisation des concepts ciblés. Ainsi, les joueurs doivent pouvoir s'engager dans les différents niveaux en codant des procédures de base utilisant les fonctions de contrôle du personnage (avancer, sauter, etc.). Ensuite, ils doivent mobiliser les concepts visés s'ils veulent réussir à ouvrir le coffre (procédure gagnante). La Figure 5-20 présente la map et une stratégie gagnante pour chacun des huit niveaux de Pyrates.



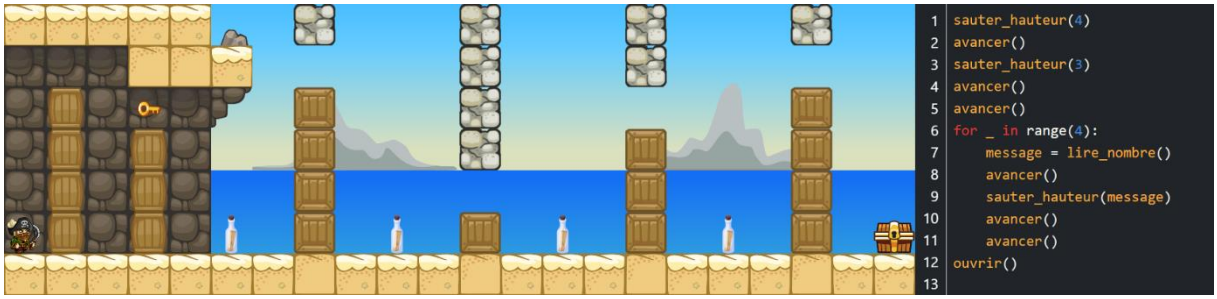
```
avancer()
droite()
avancer()
gauche()
avancer()
droite()
for _ in range(16):
    avancer()
ouvrir()
```

(a) Niveau 1



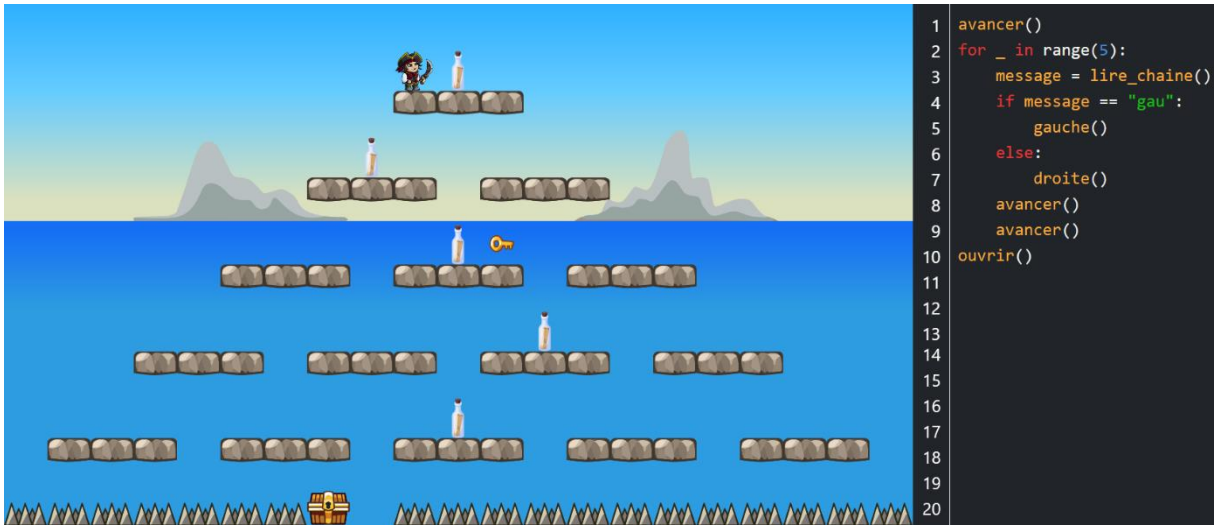
```
for _ in range(6):
    sauter()
    avancer()
avancer()
gauche()
avancer()
avancer()
for _ in range(9):
    coup()
    avancer()
avancer()
ouvrir()
```

(b) Niveau 2



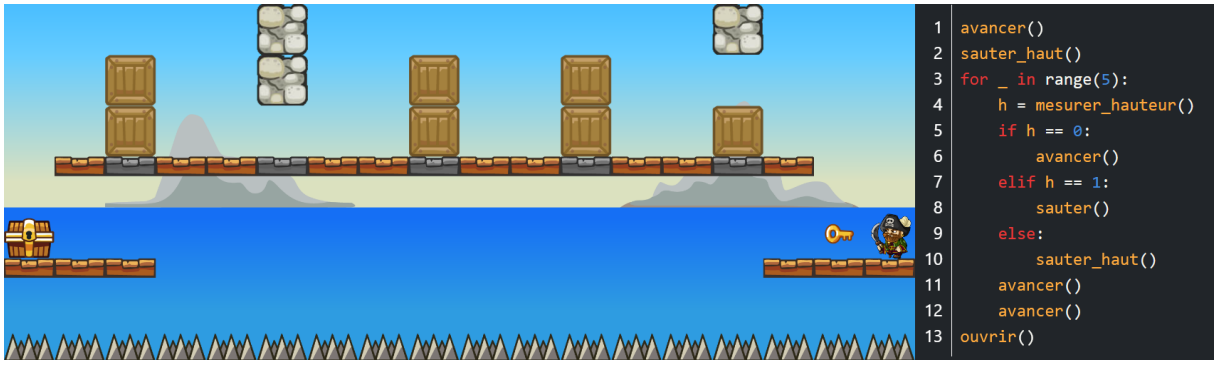
```
sauter_hauteur(4)
avancer()
sauter_hauteur(3)
avancer()
avancer()
for _ in range(4):
    message = lire_nombre()
    avancer()
    sauter_hauteur(message)
    avancer()
    avancer()
ouvrir()
```

(c) Niveau 3 : la hauteur des piles de caisses est aléatoire et change avant chaque exécution, les messages dans les bouteilles indiquent leur hauteur (1 à 5)



```
avancer()
for _ in range(5):
    message = lire_chaine()
    if message == "gau":
        gauche()
    else:
        droite()
    avancer()
    avancer()
ouvrir()
```

(d) Niveau 4 : l'emplacement de la clé et du coffre sont aléatoires et changent avant chaque exécution, les messages dans les bouteilles indiquent le chemin à suivre (« droi » ou « gau »)

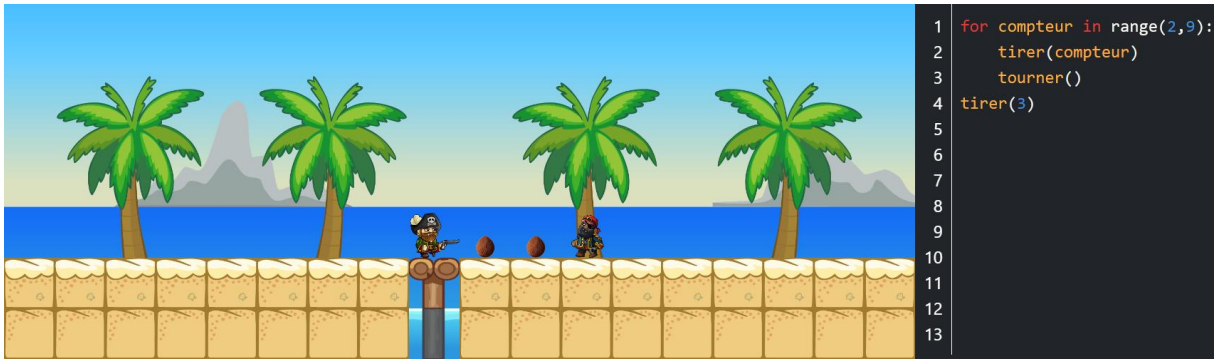


```
avancer()
sauter_haut()
for _ in range(5):
    h = mesurer_hauteur()
    if h == 0:
        avancer()
    elif h == 1:
        sauter()
    else:
        sauter_haut()
    avancer()
    avancer()
ouvrir()
```

(e) Niveau 5 : la hauteur des piles de caisses est aléatoire (0 à 2) et change avant chaque exécution

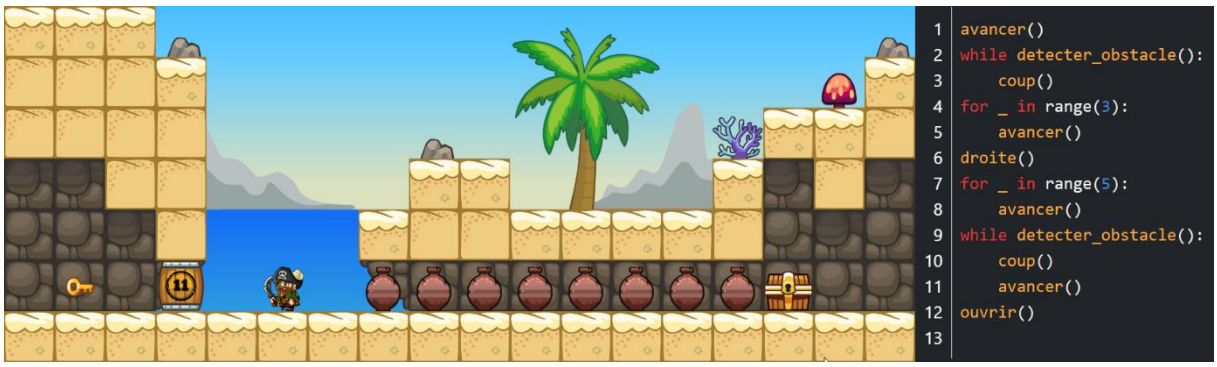
```
for cpt in range(6):
    sauter_hauteur(cpt)
    avancer()
ouvrir()
```

(f) Niveau 6 : la hauteur des piliers est fixe



```
for compteur in range(2,9):
    tirer(compteur)
    tourner()
tirer(3)
```

(g) Niveau 7 : des séries de noix de coco de plus en plus longues apparaissent de part et d'autre, l'autre pirate ne doit jamais être touché



(h) Niveau 8 : la solidité du tonneau, le nombre de pots ainsi que l'emplacement du coffre sont aléatoires et changent avant chaque exécution

Figure 5-20 – Différents niveaux de l'application Pyrates : maps et stratégies gagnantes

Comme expliqué dans la méthodologie de ce chapitre (voir section 5.5.1), la conception des niveaux a consisté à mettre en œuvre des briques de gameplay dans les différentes maps puis à réaliser plusieurs itérations d'analyses à priori des situations générées. Ces itérations ont permis de s'assurer que les concepts ciblés étaient bien nécessaires afin d'ouvrir les différents coffres et réussir les niveaux. L'analyse complète est disponible en Annexe G. Nous résumons dans le Tableau 5-13 les caractéristiques de chaque niveau : concepts principaux visés, briques de gameplay mises en œuvre (objectifs et moyens), caractéristiques de la map et de l'éditeur de code, et fonctions de contrôle disponibles. Précisons que certains niveaux comportent des contraintes dans l'éditeur de code permettant de limiter le nombre de lignes des programmes.

Niv.	Concepts	Brique obj.	Brique moy.	Map / Éditeur	Fct. contrôle
1	for-re	Atteindre	Déplacer	Parcours répétitif / Lignes limitées	avancer, droite, gauche, ouvrir
2	for-re	Atteindre, détruire	Déplacer, tirer	Parcours répétitif / Lignes limitées	avancer, droite, gauche, sauter, coup, ouvrir
3	var-af	Atteindre	Déplacer, collecter	Rétention d'informations	avancer, sauter_hauteur, lire_nombre, ouvrir
4	if, else	Atteindre, éviter	Déplacer, collecter	Parcours aléatoire	avancer, gauche, droite, lire_chaine, ouvrir
5	if, elif, else	Atteindre, éviter	Déplacer, collecter	Parcours aléatoire	avancer, sauter, sauter_haut, mesurer_hauteur
6	for-va-o	Atteindre	Déplacer	Parcours structuré / Lignes limitées	avancer, sauter_hauteur, ouvrir
7	for-va-n	Atteindre, détruire, éviter	Déplacer, tirer	Parcours structuré / Lignes limitées	tourner, tirer
8	whi	Atteindre, détruire	Déplacer, tirer, collecter	Parcours répétitif et aléatoire	avancer, gauche, droite, coup, detecter_obstacle, ouvrir

Tableau 5-13 – Caractéristiques des différents niveaux du jeu Pyrates

Le but commun de collecte d'une clé afin d'ouvrir un coffre implique la mise en œuvre d'une brique objectif de type « atteindre » et d'une brique de moyen de type « déplacer » pour tous les niveaux. Cela entraîne la mise à disposition des fonctions de contrôle de déplacement : avancer, gauche, droite, sauter, etc.

Détaillons maintenant, concept par concept, les caractéristiques des niveaux qui doivent mener à leur mobilisation par les élèves.

Le concept de boucle-répétition (for-re) est mis en jeu par l'intermédiaire de parcours répétitifs dans les maps associés à une contrainte de limitation de lignes dans l'éditeur de code. C'est par exemple le cas de la longue ligne droite dans le niveau 1 (voir Figure 5-20-a) ou des séries de tonneaux à franchir et des pots à détruire dans le niveau 2 (voir Figure 5-20-b).

La mise en œuvre du concept de variable (var-af) est rendue nécessaire par le besoin de stocker temporairement une information. Ainsi dans le niveau 3, la hauteur fluctuante des piles de caisses à franchir est indiquée dans des bouteilles qui les précèdent (voir Figure 5-20-c). Cela appelle l'utilisation d'un moyen de rétention d'information entre la lecture de l'information et sa restitution lors du saut. Notons qu'il est ici nécessaire d'implémenter la brique de moyen « collecter » afin d'avoir accès aux informations dans les bouteilles (fonction lire_nombre).

La présence de l'aléatoire dans les maps des niveaux 4 et 5 a pour objectif de contraindre les élèves à utiliser une structure conditionnelle et ses différents types de branches en fonction de la situation. Par exemple, le chemin aléatoire vers la clé et le coffre dans le niveau 4 (voir Figure 5-20-d) doit permettre d'imposer l'utilisation d'une conditionnelle à deux branches (if-else). Les hauteurs aléatoires des différentes piles de caisses du niveau 5 associées aux trois fonctions de saut disponibles (voir Figure 5-20-e) ont pour but de contraindre l'implémentation d'une conditionnelle à trois branches (if-elif-else). Ces parcours aléatoires doivent également être accompagnés d'une brique de moyen « collecter » qui assure la prise d'information via des fonctions de contrôle avec retour : lire_chaine ou mesurer_hauteur.

La conception des maps selon un parcours structuré, comme un escalier dans le niveau 6 (voir Figure 5-20-f) ou des séries de noix de coco de plus en plus longues dans le niveau 7 (voir Figure 5-20-g), associée à une limitation des lignes de code, doivent inciter les joueurs à mettre en œuvre les variables de boucles (for-va-0 et for-va-n).

Enfin, la conjonction d'un parcours aléatoire et répétitif, comme la destruction d'un tonneau qui possède une solidité aléatoire, ou d'une suite indéterminée de pots dans le niveau 8 (voir Figure 5-20-f), sont en mesure d'amener à l'utilisation d'une boucle while (whi). Afin de pouvoir tirer profit de l'aspect conditionnel de la boucle while, il est ici aussi nécessaire d'implémenter la brique de moyen « collecter » par l'intermédiaire de la fonction de contrôle detecter_obstacle.

Synthèse de la sous-section

Q5.1 : Comment concevoir les niveaux du jeu Pyrates sur le modèle des situations adidactiques en ciblant les concepts fondamentaux de la programmation étudiés au collège ?

En suivant notre méthodologie basée sur la mise en œuvre de briques de gameplay et sur l'analyse à priori des situations élaborées, nous sommes parvenu à concevoir les huit niveaux de jeu avec les résultats suivants :

- les **concepts** de programmation que nous avons décidé de cibler principalement sont **communs** aux programmes du **collège** et du **lycée** : **variable**, **conditionnelle**, **boucle for** et **boucle while**, l'objectif étant d'accompagner le changement de registres sémiotiques des blocs vers le texte ;
- le **thème** du jeu est celui des **pirates**, cet univers n'étant **pas neutre** du point de vue du genre, nous avons **limité** les **scènes** de **violence** et de **combat** et proposé le choix de **deux personnages mixtes** à même de favoriser l'**identification** des joueurs et des joueuses ;
- l'**objectif** commun aux huit niveaux du jeu est de **collecter une clé** permettant d'**ouvrir un coffre** au trésor, ensuite les différentes maps ont été conçues pour rendre nécessaire l'implémentation des concepts visés : **parcours répétitifs ou structurés** avec limitation des lignes pour les **boucles for**, **rétention d'information** pour les **variables**, **parcours aléatoires** et présence de fonction de collecte de données pour les **conditionnelles**, et **parcours répétitifs et aléatoires** avec fonction de collecte de données pour les **boucles while**.

Nous venons de présenter la conception des différents niveaux du jeu suivant le modèle adidactique. Il s'agit maintenant d'évaluer cette conception en analysant les traces d'activités des élèves lors d'expérimentations que avons menées sur le terrain, ainsi qu'en prenant en compte les programmes exécutés par les utilisateurs finaux dans le version standard de l'application qui est en production.

5.8.2 Évaluation de la conception

Cette sous-section répond à la question de recherche Q5.2, il s'agit de se demander si les niveaux du jeu Pyrates permettent aux élèves d'appréhender les différents concepts de programmation ciblés dans des conditions adidactiques. Selon Bessot, cela revient à avoir l'opportunité d'essayer des procédures de base non efficaces avant de pouvoir gagner les niveaux en mettant en œuvre les concepts ciblés. L'analyse qui suit est basée sur les traces d'activités des élèves concernant les programmes exécutés. Nous exploitons plus particulièrement parmi ces traces les concepts mis en œuvre et les procédures adoptées. Comme expliqué dans la partie méthodologie (voir 5.5.2), nous différencions selon les figures, les données des élèves de l'expérimentation (traces d'activités détaillées pour 240 élèves), et les données issues des utilisateurs finaux en production (traces sommaires pour 75 000 utilisateurs).

Procédures de base

Déterminons d'abord si les niveaux conçus permettent aux élèves d'envisager une réponse initiale pour s'engager dans le problème posé. La Figure 5-21 représente la fréquence d'apparition des différents concepts dans les programmes conduisant à des procédures non gagnantes pour chaque niveau. Précisons que ces programmes non

gagnants ont été entièrement exécutés, ils sont donc syntaxiquement corrects. Cependant, ils ne permettent pas de terminer le niveau, c'est-à-dire de ramasser la clé et d'ouvrir le coffre au trésor.

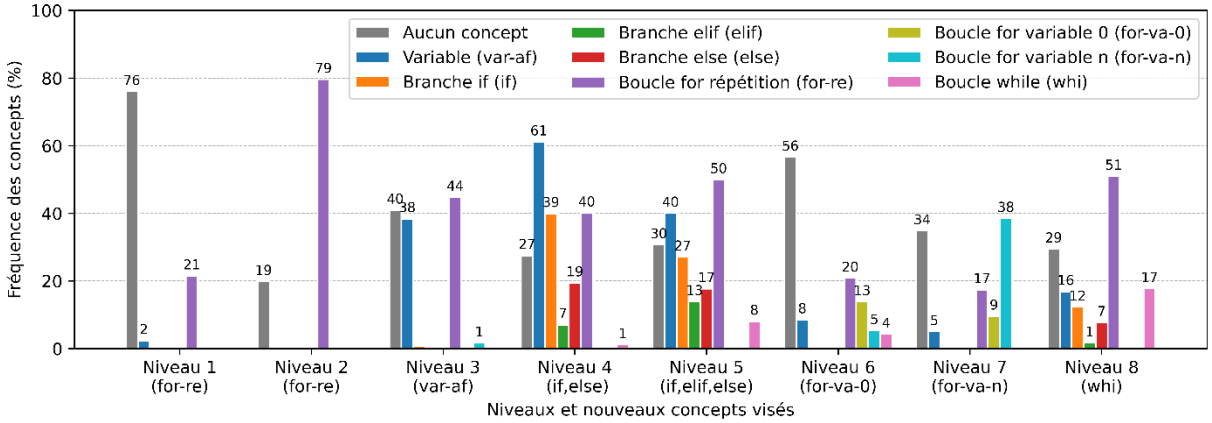


Figure 5-21 – Concepts mis en œuvre dans les procédures non-gagnantes exécutées par les 240 élèves de l'expérimentation

Cette figure montre que les élèves mettent en œuvre des procédures de base sans concepts (c'est-à-dire en utilisant uniquement des fonctions de contrôle) dans tous les niveaux. Ceci semble être particulièrement le cas lorsque le concept visé est nouveau et non une déclinaison d'un concept précédemment mis en œuvre (niv.1, niv.3, niv.6 et niv.8). Nous observons également des essais que nous nommons procédures intermédiaires mettant en œuvre le concept visé dans le niveau mais nécessitant quelques ajustements sémantiques afin de le terminer. En résumé, on peut affirmer que les joueurs sont capables de s'engager dans les niveaux grâce à des procédures de base composées uniquement de fonctions de contrôle.

Procédures gagnantes

Ensuite, afin de vérifier la deuxième condition énoncée par Bessot, nous nous demandons si la mise en œuvre des concepts visés permet de terminer les niveaux et si c'est le seul moyen de le faire. En d'autres termes, le concept ciblé dans un niveau est-il une condition nécessaire et suffisante pour établir une procédure gagnante ? Nous répondons à cette question tour à tour pour chaque concept.

Nous nous appuyons pour cela sur les Figure 5-22 et Figure 5-23 qui détaillent les procédures gagnantes mises en œuvre respectivement par les élèves de notre expérimentation et par les utilisateurs finaux de l'application. Pour ces deux figures, les noms des procédures font référence à l'analyse à priori détaillée en Annexe G. Rappelons que cette analyse à priori a été mise à jour après l'expérimentation à partir des procédures gagnantes exécutées par les 240 élèves qui y participaient. Cela explique l'absence de procédures de type « Other » dans la Figure 5-22. En revanche, les très nombreuses procédures gagnantes soumises par les utilisateurs finaux (n=261 767) ne nous ont pas permis d'actualiser l'analyse à priori avec les procédures non anticipées qui s'y trouvent. Nous retrouvons par conséquent des procédures de type « Other » dans la Figure 5-23. Enfin, du fait du temps limité dont ils disposaient et de la difficulté croissante des niveaux, les élèves de l'expérimentation n'ont pas tous pu accéder aux derniers niveaux du jeu : seuls 29 sur 240 élèves terminent le niveau 7,

et 18 élèves parviennent à achever le niveau 8. Nous privilégierons donc les données issues des utilisateurs finaux qui sont plus robustes pour ces deux derniers niveaux.

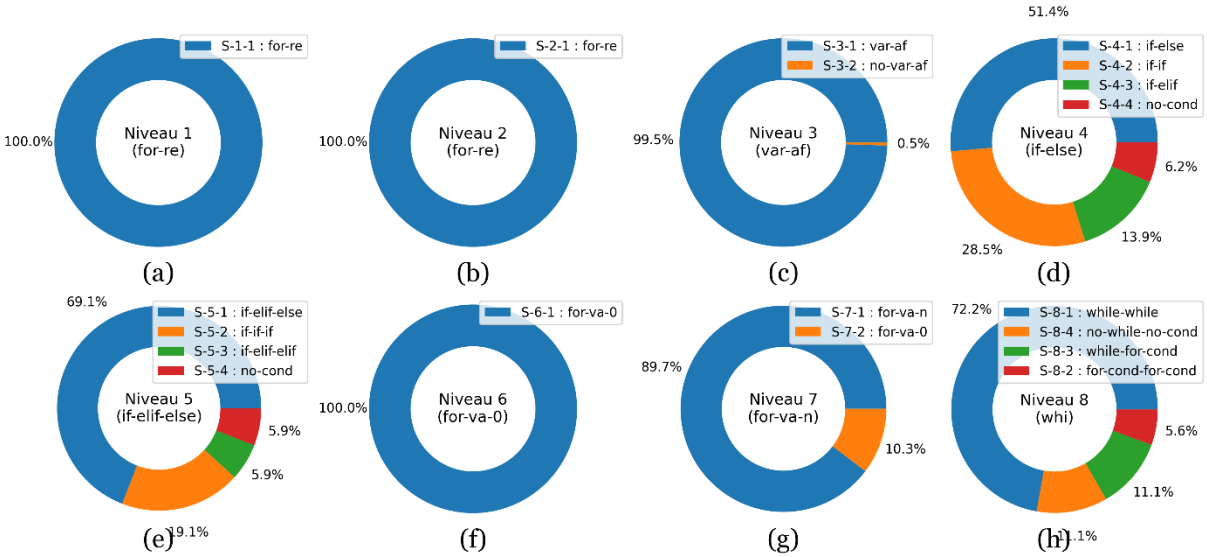


Figure 5-22 – Procédures gagnantes mises en œuvre par les 240 élèves de l'expérimentation pour chaque niveau du jeu

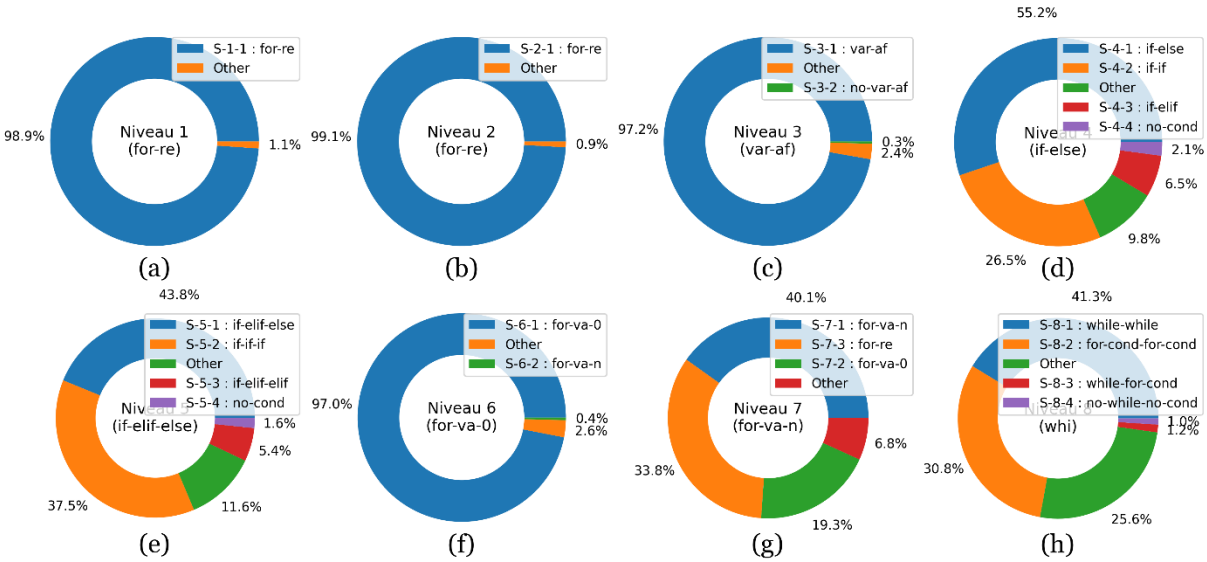


Figure 5-23 – Procédures gagnantes mises en œuvre par les 75 000 utilisateurs finaux pour chaque niveau du jeu

Pour les niveaux 1 et 2, qui ciblent la boucle for répétition (for-re), les Figure 5-22-a et Figure 5-22-b indiquent qu'elles sont systématiquement implémentées par les élèves de l'expérimentation. Notons que c'est également le cas pour 99% des utilisateurs finaux (voir Figure 5-23-a et Figure 5-23-b). Ceci valide la conception des niveaux 1 et 2 pour ce concept de boucle for répétition.

Concernant le concept de variable (var-af) visé par le niveau 3, nous pouvons voir dans les Figure 5-22-c et Figure 5-23-c qu'il est quasi-systématiquement mis en œuvre par les élèves de l'expérimentation et les utilisateurs finaux. Seule une infime partie des joueurs parviennent à y échapper en programmant un parcours fixe sans se soucier des variations aléatoires de la carte. Il exécutent ensuite leur code à de très nombreuses

reprises (boucles « Exécuter »-« Arrêter ») jusqu'à obtention de la configuration désirée (voir S3.2 en Annexe G.3.4). Nous qualifions cette stratégie de *contournement didactique* car elle permet de réussir des niveaux sans mettre en œuvre les concepts de programmation en jeu. Ces élèves qui restent à tout prix dans le domaine ludique ne veulent pas ou ne peuvent pas entrer dans un apprentissage conceptuel en explorant le milieu didactique à la recherche d'un concept qui pourrait leur permettre de terminer le niveau. Le coût très important de cette stratégie pour ce niveau (0,16% de chance de réussite, voir Annexe G.3.3) explique sans doute sa faible utilisation. Nous pouvons conclure que notre conception est validée pour le concept de variable dans le niveau 3.

La conditionnelle (if, elif et else) est un concept ciblé par les niveaux 4 et 5. Les Figure 5-22-d et Figure 5-22-e montrent que la grande majorité des élèves de l'expérimentation la mettent en œuvre (autour de 93%). C'est aussi le cas d'un grand nombre d'utilisateurs finaux (voir Figure 5-23-d et Figure 5-23-e). Néanmoins, environ 6% des élèves et au moins 2% des utilisateurs finaux utilisent un mode opératoire de contournement didactique similaire à celui que nous avons décrit ci-dessus (voir S4.4 en Annexe G.4.4 et S5.4 en Annexe G.5.4). Il est probable que, pour les niveaux 4 et 5, ces stratégies ne sont pas assez coûteuses en comparaison du niveau 3. Ainsi avec des taux de succès aléatoire de respectivement de 3,1% (voir Annexe G.4.3) et 0,4% (voir Annexe G.5.3), il est sans doute plus attractif ici pour un élève de tenter sa chance. Une manière de limiter l'usage de ces stratégies d'évitement serait d'augmenter leur coût en jouant sur les variables didactiques. Il est par exemple possible d'ajouter un niveau sur la plateforme pour le niveau 4 ou d'adjoindre une pile de caisses supplémentaire au niveau 5. Ensuite, si nous entrons dans le détail des branches utilisées par les joueurs, les Figure 5-22-d et Figure 5-23-d indiquent que pour la conditionnelle à deux branches (if-else), plus d'un tiers des joueurs utilisent une combinaison if-if ou if-elif qui fonctionne sans être optimale. Ces stratégies (voir S4.2 et S4.3 en Annexe G.4.4) impliquent des tests inutiles (si le message n'est pas « gau », c'est forcément « droi ») qui sont sans conséquence dans un contexte scolaire dans lequel la contrainte d'efficacité algorithmique ne pèse pas. Nous retrouvons logiquement le même phénomène pour la conditionnelle à trois branches (if-elif-else) dans le niveau 5, avec des stratégies non optimales if-if-if ou if-elif-elif (voir S5.2, S5.3 en Annexe G.5.4) utilisées par un quart des élèves de l'expérimentation (voir Figure 5-22-e) et près de la moitié des utilisateurs finaux (voir Figure 5-23-e). Nous ne disposons pas en l'état de variable didactique sur laquelle agir afin d'augmenter le coût de ces stratégies correctes mais moins efficaces. Une contrainte portant sur le temps d'exécution des programmes pour ces deux niveaux pourrait être une piste à suivre. En conclusion, nous pouvons dire que le concept de conditionnelle est mis en œuvre par la majorité des utilisateurs ce qui valide notre conception. Cependant, un ajustement des variables didactiques doit pouvoir limiter l'usage des stratégies de contournement didactique et des stratégies moins efficaces.

Les niveaux 6 et 7 mettent en jeu les concepts de boucle avec variable (for-va-0 et for-va-n). Concernant les variables de boucle qui commencent à zéro (for-va-0), nous pouvons constater que la totalité des élèves de l'expérimentation (voir Figure 5-22-h) et 97% des utilisateurs finaux (voir Figure 5-23-h) l'implémente. Cependant, pour les variables de boucle qui ne commencent pas à zéro (for-va-n), nous remarquons que 10,3% (Figure 5-22-g) des élèves, et 19,3% des utilisateurs finaux (Figure 5-23-g) parviennent à gagner le niveau 7 avec un boucle qui commence à zéro (for-va-0). Ils

ont pour cela recours à une astuce consistant à utiliser un décalage d'indice dans le corps de la boucle (voir S7.2 en Annexe G.7.4). Ajoutons que 33,8% des utilisateurs finaux terminent le niveau 7 sans utiliser la variable de boucle mais uniquement une boucle répétition (for-re). Pour y parvenir, ils utilisent un artifice de Python permettant d'inclure plusieurs instructions sur une même ligne à l'aide d'un point-virgule. Ils peuvent ainsi s'affranchir des contraintes limitant les lignes de code et implémenter eux-mêmes un compteur en le déclarant puis en l'incrémentant explicitement (voir S7.3 en Annexe G.7.4). L'utilisation de cette dernière stratégie par un tiers des joueurs peut être relativisée dans la mesure où ils ont pour la plupart déjà utilisé la variable de boucle dans le niveau précédent. Cependant, elle pose question quant au passage de l'utilisation du « range » d'un à deux arguments qui, malgré les contenus explicatifs disponibles (voir mémo Python dans section 5.9.1), est plus difficile que ne nous l'avions anticipé. Au bilan, nous pouvons affirmer que notre conception du niveau 6 est validée pour le concept for-va-0, en revanche, le niveau 7 doit être revu pour le concept for-va-n afin d'empêcher l'utilisation du point-virgule et ainsi bannir l'enchaînement d'instructions sur une même ligne.

Enfin, le niveau 8 cible le concept de boucle while (whi). Comme le montre la Figure 5-22-h, environ 80% des élèves de l'expérimentation la mettent en œuvre. En revanche, la Figure 5-23-h indique que seuls 40% des utilisateurs finaux l'utilisent pour ce niveau. Les autres joueurs parviennent à remplacer le concept de boucle while par l'imbrication d'une conditionnelle dans une boucle for (voir S8.2 en Annexe G.8.4). Cette *équivalence conceptuelle* est possible car les utilisateurs parviennent à majorer le nombre maximum d'itérations attendues malgré son caractère aléatoire. C'est par exemple le cas pour la solidité du tonneau (valeur aléatoire entre 1 et 20) et pour le nombre de pots avant le coffre (valeur aléatoire entre 1 et 10). Il est probable qu'une augmentation de la valeur de ces variables didactiques ne résolve pas le problème dans la mesure où il sera toujours possible de choisir un très grand nombre de tours de boucle for sans surcoût pénalisant. Afin de bloquer l'usage de cette stratégie d'équivalence conceptuelle, nous pourrions limiter le nombre de lignes de code dans le niveau 8 afin d'empêcher l'utilisation de la conditionnelle associée à la boucle for répétition. Enfin, nous retrouvons comme pour tous les niveaux basés sur le hasard, une part des élèves de l'expérimentation (11% voir Figure 5-22-h) et des utilisateurs finaux (1% voir Figure 5-23-h) qui utilisent une stratégie de contournement didactique ne mettant en œuvre aucun test (voir S8.4 en Annexe G.8.4). Le taux de réussite à priori de ces stratégies fixé ici à 0,5% (voir Annexe G.8.3) pourrait être réduit en augmentant la valeur de la variable didactique de la solidité du tonneau. Au bilan, les concept de boucle while n'est implémenté dans ce niveau que par une minorité de joueurs. Les autres parviennent à s'en passer en utilisant une équivalence conceptuelle permise par la majoration de variables didactiques ou par contournement didactique. Limiter le nombre de lignes de code et augmenter la solidité du tonneau doit permettre de limiter ces stratégies.

Synthèse de la sous-section

Q5.2 : Quel est le niveau d'adidacticité effectif des situations conçues lorsqu'elles sont soumises à des élèves ?

En analysant les programmes exécutés, et en particulier les concepts et les procédures mises en œuvre par les joueurs, nous avons vérifié si les situations conçues respectaient les conditions d'adidacticité établies par Bessot, nos résultats sont les suivants :

- les joueurs ont la capacité de s'**engager** dans les niveaux par le biais de **procédures de base** constituées uniquement de **fonctions de contrôle** ;
- la conception des niveaux est **très efficace** pour conduire à la mise en œuvre des concepts de **boucle for** et de **variable**, mais n'est **pas suffisante** pour imposer systématiquement l'utilisation des concepts basés sur des tests (**conditionnelle** et **boucle while**) ;
- un ajustement des **variables didactiques** ainsi que le **bannissement** du **point-virgule** dans les programmes sont en mesure de **limiter** l'usage des stratégies de **contournement didactique** et d'**équivalence conceptuelle**.

Puisqu'elles **respectent** globalement les **conditions d'adidacticité** édictées par Bessot, nous pouvons estimer que la conception des huit situations que constituent les niveaux du jeu peuvent « être vécue comme adidactique » par les élèves.

Nous venons de présenter l'évaluation de la conception adidactique des niveaux du jeu et de proposer des solutions permettant de l'améliorer. Nous exposons dans la section suivante l'aménagement de l'environnement de Pyrates en vue d'accompagner la transition Scratch-Python.

5.9 Fonctionnalités pour la transition Scratch-Python

Le deuxième aspect de la conception de l'application Pyrates que nous évoquons dans ce chapitre est la mise en œuvre de fonctionnalités dans le but de soutenir le passage de la programmation en Scratch vers la programmation en Python. Nous exposons d'abord l'aménagement de l'environnement (Q5.3) puis son évaluation dans les classes (Q5.4).

5.9.1 Aménagement de l'environnement

La présentation est basée sur la Figure 5-24 qui décrit l'interface utilisateur de l'application et ses différentes zones.

Comme nous l'avons expliqué dans la méthodologie de ce chapitre (voir section 5.5.1) cette partie de l'application a été conçue sur la base des résultats établis dans le Chapitre 3 concernant les différences entre Scratch et Python. Ainsi, certaines caractéristiques des environnements de programmation par blocs (catalogue de commandes, composition des programmes, moins d'erreurs, exécution améliorée) ont été incorporées en raison de leurs avantages mis en évidence dans la littérature.

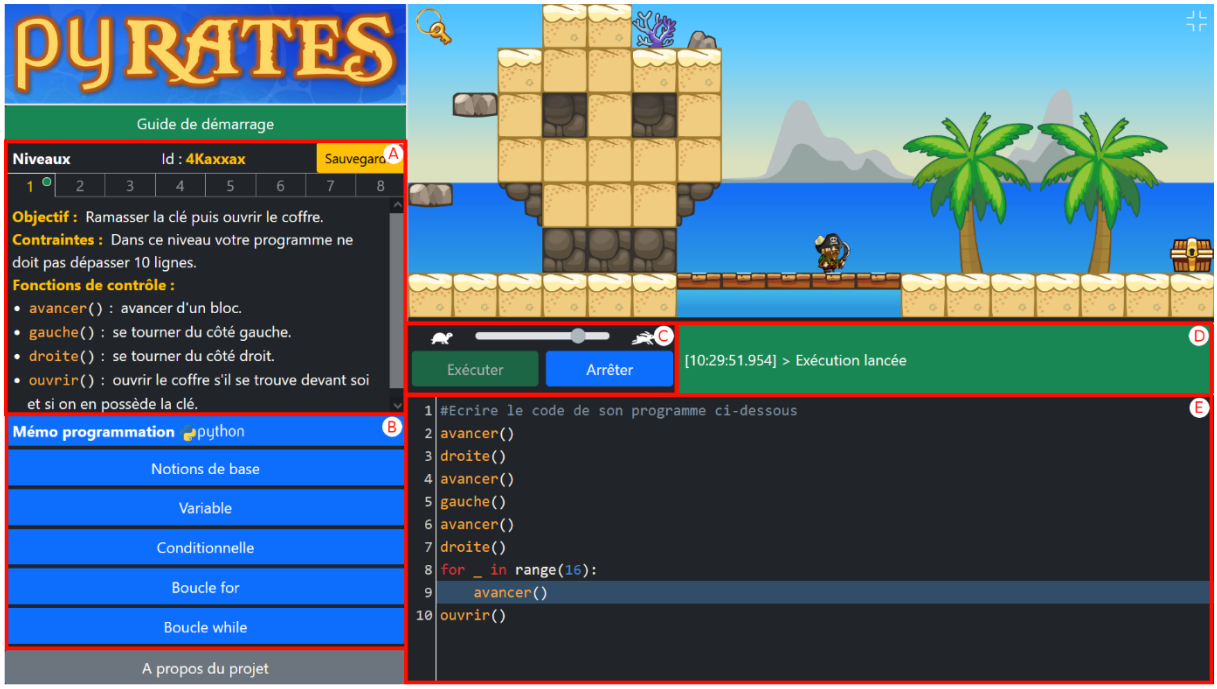


Figure 5-24 – Différentes zones de l'interface utilisateur de l'application Pyrates : zone des niveaux (a), mémo Python (b), panneau de commande (c), console (d) et éditeur de code (e)

Mémo Python

Tout d'abord, une barre latérale fixe a été ajoutée sur la gauche de l'écran et comprend, entre autres, un « mémo programmation Python » (voir Figure 5-24-b). Ce mémo est une documentation synthétique expliquant les bases de la programmation en Python. Cet élément a été conçu en s'inspirant du catalogue de commandes présent dans l'environnement Scratch. Le contenu du mémo est accessible en cliquant sur les différents boutons bleus qui indiquent des noms de concepts. Ces contenus portent sur les concepts fondamentaux de la programmation communs au collège et au lycée que nous avons choisi de mettre en jeu dans l'application : variable, conditionnelle, boucles for et while (voir section 5.8.1). Nous avons également ajouté une partie relative aux « notions de base » que sont les concepts élémentaires communs à tous les langages de programmation textuels : programme, erreurs, structuration et commentaires. Lorsque le pointeur de la souris survole un bouton, le titre est modifié pour refléter l'utilité du concept afin de guider les utilisateurs dans leurs recherches. Par exemple, « Variable » devient « Garder des informations en mémoire ».

En cliquant sur un bouton, un panneau latéral apparaît, expliquant le concept choisi. Nous reproduisons par exemple dans la Figure 5-25 deux extraits du mémo relatifs à la boucle for et à la conditionnelle. L'intégralité des panneaux du mémo Python sont reproduits en Annexe I. Ainsi, pour chaque concept, le mémo commence par rappeler son utilité, par exemple « Permet de répéter des instructions un certain nombre de fois » pour la boucle for. Ensuite le concept est détaillé en sous-concepts. Ces sous-concepts sont les mêmes que pour les concepts principaux visés par les niveaux (voir section 5.8.1).

(a) (b)

Figure 5-25 – Extraits du mémo Python concernant la boucle for (a) et la conditionnelle (b)

Chaque sous-concept est expliqué et suivi d'un modèle générique (sorte de grammaire simplifiée) et d'un exemple illustratif (voir Figure 5-25). Ces programmes sont exprimés à la fois en Python et en Scratch. Le modèle générique en Python et sa traduction en Scratch ont pour but d'aider les apprenants à passer d'un registre sémiotique à l'autre ce qui, comme nous l'avons vu dans la section 3.3.3, peut constituer un obstacle pour les élèves. L'intention est de favoriser l'appréhension de la syntaxe Python par plus grands fragments plutôt qu'élément par élément. Pour illustrer ce point, dans la partie « Répétition simple » du panneau « Boucle for » (voir Figure 5-25-a), les apprenants doivent se concentrer sur le nombre entre parenthèses à la suite du « range » et essayer de traiter le reste du code de la boucle comme un seul agrégat. La présence d'exemples est justifiée par des résultats de recherche. Ainsi Ginat et ses collègues (2011) affirment que l'inclusion d'exemples illustratifs dans les contenus pédagogiques permet de réduire la charge cognitive des étudiants et de les rendre plus efficaces lorsqu'ils abordent de nouveaux problèmes.

Enfin, nous avons agrémenté les contenus du mémo d'éléments issus de notre analyse des différences Scratch-Python (section 3.3.2). Par exemple, le panneau du concept de boucle while contient l'indication suivante précédée d'un panneau danger : « en Python la boucle tourne tant que la condition est vraie, en Scratch la boucle s'arrête quand la condition est vraie » (voir Annexe I.5).

Copiés-collés

Dans le mémo, chaque extrait de code Python est associé à un bouton de copie afin de réduire les saisies au clavier. L'objectif est ici d'encourager la pratique du copier-coller du mémo vers l'éditeur de code. Cette méthode peut être considérée comme une forme de continuité de la fonctionnalité glisser-déposer des environnements basés sur les blocs comme Scratch.

Erreurs de programmation

Malgré les efforts de conception présentés ci-dessus visant à réduire les erreurs de syntaxe, il semble illusoire de s'attendre à leur éradication complète comme c'est le cas lors de l'édition en Scratch. Les élèves devront donc irrémédiablement faire face à des messages d'erreur lors de la mise au point de leurs programmes Python.

Becker (2016) a établi que la compréhension des messages d'erreur est un défi majeur pour les programmeurs inexpérimentés. C'est pourquoi nous avons décidé d'enrichir l'environnement de l'application en intégrant un analyseur syntaxique issu de la recherche et créé spécialement pour les débutants (Kohn, 2017). Ce module analyse les programmes Python des joueurs avant l'interpréteur Python. Il délivre des messages d'erreur dans la langue maternelle des utilisateurs (français, anglais, hollandais, allemand, italien et russe pour le moment). En effet, selon Qian et Lehman (2016), la maîtrise de l'anglais par les étudiants non anglophones est significativement corrélée à leur réussite dans l'apprentissage de la programmation. Les énoncés des messages issus de ce module sont également formulés dans un style moins technique et plus direct et donc mieux adapté aux novices. Nous avons malgré tout légèrement modifiés les messages prédéfinis dans l'analyseur afin de mettre le vocabulaire en cohérence avec la formulation du mémo. Ainsi, lorsqu'une erreur de syntaxe se produit, un message amélioré s'affiche dans la console de Pyrates et la ligne de code erronée est surlignée en rouge dans la zone d'édition des programmes. Nous donnons dans la Figure 5-26 un exemple d'erreur d'indentation.

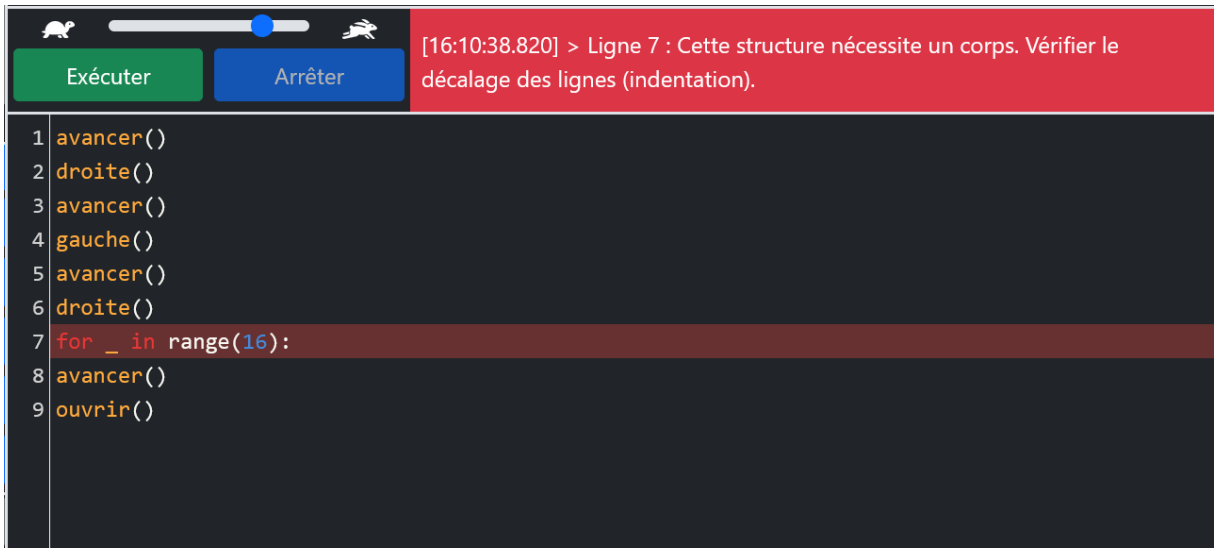


Figure 5-26 – Détail de l'interface utilisateur de Pyrates lors de la survenue d'une erreur syntaxique : message d'erreur et mise en valeur de la ligne erronée

Précisons que le message d'erreur standard de l'interpréteur Python pour une telle erreur d'indentation est : « IndentationError: expected an indented block after 'for' statement ».

Ajoutons qu'un programme sans erreur de syntaxe ne signifie pas que le code est exécutable. Des erreurs de type ou d'autres erreurs sémantiques peuvent encore apparaître pendant l'interprétation. Elles sont dans ce cas détectées par l'interpréteur Python standard implémenté par la bibliothèque Skulpt (voir section 5.7.2) et associées à des messages d'erreurs classiques qui sont remontés dans la console de Pyrates.

Contrôle et suivi de l'exécution

Enfin, un panneau de contrôle (voir Figure 5-24-c) a été développé pour améliorer la gestion des exécutions. Les joueurs ont la possibilité de démarrer et d'arrêter l'exécution du code et de modifier sa vitesse à l'aide d'un curseur. Ce curseur modifie la vitesse des mouvements des personnages en agissant sur un facteur multiplicateur. Au lancement du jeu, ce facteur est fixé à 1 (tortue), et peut être augmenté jusqu'à 3 (lièvre).

Le suivi de l'exécution est garanti par la mise en évidence de la ligne exécutée dans la zone de l'éditeur de code (voir Figure 5-24-e). Cela doit aider les étudiants à faire le lien entre le code en cours d'exécution et l'action du personnage dans la scène.

Synthèse de la sous-section

Q5.3 : De quelles fonctionnalités doter l'environnement de l'application Pyrates afin de soutenir la transition Scratch-Python ?

Nous avons intégré dans l'environnement de l'application certaines caractéristiques des logiciels d'édition de blocs afin de pouvoir profiter des leurs avantages tout en programmant en Python :

- création d'un **mémo Python** qui **présente** de façon synthétique les **concepts** en jeu dans les différents niveaux à travers des explications, la production de **modèles génériques** d'implémentation, ainsi que des **exemples illustratifs**, ces programmes étant proposés dans les deux modalités **Scratch et Python** afin de faciliter la transition du point de vue sémiotique ;
- présence d'un **bouton copier** dans tous les extraits de code afin d'encourager la pratique du copier-coller et de limiter la saisie au clavier et les erreurs ;
- intégration d'un **analyseur syntaxique** issu de la recherche fournissant des messages d'erreur **adaptés aux débutants** (vocabulaire moins technique et en français) dans le but de faciliter la mise au point des programmes ;
- création d'un **panneau de contrôle** permettant de maitriser finement l'exécution (lancer, arrêter, changer de vitesse) et **mise en valeur de la ligne en cours d'exécution** afin d'aider les élèves à faire le lien entre programme et actions.

Nous venons de décrire l'aménagement d'environnement que nous avons effectué en nous inspirant des caractéristiques de la programmation par blocs. La sous-section qui vient présente l'évaluation de ces fonctionnalités.

5.9.2 Évaluation des fonctionnalités destinées à créer un lien Scratch-Python

L'objectif de cette sous-section est de répondre à la question de recherche Q5.4 à travers l'étude de l'appropriation des fonctionnalités destinées à créer un lien entre Scratch et Python. Pour ce faire, comme le décrit la méthodologie de ce chapitre (voir 5.5.2), nous nous appuyons sur les traces d'activités des 240 élèves ayant participé aux expérimentations dans les classes ainsi que sur l'enquête en ligne administrée à la fin de la dernière séance. Les traces d'utilisations considérées sont les suivantes : consultations du mémo, copiés-collés du mémo vers l'éditeur de code, erreurs détectées par l'analyseur syntaxique et par l'interpréteur, aides syntaxiques et sémantiques données par les enseignants lors de leurs interventions, manipulations du curseur de vitesse, et vitesse choisie lors de l'exécution des programmes.

Mémo Python

Pour commencer, examinons l'utilisation du mémo Python. La Figure 5-27 représente la durée moyenne en secondes de consultation par élève des différentes parties du mémo pour chaque niveau. La granularité est celle du sous-concept. Précisons que dans cette section nous prenons en compte tous les concepts visés par les niveaux et pas seulement les nouveaux concepts introduits comme nous le faisions dans la section précédente. Ces nouveaux concepts sont tout de même repérables étant les premiers de la liste.

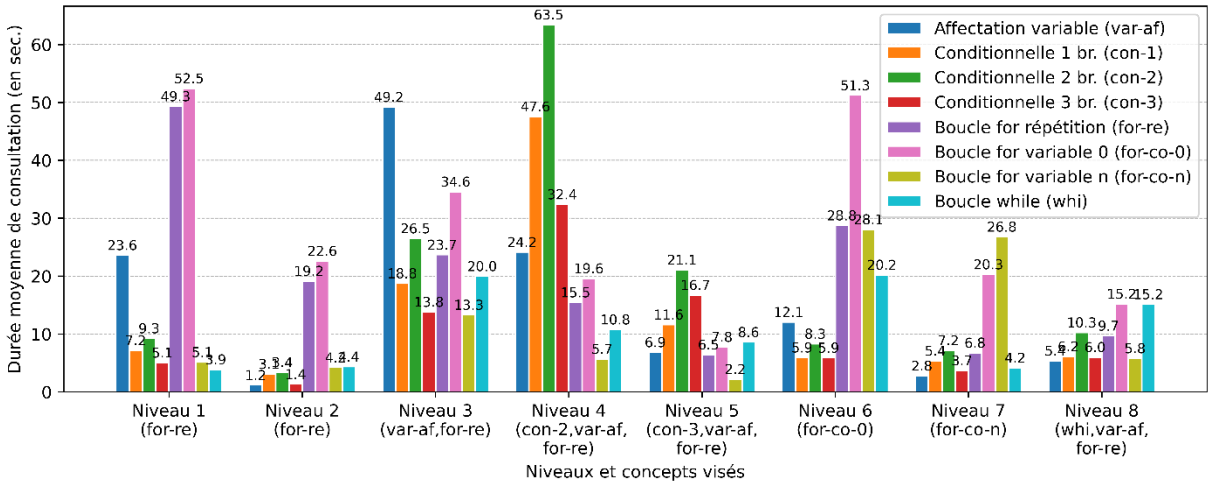
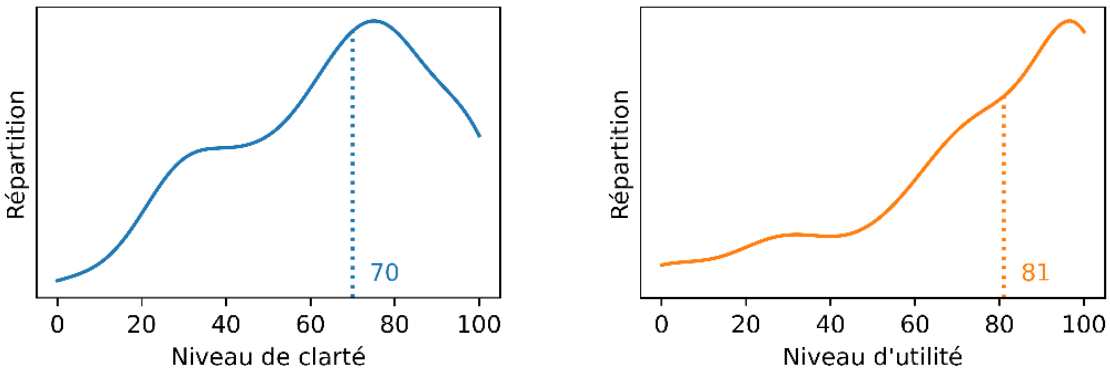


Figure 5-27 – Durée moyenne de consultation des différentes parties du mémo programmation par élève et par niveau

Nous pouvons d'abord remarquer que le temps de consultation moyen cumulé du mémo atteint entre 1,4 et 3,7 minutes par élève en fonction des niveaux. Cette durée moyenne représente entre 9,7% et 24,8% du temps de résolution disponible selon les niveaux (en se basant sur 15 minutes par niveau) ce qui est important. Nous pouvons également constater que le mémo est employé à bon escient car, excepté dans le niveau 1, les nouveaux concepts visés sont toujours les plus consultés.

Ensuite, comme pour le catalogue de blocs mis à disposition dans l'environnement Scratch (voir section 3.4.1), le mémo favorise la découverte des concepts. En effet, à chaque fois qu'un nouveau concept est impliqué dans un niveau sans être une déclinaison d'un concept déjà utilisé (niv.1, niv.3, niv.4, niv.6, et niv.8), nous retrouvons une plus grande variété dans les consultations. Cette diversité est la marque d'une démarche de recherche engagée par les élèves. En revanche, lorsque les concepts ont déjà été utilisés ou sont des variations de concepts déjà introduits (niv.2, niv.5, et niv.7), la consultation semble plus concentrée sur les concepts visés. Nous pouvons faire l'hypothèse que, dans cette situation, les apprenants ont besoin de retrouver la syntaxe du concept qu'ils souhaitent mettre en œuvre. Ce processus est assez similaire à la fonction de rappel du catalogue présent dans les environnements d'édition de blocs (voir section 3.4.1).

Les traces générées par l'application donnent un aperçu quantitatif de l'utilisation du mémo. Pour aller plus loin, ces analyses peuvent être complétées qualitativement par les résultats de l'enquête. Ainsi, les élèves devaient évaluer plusieurs aspects de l'application en plaçant des curseurs entre deux extrêmes (« Pas clair » - « Très clair », « Pas utile » - « Très utile », etc.), ce qui a eu pour effet de générer des scores entre 0 et 100. L'enquête comprenait en particulier des questions relatives au mémo Python. La Figure 5-28 présente la distribution des scores et la médiane pour les réponses à ces questions.



(a) Clarté des explications du mémo

(b) Utilité des comparaisons Scratch-Python

Figure 5-28 – Extraits des résultats de l'enquête auprès des élèves concernant le mémo Python (répartition et médiane)

Il ressort de cette figure que les explications contenues dans le mémo sont considérées comme claires par la majorité des élèves (voir Figure 5-28-a). Malgré cela, on peut distinguer un groupe d'apprenants autour du score de 30 pour qui ces contenus sont plus confus. De plus, les comparaisons avec Scratch proposées dans les modèles et les exemples sont jugées utiles voire très utiles par la grande majorité des élèves (voir Figure 5-28-b).

Copiés-collés

Concernant les copiés-collés, la Figure 5-29 indique leur nombre moyen par élève en fonction des concepts et des niveaux. La granularité est celle du sous-concept.

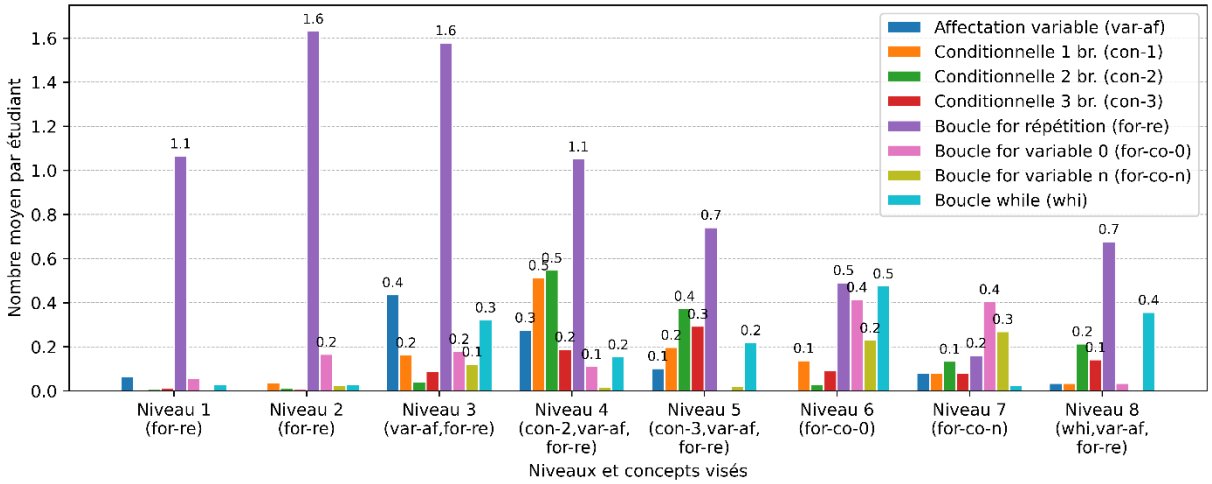


Figure 5-29 – Nombre moyen de copiés-collés du mémo Python par élève et par niveau

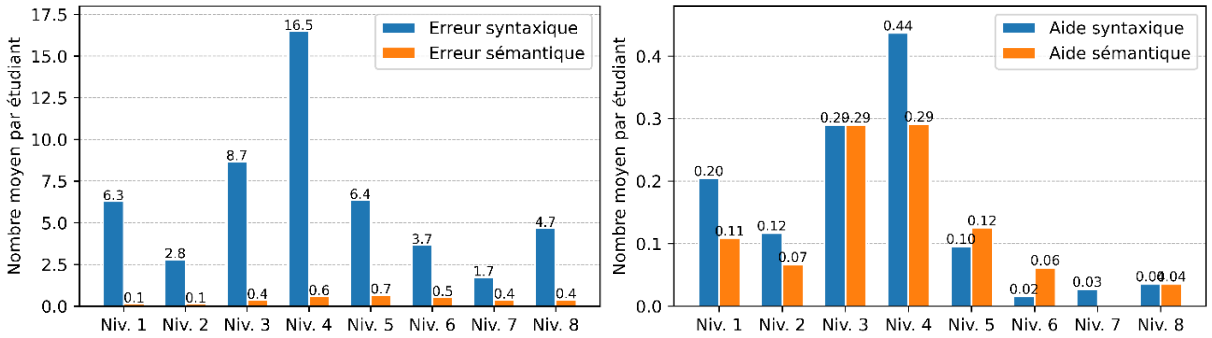
Cette figure montre que les élèves utilisent fréquemment la fonctionnalité copier-coller pour mettre en œuvre un concept. Ainsi, chaque fois qu'un concept est ciblé dans un niveau, il est copié-collé en moyenne entre 0,1 et 1,6 fois. Nous remarquons que ce nombre varie en fonction de la complexité de la syntaxe d'implémentation des concepts. Ainsi, l'affectation de variable (var-af), nécessitant un simple « = », fait l'objet de peu de copies (entre 0,4 et 0,1). À l'inverse, la boucle for répétition (for-re), dont la syntaxe est plus sophistiquée, donne lieu à en moyenne entre 0,7 à 1,6 copiés-collés selon les niveaux.

Notons également que, pour un même concept, le nombre moyen de copiés-collés diminue d'un niveau à l'autre. On peut avancer que de plus en plus d'élèves parviennent à mémoriser la syntaxe des concepts après plusieurs utilisations.

Au final, cette pratique du copier-coller est en quelque sorte l'équivalent du glisser-déposer des blocs. Elle permet de limiter la saisie au clavier et, dans une certaine mesure, d'aider à établir la structure des programmes.

Erreurs de programmation

En ce qui concerne les erreurs de programmation, la Figure 5-30 présente les erreurs détectées par l'application ainsi que les aides des enseignants relatives à la syntaxe et à la sémantique des programmes.



(a) Erreurs détectées par l'analyseur syntaxique et l'interpréteur Python

(b) Aides reçues par les élèves de la part des enseignants

Figure 5-30 – Erreurs détectées par l'application et aides des enseignants

L'examen de la Figure 5-30-a montre que les erreurs syntaxiques (provenant de l'analyseur syntaxique) sont nombreuses et dans une proportion beaucoup plus élevée que les erreurs sémantiques (provenant de l'interpréteur Python). Ceci est cohérent avec les résultats d'Altadmri et Brown (2015) qui avancent que, lors de l'apprentissage de la programmation informatique, les étudiants ont d'abord des problèmes de syntaxe avant de commencer à se confronter à des erreurs de type et de sémantique. Si l'on considère les aides apportées par les enseignants (voir Figure 5-30-b), il est intéressant de noter la faible fréquence des interventions en lien avec la syntaxe par rapport au nombre d'erreurs du même type. En effet, il y a une intervention pour 30 à 40 erreurs syntaxiques dans les quatre premiers niveaux. Nous pouvons donc supposer que les élèves sont capables d'ajuster leurs programmes erronés grâce aux messages d'erreurs proposés, et sans mobiliser les enseignants. Cela peut sans doute être mis au crédit des messages améliorés fournis par l'analyseur syntaxique que nous avons intégré. Néanmoins, il est important de garder à l'esprit que, d'une manière générale, les erreurs syntaxiques sont plus faciles à corriger que les erreurs sémantiques (Altadmri & Brown, 2015).

Enfin, nous avions demandé aux élèves lors de l'enquête de fin d'expérimentation d'évaluer la clarté des messages d'erreur. Les résultats présentés dans la Figure 5-31 montrent qu'ils sont jugés clairs par le plus grand nombre des répondants.

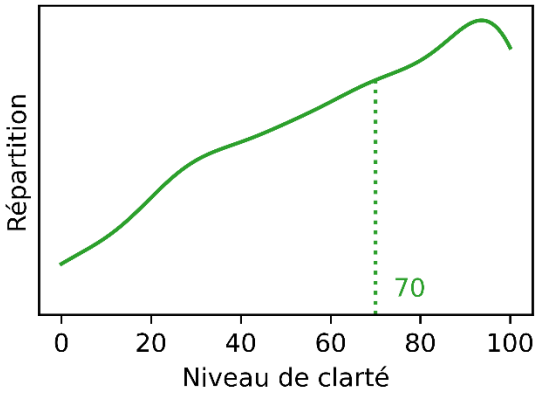


Figure 5-31 – Extraits des résultats de l'enquête auprès des élèves concernant les messages d'erreur (répartition et médiane)

Contrôle de l'exécution

Évaluons maintenant l'utilisation du panneau de contrôle des programmes. La Figure 5-32 présente le nombre moyen d'actions sur les programmes par élève et par niveau. Dans cette figure, un programme est considéré comme *correct* s'il ne contient pas d'erreurs syntaxiques ou sémantiques (voir section 3.3.4). Cela ne signifie pas qu'il permet forcément de gagner le niveau mais simplement qu'il s'exécute. Dans le cas contraire, il est identifié comme *erroné*. Un programme est *lancé* dès lors qu'un élève appuie sur le bouton « Exécuter » et quel que soit son issue. Il est *arrêté* si pendant l'exécution, le même élève clique sur le bouton « Arrêter ».

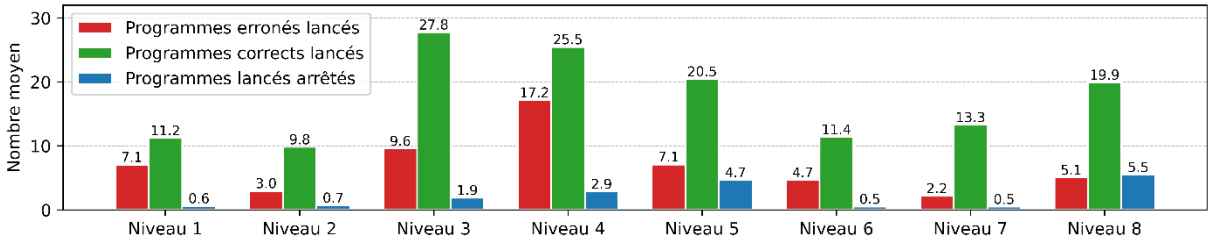


Figure 5-32 – Données concernant les actions des élèves sur les programmes

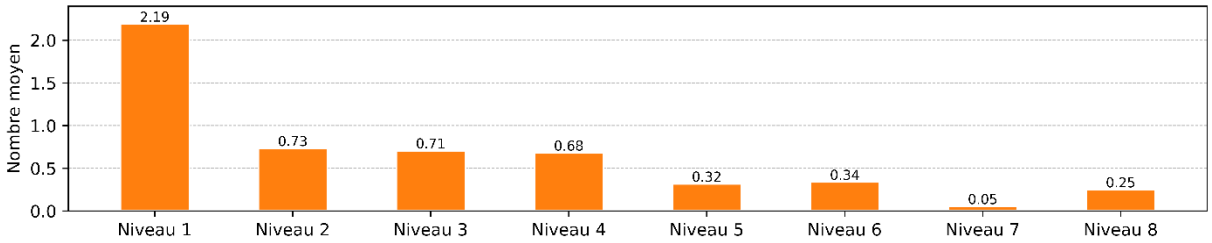
Nous constatons la présence d'un très grand nombre de programmes lancés en moyenne par élève. Beaucoup d'entre eux sont erronés, ce qui suggère que les élèves adoptent une approche de programmation par essais-erreurs afin d'obtenir la correction syntaxique de leur programme. De plus, de nombreux programmes corrects sont également lancés, ce qui montre que les joueurs progressent à travers les niveaux par étapes intermédiaires incrémentales jusqu'à l'objectif final d'ouverture du coffre. Ce mode de programmation (manière dont l'élève procède pour construire son programme) incrémental et par essais-erreurs pourrait s'expliquer par des habitudes héritées de l'apprentissage de la programmation en Scratch. En effet, Meerbaum-Salant et ses collègues (2011) ont découvert que les apprenants assimilaient certains comportements de programmation particuliers lorsqu'ils apprenaient à programmer avec des blocs. Ainsi, ils ont tendance à adopter une approche de programmation totalement ascendante :

> When faced with a programming task, the students did not approach it by thinking on the algorithmic level and not even on the level of software design. Instead, they began to solve a problem by dragging all the blocks that seemed to be appropriate for solving the task, and then combining them into a script. (Meerbaum-Salant et al., 2011, p. 169)

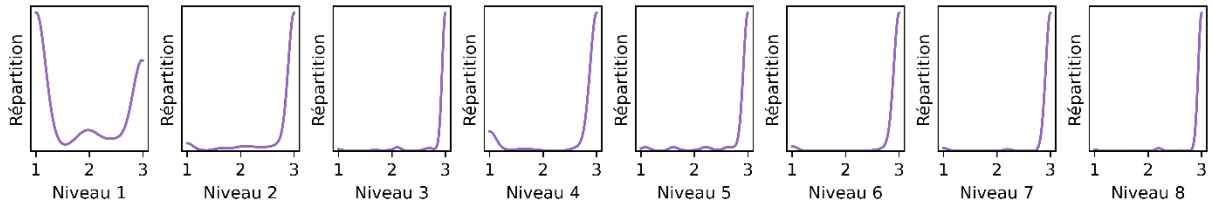
Cependant, Edwards (2004) affirme que les débutants en informatique réussissent mieux à apprendre s'ils passent d'une approche par essais-erreurs à une pratique de « réflexion en action ». Il serait ainsi avantageux de modifier les possibilités de contrôle de l'exécution dans notre application de manière à contraindre les élèves à moins d'actions et à plus de réflexion. Un moyen pourrait être de limiter le nombre d'exécutions à l'aide de pénalités.

L'utilisation du bouton d'arrêt pour interrompre l'exécution d'un programme est rare. Néanmoins, il est possible de distinguer deux types de comportements en fonction de la manière dont les maps des niveaux sont conçues. Pour un premier groupe de niveaux ayant des maps fixes non aléatoires (niv.1, niv.2, niv.6, et niv.7), les élèves utilisent en moyenne entre 15 et 20 lancements et presque aucun arrêt. Dans les niveaux contenant des maps aléatoires qui changent avant chaque exécution (niv.3, niv.4, niv.5 et niv.8), les élèves ont tendance à utiliser plus de lancements et à en arrêter un certain nombre. Comme nous l'avons déjà évoqué (voir section 5.8.2), pour ces niveaux générés aléatoirement, certains élèves adoptent des procédures d'évitement didactique consistant en une série de lancements et d'arrêts jusqu'à ce qu'ils obtiennent une configuration de map aléatoire adaptée à leur programme.

Enfin, prêtons attention au curseur de changement de vitesse d'exécution. La Figure 5-33 présente les données concernant la vitesse d'exécution des programmes.



(a) Nombre moyen de changements de vitesse d'exécution des programmes par élève



(b) Répartition des vitesses d'exécution des programmes par niveau

Figure 5-33 – Données concernant la vitesse d'exécution des programmes

La Figure 5-33-a indique que le curseur de vitesse est en moyenne peu utilisé et de manière décroissante dans le temps. La Figure 5-33-b donne la répartition des vitesses d'exécution des programmes lancés pour chaque niveau. Nous pouvons constater qu'à partir du niveau 2, les programmes sont quasiment tous lancés à la vitesse maximale (facteur multiplicateur de 3). L'approche de programmation incrémentale et par essais-erreurs décrite précédemment nécessite cette vitesse d'exécution élevée afin de pouvoir enchainer les essais sans délais. Ajoutons à ce sujet que trois élèves ont fait remarquer dans le champ ouvert de l'enquête que « le personnage ne bouge pas assez vite ». Néanmoins, une pratique marginale a pu être constatée dans les niveaux plus avancés (niveau 4 et niveau 5). Elle consiste à revenir à des vitesses d'exécution plus lentes. En effet, les observations faites lors des expérimentations indiquent que certains élèves peuvent avoir besoin de suivre plus facilement les lignes exécutées dans un mode d'exécution se rapprochant du pas à pas.

Synthèse de la sous-section

Q5.4 : Sur le terrain, comment les élèves interagissent-ils avec les fonctionnalités de l'environnement ? Quel usage en font-ils ? Comment les jugent-ils en termes de clarté et d'utilité ?

Suite aux analyses des traces d'activités et des réponses au questionnaire collectées lors de notre expérimentation, nous pouvons avancer les éléments suivants. Certains choix de conception ont des conséquences positives :

- le **mémo** Python est **très fréquemment consulté** par les élèves, il est le **support** de la **découverte** et de la **remémoration** des concepts ;
- les **traductions Scratch-Python** incluses dans le mémo sont considérées comme « **utiles** » par une grande majorité d'élèves, elles doivent **accompagner** la **transition** du **registre sémiotique** des blocs à celui des instructions ;

- le **copier-coller** à partir du mémo de programmation est **largement pratiqué**, ce qui a pour effet de **limiter** la **saisie** au clavier ;
- les rétroactions fournies par l'**analyseur syntaxique** via des messages d'erreur jugés comme « **clairs** » permettent aux élèves de **corriger** leurs programmes en **autonomie** avec peu d'intervention des enseignants.

Le panneau de contrôle devait permettre aux élèves de mieux appréhender l'exécution des programmes. Cependant, il ne produit pas les résultats escomptés :

- le **bouton de lancement** des programmes est fréquemment utilisé et le **curseur de contrôle** de la vitesse est réglé très tôt au maximum afin de faciliter un **mode de programmation incrémental** et par **essais-erreurs** hérité de l'utilisation de Scratch ;
- le **bouton** permettant d'**arrêter** les exécutions est peu utilisé, et lorsqu'il l'est, c'est surtout pour tenter de réussir certains niveaux basés sur le hasard en utilisant des procédures de **contournement didactique**.

Après avoir exposé nos résultats concernant les différents aspects de la conception de Pyrates, nous donnons dans la section suivante quelques éléments concernant la perception générale de l'application pour les élèves.

5.10 Perception globale de l'application

Nous présentons ici les éléments de réponse à la question de recherche Q5.5 portant sur la perception globale des utilisateurs de Pyrates concernant les divers aspects que sont : la prise en main, la difficulté, le caractère ludique et la motivation.

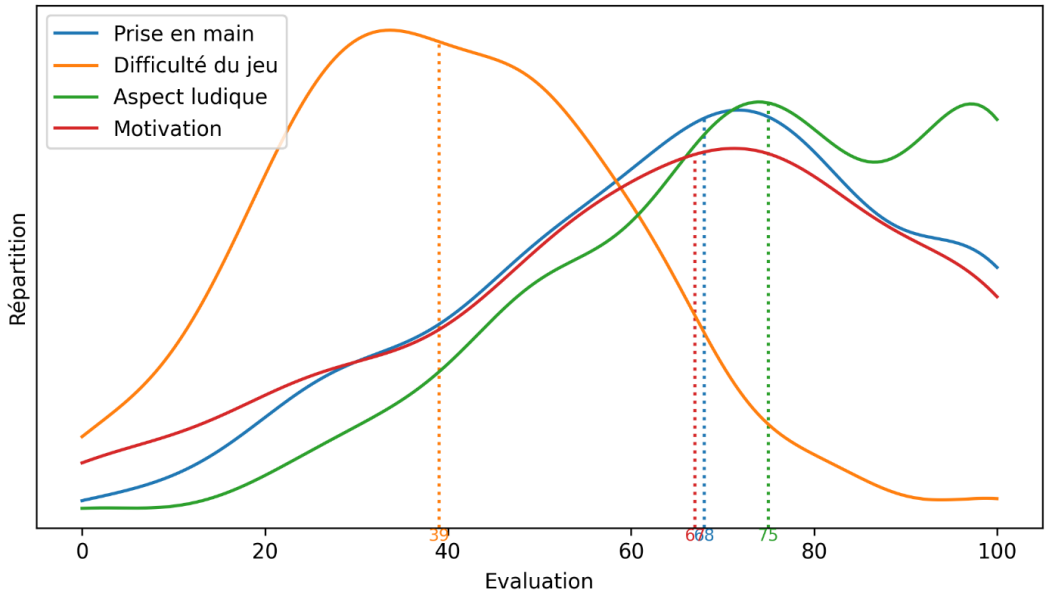


Figure 5-34 – Évaluation des joueurs concernant différents aspects du jeu (répartition et médiane)

La question Q8 de l'enquête de fin d'expérimentation portait sur la perception globale de l'application : « Évaluer chaque aspect de votre expérience : Prise en main de l'application (difficile – facile) ; Difficulté du jeu (difficile – facile) ; Aspect ludique (ennuyeux-divertissant) ; Motivation (décourageant – motivant) » (Annexe C.2). Différents curseurs permettaient aux élèves de se positionner entre les deux extrémités proposées et ont généré des scores entre 0 et 100. La Figure 5-34 regroupe les réponses à cette question sous la forme de courbes de répartitions.

5.10.1 Prise en main

Commençons par remarquer que la prise en main de l'application est jugée plutôt facile (médiane 68), cela peut être expliqué par la présence d'un guide de démarrage associé à la rapide présentation du chercheur en début d'expérimentation.

5.10.2 Difficulté

Ensuite, le jeu est globalement considéré comme difficile par les élèves (médiane 39). Comme nous le verrons dans le chapitre suivant, cela est cohérent avec le fait que peu d'élèves ont eu la possibilité de terminer les huit niveaux et seuls 60% d'entre eux ont atteint au moins le niveau 5.

5.10.3 Motivation

Malgré la difficulté ressentie par les élèves, il est étonnant de constater que leur motivation n'est de façon générale pas entamée (médiane 67). En effet peu d'élèves se sentent découragés à l'issue de l'expérimentation. Nous retrouvons ici les résultats de recherche présentés en début de chapitre portant sur les bénéfices de la ludification sur la motivation des apprenants (Zhan et al., 2022).

5.10.4 Aspects ludiques

Enfin, l'expérience vécue est globalement qualifiée de divertissante (médiane 75), avec un groupe d'élèves ayant placé le curseur au maximum. Ce résultat prévisible fait également partie des bénéfices démontrés des jeux sérieux (Vahldick et al., 2014).

5.10.5 Synthèse de la section

Q5.5 : Quelle est la perception globale de l'application par les élèves concernant la prise en main, la difficulté, l'aspect ludique et la motivation qu'elle engendre ?

En analysant les réponses au questionnaire proposé en fin d'expérimentation nous retenons que globalement :

- la **prise en main** de l'application est jugée **facile**;
- le **jeu** est perçu comme **difficile** ;
- les **élèves** se déclarent **motivés** ;
- l'**expérience** de jeu est estimée **ludique**.

Après avoir évoqué la perception générale de l'application Pyrates par les élèves engagés dans l'expérimentation, nous exposons dans la dernière section sa diffusion dans les classes.

5.11 Diffusion de l'application

Cette dernière section relate la diffusion de Pyrates et son appropriation par les utilisateurs finaux, c'est-à-dire les enseignants et les élèves en France et à l'étranger.

La première version de l'application, dont la conception a été évaluée dans ce chapitre, a été diffusée auprès des enseignants par courriel le 12/10/21 (voir Annexe J.3). Nous avons pour cela utilisé les listes RENATER de mathématiques, SNT et NSI auxquelles nous avions eu recours pour notre enquête auprès des enseignants. Nous avons également fait connaitre l'application en publiant une présentation sur le forum national des enseignants de NSI (*Forum NSI home page*, 2023) et sur notre compte Twitter.

Au moment de la rédaction de cette thèse et depuis sa mise en ligne le 07/09/21, l'application Pyrates a enregistré 83 842 parties jouées[34]. La Figure 5-35 donne la répartition des parties par jour.

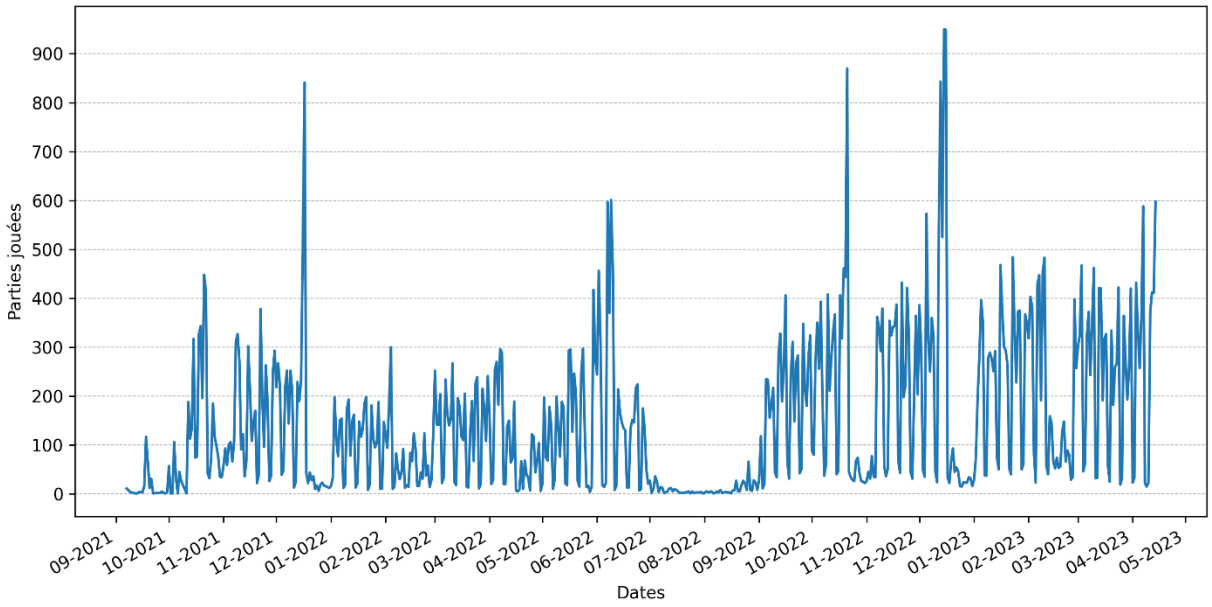


Figure 5-35 – Parties du jeu Pyrates jouées depuis sa mise en ligne

Même si nous n'avons pas d'informations précises sur le contexte dans lequel est utilisée l'application, la répartition cyclique des parties dans le temps ne laisse que peu de doute sur son caractère scolaire. Nous pouvons en effet observer que les périodes de week-end et de vacances scolaires connaissent de forte baisse d'affluence. Les affluences moyennes permettent de confirmer cette hypothèse : la moyenne journalière des parties en période scolaire (en semaine hors vacances) est de 265,11 et la moyenne hors période scolaire est de 80,28.

[34] Nous comptabilisons une partie si le joueur parvient au moins à terminer le premier niveau

Nous pouvons également observer des pics d'activité les jours précédant les vacances scolaires, ce qui laisse penser que certains enseignants utilisent l'application pour son caractère ludique.

Ajoutons que la dynamique d'utilisation de l'application est croissante. Alors que le nombre moyen de parties jouées quotidiennement sur le temps scolaire durant l'année scolaire 21-22 est de 203,11, il est de 335,66 pour l'année scolaire 22-23, ce qui représente un augmentation de 65%. De plus, l'affluence semble plus stable sur cette deuxième année d'utilisation.

Enfin, évoquons le choix de la langue de l'application. La Figure 5-36 donne la proportion des langues choisies lors de l'utilisation de Pyrates depuis sa mise en ligne.

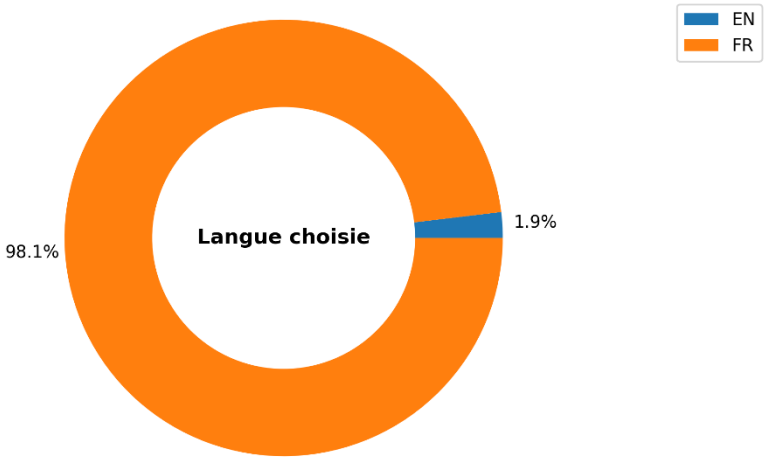


Figure 5-36 – Proportion des langues choisies lors de l'utilisation de l'application Pyrates depuis sa mise en ligne

Le français domine très largement les usages avec 98% des parties jouées. Néanmoins, 2% des parties jouées sont en anglais, ce qui représente tout de même 1 589 parties. La diffusion de l'application dans les pays non-francophones est un potentiel à exploiter pour la suite dans la mesure où la problématique de transition blocs-texte est partagée dans de nombreux pays à travers le monde bien au-delà du contexte de transition collège-lycée en France qui est l'objet de cette thèse.

Pour terminer, nous avons déposé le code de Pyrates le 24/01/22 auprès de l'Agence pour la Protection des Programmes (voir certificat en Annexe K) afin d'en attester l'existence à cette date. Nous avons pour objectif de distribuer le logiciel sous licence libre GPLv3, en ouvrant l'accès au dépôt GitHub du projet, dans le but de permettre à la communauté de recherche autour des EIAH ainsi qu'aux contributeurs potentiels de s'en emparer.

Dans ce chapitre, nous avons présenté la conception d'une ressource d'enseignement de la programmation à la transition collège-lycée. Elle prend la forme d'une application en ligne nommée Pyrates et intégrant un serious game permettant de contrôler un avatar dans un jeu de plateforme au moyen d'un programme Python. Les niveaux du jeu ont été conçus sur le modèle des situations adidactiques dans le but d'accompagner l'appropriation en Python des concepts fondamentaux de la programmation enseignés en Scratch au collège. De plus, l'application comprend un environnement d'édition s'inspirant des éditeurs de blocs afin de bénéficier de leurs

avantages tout en programmant dans un langage textuel. La prise en main de l'application est considérée comme bonne par les élèves qui se déclarent motivés et divertis même si le jeu est perçu comme difficile. Enfin, notre ressource a été largement diffusée dans les classes pendant deux années scolaires et atteint au moment de l'écriture de ces lignes plus de 80 000 parties jouées en France et 1500 à l'international.

Nous présentons dans le chapitre suivant une évolution de Pyrates visant à améliorer l'autonomie des élèves en leur fournissant des rétroactions automatisées en fonction de leur activité dans l'application. Ces rétroactions sont déclenchées par les élèves et choisies par des modèles d'intelligence artificielle entrainés sur de nouvelles données récoltées lors d'une deuxième expérimentation sur le terrain.

Chapitre 6

Rétroactions épistémiques visant à favoriser l'autonomie des élèves

Dans les cinq premiers chapitres de cette thèse, nous avons suivi la méthodologie de l'ingénierie didactique et réalisé une itération complète allant des études préalables à l'analyse des traces issues d'une expérimentation sur le terrain. Nous avons ainsi eu l'occasion d'évaluer la conception des différents niveaux du jeu Pyrates, son environnement d'édition, ainsi que sa perception globale par les élèves.

Cependant, nous avions indiqué dans le Chapitre 4 que notre ressource devait être conçue pour une durée optimale de deux heures (10 séances annuelles sont consacrées en moyenne à la programmation informatique en seconde), et devait favoriser l'autonomie des élèves dans un contexte de classes chargées au lycée (31,7 élèves par classe en moyenne en seconde).

Dans ce sixième chapitre, nous commençons donc par évaluer l'avancée des élèves après deux séances de jeu ainsi que leur autonomie vis-à-vis des enseignants. Nous proposons ensuite une évolution de l'application visant à accroitre cette autonomie par l'intermédiaire d'aides fournies automatiquement en fonction de l'activité des élèves dans l'environnement. Ces aides prennent la forme de *rétroactions*, définies par Shute (2008) comme des informations communiquées à l'apprenant et destinées à modifier sa pensée ou son comportement dans le but d'améliorer son apprentissage : « Formative feedback is defined [...] as information communicated to the learner that is intended to modify his or her thinking or behavior for the purpose of improving learning » (2008, p. 154). À la suite de Luengo (2009), nous qualifions plus précisément ces aides de *rétroactions épistémiques* dans la mesure où nous verrons qu'elles sont « relative[s] à la connaissance en jeu de l'activité. » (2009, p. 14). Les rétroactions épistémiques que nous proposons sont déclenchées par des modèles d'Intelligence Artificielle (IA) entrainés à partir de nouvelles données que nous avons collectées lors d'une deuxième expérimentation sur le terrain.

Ainsi, la section 6.1 expose d'abord l'état de l'art concernant les rétroactions dans les environnements d'apprentissage de la programmation, suivie par les sections 6.2 et 6.3 qui présentent les questions de recherche et la méthodologie que nous employons dans ce chapitre. Ensuite, la section 6.4 détaille l'évaluation de l'avancement des élèves dans le jeu ainsi que leur autonomie. Enfin, la section 6.5 relate la conception des différentes rétroactions épistémiques, puis la section 6.6 décrit le mécanisme de sélection de ces rétroactions.

6.1 État de l'art

Dans cette section, nous présentons un ensemble de travaux relatifs aux rétroactions dans les environnements d'apprentissage de la programmation informatique. Nous décrivons d'abord nos critères de sélection des articles, puis notre

grille d'analyse, avant de considérer le contexte pédagogique et les rétroactions proposées par les différentes recherches sélectionnées.

6.1.1 Sélection des travaux et grille d'analyse

Nous avons recherché dans la littérature des contributions qui répondent aux critères suivants : les rétroactions sont fournies lors d'une tâche de programmation (indépendamment du paradigme et de la modalité blocs ou texte) ; la génération des rétroactions est guidée par les données des apprenants (data-driven) avec ou sans l'intervention d'experts du domaine (expert-driven).

Afin d'analyser ces travaux, nous considérons d'abord le contexte pédagogique des études réalisées, puis les éléments relatifs aux rétroactions elles-mêmes.

Concernant le contexte, nous examinons :

- le *niveau* scolaire des apprenants : primaire (PR), secondaire (SE), ou supérieur (SU) ;
- le *langage* de programmation ciblé : langage (C++, Python) ou type de langage (logique, blocs) visé ;
- le type d'activités proposées et plus particulièrement le fait d'être ou non un *serious game* (SG).

Dans le but de caractériser les rétroactions proposées dans les travaux que nous analysons, nous avons adapté deux cadres d'analyse de rétroactions existants : le « Technologies for Automated Feedback Classification Framework » (TAF-ClaF) introduit par Deeva et ses collègues (2021), que nous avons articulé avec l'« Interactive Tutoring Feedback model » (ITF-model) proposé par Narciss (2013). En effet, le premier cadre est plus complet mais le deuxième offre une granularité plus fine concernant la description du contenu des rétroactions. Nous avons donc sélectionné les éléments suivants comme étant les plus pertinents pour notre analyse des rétroactions :

- le *moment* de déclenchement de la rétroaction (Deeva et al., 2021) : immédiat (IM), proposé (PR), à la demande (AD), à la fin de la tâche (FT) ;
- le *contenu* de la rétroaction (Narciss, 2013) : métacognition (MT), règle de la tâche (RT), concept (CO), indication procédurale (IP), réponse correcte (RC), erreur (ER), performance (PE), ou résultat (RE) ;
- les *données de l'élève* utilisées pour la génération des rétroactions (Deeva et al., 2021) : état du système suite aux actions de l'élève (ET) (par exemple l'emplacement des entités dans la map d'un jeu, ou la valeur des variables d'un programme), réponse ouverte de l'élève (RO) (par exemple les programmes soumis), comportement dans le système (CS) (par exemple les traces d'activités de l'élève), comportement dans la vie réelle (VR) (par exemple les mouvements du regard ou l'expression faciale de l'élève), caractéristiques de l'élève (CA) (par exemple les compétences acquises ou l'état affectif) ;
- les *connaissances expertes* utilisées pour la génération des rétroactions (Deeva et al., 2021) : solution et réponse correcte (SO), erreur typique (ER), modèle de rétroaction (MO), et règle de déclenchement (RE).

Le Tableau 6-1 décrit les caractéristiques relatives au contexte et aux rétroactions pour les travaux que nous avons sélectionnés. Pour des raisons de place, nous ne donnons que le premier auteur des articles.

	Contexte			Rétroactions			
Travaux	**Niv.**	**Langage**	**SG**	**Mo.**	**Contenu**	**Données**	**Exp.**
Merceron et al. (2005)	SU	Logique	Non	IM	CO, ER	RO, CS	MO
Barnes et al. (2010)	SU	Logique	Non	PR	IP	RO	∅
Hicks et al. (2014)	SE	Blocs	Oui	?	IP	ET	∅
Fossati et al. (2015)	SU	C++	Non	IM	IP, ER, RE	RO	∅
Piech et al. (2015)	SE	Blocs	Oui	?	CO, IP, RE, RC	ET, RO	MO
Price et al. (2016)	SU	Blocs	Non	IM	IP	RO	∅
Rivers et al. (2017)	SU	Python	Non	AD	IP	RO	∅
Chow et al. (2017)	SE	Python	Non	IM	CO, IP, ER	RO, CS	∅
Price et al. (2017)	SU	Blocs	Non	AD	IP	RO	∅
Head et al. (2017)	SU	Python	Non	?	CO, IP, ER, RC	RO	MO
Marwan et al. (2022)	SU	Blocs	Non	IM	PE	RO	∅

Tableau 6-1 – Caractéristiques des travaux analysées relatives au contexte et aux rétroactions, « ? » indique une information non disponible

Nous analysons à la suite les aspects liés au contexte et aux rétroactions.

6.1.2 Analyse du contexte pédagogique

Nous constatons d'abord dans le Tableau 6-1 que pour huit études sur onze, le niveau scolaire visé est celui de l'enseignement supérieur. Seules trois contributions portent sur le secondaire (Chow et al., 2017; Hicks et al., 2014; Piech et al., 2015). Ensuite, trois propositions sont centrées sur les rétroactions dans le cadre de l'enseignement du langage Python (Chow et al., 2017; Head et al., 2017; Rivers & Koedinger, 2017). Cependant, elles concernent toutes des plates-formes d'apprentissage en ligne basées sur des exercices de programmation qui sont des environnements moins ouverts que les jeux sérieux. Enfin, deux articles parmi ceux sélectionnés portent sur des jeux sérieux (Hicks et al., 2014; Piech et al., 2015), il s'agit cependant d'enseigner la programmation par blocs.

Nous pouvons donc avancer que, dans la limite de nos connaissances, aucune plateforme ayant fait l'objet d'une publication scientifique ne propose actuellement de rétroactions au service de l'apprentissage du langage Python dans le cadre d'un jeu sérieux comme nous projetons de le faire.

6.1.3 Analyse des rétroactions

Concernant les rétroactions elles-mêmes, nous présentons ici les éléments relatifs au moment du déclenchement, à leur contenu, ainsi qu'à la méthode de génération (données des élèves et interventions éventuelles des experts).

Moment du déclenchement

Nous pouvons tout d'abord remarquer que, dans la plupart des solutions étudiées, les rétroactions sont déclenchées en temps réel (IM), par le biais d'une fenêtre contextuelle ou d'un texte affiché. La recherche montre cependant que cette apparition instantanée peut perturber les élèves et entraver leurs apprentissages (Shute, 2008). Deux contributions offrent des rétroactions à la demande (AD), c'est-à-dire lorsque les élèves cliquent sur un bouton de demande d'aide (Price et al., 2017; Rivers & Koedinger, 2017). Notons qu'il existe une option intermédiaire, consistant à activer un bouton de demande d'aide lorsqu'une rétroaction est disponible (PR). Pour certains travaux (Head et al., 2017; Hicks et al., 2014; Piech et al., 2015), l'information du moment de déclenchement n'est pas disponible compte tenu du fait que le déploiement effectif des rétroactions n'est pas décrit.

Contenu des rétroactions

Ensuite, à propos du contenu des rétroactions, la quasi-totalité des études fournissent des informations procédurales (IP) indiquant comment procéder dans la tâche, en fournissant le code à écrire à l'étape suivante (next step hint) ou des directives indiquant des modifications à réaliser dans les programmes. Il peut également s'agir d'informations relatives aux erreurs de codage (ER) (Chow et al., 2017; Head et al., 2017; Merceron & Yacef, 2005), à la performance globale (PE), qui prend ici la forme de barres d'avancement (Marwan et al., 2022), ou à la validité du programme soumis (RE) (Fossati et al., 2015; Piech et al., 2015). Quelques recherches donnent des indications au sujet des concepts qui doivent être mis en œuvre (CO) (Chow et al., 2017; Piech et al., 2015) ou pointent vers des éléments de cours y faisant référence (Head et al., 2017; Merceron & Yacef, 2005). Remarquons qu'aucune étude ne propose de rétroactions de plus haut niveau concernant la tâche à accomplir (RT) ou la métacognition (MT).

Comme l'avance Shute (2008), les apprenants les moins performants ont besoin d'une forme de rétroactions concrètes et directives (IP) qui peuvent aller jusqu'à donner la bonne réponse (RC). En outre, il est également important de proposer des rétroactions de plus haut niveau en précisant par exemple les objectifs de la tâche à accomplir (TR) afin d'améliorer la motivation et l'engagement des apprenants (Shute, 2008). Ainsi en suivant cette auteure, l'idéal serait de pouvoir fournir un éventail de rétroactions progressives allant de l'énoncé de la tâche à sa solution complète.

Génération avec données des apprenants seuls

Dans un troisième temps, intéressons-nous au processus de génération des rétroactions. Au regard de nos critères de sélection, le mode de génération s'appuie toujours sur les données des apprenants dans les travaux analysés. Néanmoins, certains travaux se basent uniquement sur ces données afin de générer les rétroactions, sans intervention d'experts du domaine. Ces méthodes exploitent généralement le code des apprenants (RO). Cela permet, en analysant l'historique des programmes soumis, de prédire la prochaine étape de programmation (contenu IP) pour un programme donné. Il est ensuite possible de proposer aux utilisateurs des rétroactions contenant des modifications du programme courant ou un fragment de code menant vers la solution (next step hint) (Barnes & Stamper, 2010; Fossati et al., 2015; Price et al., 2016, 2017; Rivers & Koedinger, 2017). Dans le même ordre d'idées, Hicks et ses

collèges (2014) s'appuient sur l'historique de l'état des sprites dans la map d'un jeu sérieux (ET) afin de prédire la prochaine action à effectuer vers la solution. Enfin, Chow et ses collèges (2017) tirent profit, en plus des programmes, du comportement des apprenants dans le système (CS). Il s'agit ici de la durée des actions, et des entrées et sorties des programmes. Cela leur permet de caractériser le comportement des utilisateurs concernant le style d'édition, la vitesse et la cohérence de progression, ainsi que le comportement en matière d'exécution. Ces données contribuent à fournir une plus grande variété de rétroactions, notamment concernant les concepts (CO) et les erreurs (ER).

L'intérêt de ces techniques qui s'appuient uniquement sur les données historiques des apprenants est qu'elles sont flexibles car elles offrent une grande souplesse pour les modifications de contenu. Ainsi, lors de l'ajout d'exercices sur une plateforme ou de niveaux dans un jeu sérieux, le système pourra fournir des rétroactions dès lors qu'il dispose d'assez de données d'utilisations[35]. Cette approche est par conséquent économique en temps de développement. Elle souffre en contrepartie d'inconvénients. Ainsi, les contenus proposés dans les rétroactions dépendent de la qualité du code exécuté par les utilisateurs précédents. Par conséquent, ces programmes peuvent être fonctionnellement corrects mais conceptuellement pauvres, et donc potentiellement néfastes à l'apprentissage lorsqu'ils sont utilisés pour générer le contenu des rétroactions (Head et al., 2017).

Génération avec données des apprenants et des experts

L'autre méthode de génération fait appel à des experts du domaine de la programmation informatique[36] en complément des données issues des apprenants. Dans les travaux que nous analysons, ces experts ont la charge de concevoir des rétroactions (MO) en s'appuyant sur un prétraitement des données collectées auprès des apprenants afin de gagner en efficacité. Ainsi, dans une étude pionnière, Merceron et Yacef (2005) proposent aux enseignants de rédiger des rétroactions proactives concernant des enchaînements d'erreurs probables. Ces prédictions d'erreurs prennent la forme de règles d'associations issues de la fouille de données des erreurs commises précédemment dans les données historiques. Lors de l'apparition d'une erreur, une rétroaction anticipe les prochaines erreurs en proposant explications et pointeurs vers du contenu pédagogique. Les travaux présentés par Head et ses collègues (2017) ainsi que par Piech et ses collègues (2015) sollicitent des experts afin de concevoir des rétroactions concernant des programmes d'élèves qui ont été préalablement sélectionnés pour leur représentativité par des algorithmes de clustering. Ces annotations sont effectuées à postériori en laboratoire et pour un nombre réduit de situations. Elles sont ensuite propagées à tout nouveau programme à l'aide d'algorithmes d'apprentissage automatique.

Ces méthodes qui font intervenir des experts prennent du temps et possèdent un coût de développement important. En outre, elles sont moins flexibles que celles basées uniquement sur les données. Cependant, les rétroactions générées sont plus fiables car

[35] Ces méthodes étant basées sur des données historiques issues des apprenants, le problème du démarrage à froid (cold start problem) se pose inévitablement pour les premiers utilisateurs qui ne peuvent pas bénéficier de rétroactions.

[36] Ces experts peuvent être des chercheurs spécialisés dans le domaine de la programmation (des informaticiens) ou bien des praticiens de terrain (enseignants dans le niveau scolaire visé).

conçues par des experts, et plus riches et variées que les précédentes dans la mesure où les rédacteurs ont la liberté de se placer à n'importe quel niveau de contenus (CO, IP, ER, RE, RC).

Nous retenons, concernant les rétroactions, que la littérature propose un large éventail de possibilités concernant le moment de déclenchement, le contenu, ou le mode de génération des rétroactions. Chaque solution possédant ses avantages et ses inconvénients. Nous nous appuierons sur ces analyses lors des choix de conception de notre système de rétroactions dans Pyrates.

Suite à cette revue de littérature, la section suivante présente les questions de recherche et la méthodologie utilisée.

6.2 Questions de recherche

La question de recherche Q6 que nous étudions dans ce sixième chapitre est la suivante : Comment améliorer l'autonomie des élèves vis-à-vis des enseignants par le biais de rétroactions automatisées lors de l'utilisation de notre application ?

Elle se décline en plusieurs sous-questions :

- Q6.1 : Quelle est la progression des élèves dans le jeu et leur autonomie vis-à-vis de l'enseignant lors des deux séances de la première expérimentation ? Quelles pistes proposer pour les améliorer ?
- Q6.2 : Quelles rétroactions épistémiques concevoir dans l'objectif de soutenir l'autonomie des élèves qui utilisent Pyrates ?
- Q6.3 : Quelle rétroaction épistémique fournir aux élèves lorsqu'ils demandent de l'aide afin de préserver autant que possible l'adidacticité[37] des situations tout en leur permettant d'avancer ?

6.3 Méthodologie

La méthodologie employée dans ce chapitre varie en fonction des questions de recherche. Nous détaillons à la suite les détails propres à chacune d'elles.

6.3.1 Progression et autonomie

Afin d'évaluer la progression et l'autonomie des élèves dans Pyrates (Q6.1), nous reprenons les données issues de notre première expérimentation au cours de laquelle les élèves avaient pu utiliser l'application pendant deux séances. Nous analysons ainsi pour chaque niveau : le nombre de niveaux commencés et terminés, le temps moyen passé par élève, ainsi que le nombre moyen d'aides reçues par élève.

[37] Rappelons que dans une situation adidactique, c'est par la confrontation à un problème que l'élève construit de nouvelles connaissances. Il ne s'agit donc pas de communiquer la connaissance de façon descendante, mais de créer les conditions de l'apprentissage en provoquant chez l'élève les adaptations souhaitées.

6.3.2 Conception et sélection des rétroactions

L'élaboration des différentes rétroactions (Q6.2) et la mise au point du système de sélection (Q6.3) se basent sur plusieurs étapes d'expérimentation et de conception que nous résumons dans la Figure 6-1 et détaillons par la suite.

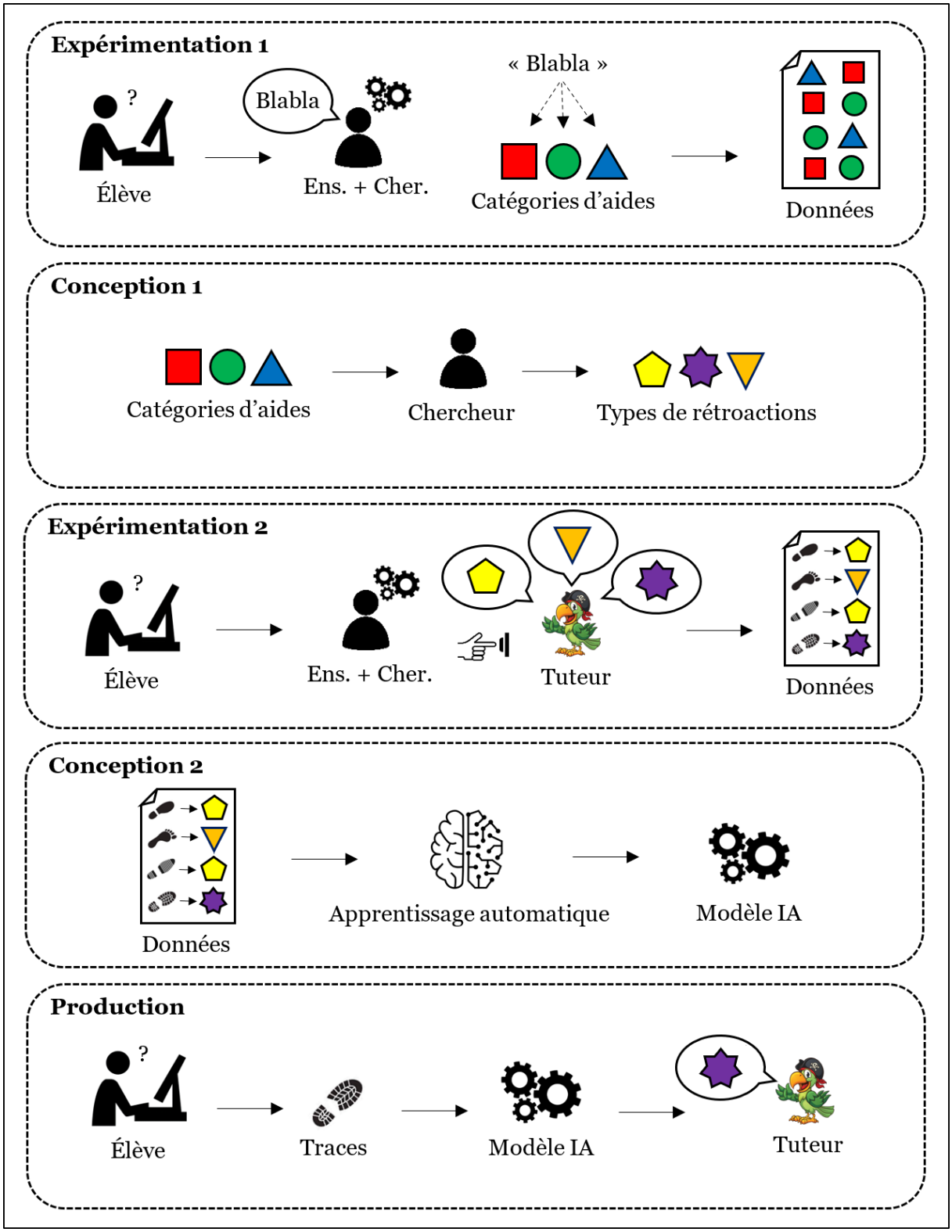


Figure 6-1 – Résumé des différentes phases de conception du système de rétroaction dans Pyrates

Expérimentation 1

Nous commençons donc par réutiliser les données de notre première expérimentation sur le terrain (voir section 5.5.2) au cours de laquelle l'enseignant de la classe et le chercheur ont fourni des aides sous forme orale aux élèves qui en avaient besoin. Après chaque intervention, ils avaient pour consigne de caractériser leurs propos en utilisant les boutons disponibles correspondant aux différentes catégories d'aides (application, jeu, syntaxe, sémantique, etc.).

Conception 1

Lors d'une première étape de conception, nous créons plusieurs types de rétroactions destinées à être délivrées automatiquement par un tuteur dans Pyrates (Q6.2). Nous nous appuyons pour cela sur les données issues de l'expérimentation initiale en privilégiant les catégories d'aides les plus fréquentes et celles compatibles avec une traduction sous forme de rétroaction. Nous implémentons ensuite ces différentes rétroactions dans Pyrates de telle manière qu'elles puissent être, dans un premier temps, déclenchées manuellement par les enseignants.

Comme l'illustre la Figure 6-2, nous modifions à cet effet le mode recherche de l'application.

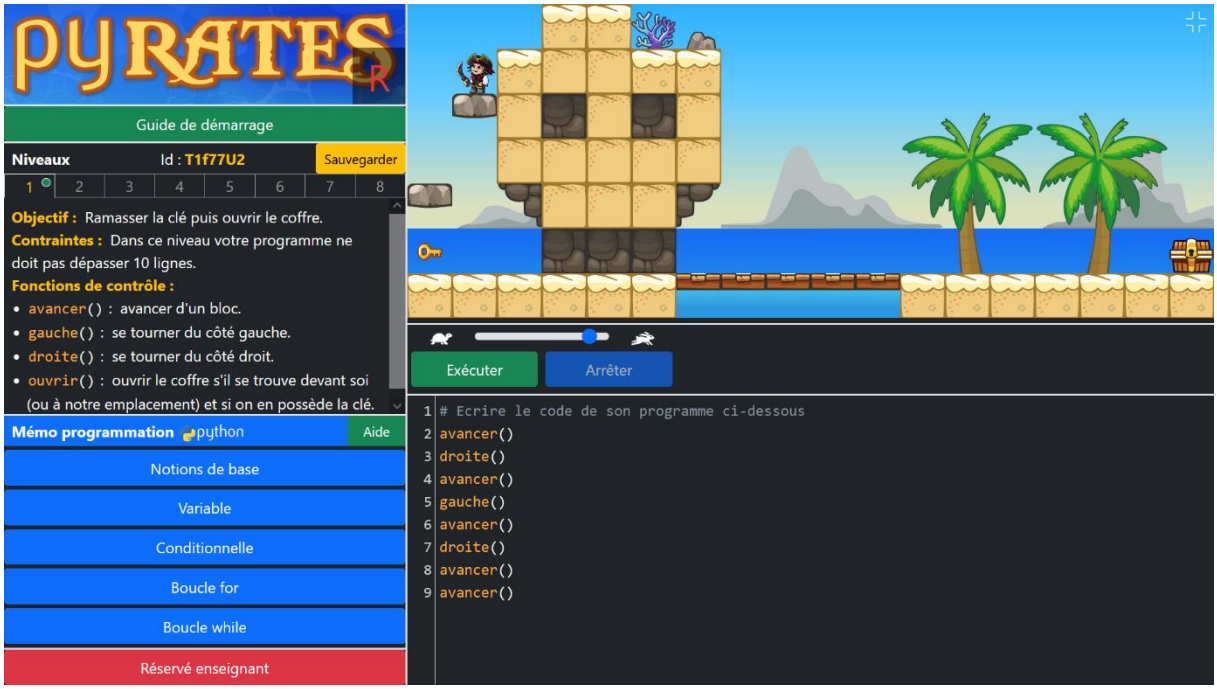


Figure 6-2 – Page principale de Pyrates dans le mode recherche

Dans cette interface modifiée, les rétroactions sont déclenchées en deux étapes. Dans un premier temps, un élève demande de l'aide par l'intermédiaire du bouton « Aide » (bouton vert dans la zone du mémo), ou un enseignant décide de proposer une aide en cliquant sur le bouton « Réservé enseignant » (long bouton rouge sous la zone mémo).

Ensuite, l'appui sur l'un ou l'autre de ces boutons entraine le gel de l'interface ce qui a pour conséquence d'empêcher toute action (consultation du mémo, édition du code, lancement de programmes, etc.) (voir Figure 6-3). Dans cette configuration, l'interface est également augmentée d'un chronomètre qui permet à l'enseignant et au

chercheur d'ordonner plus facilement leurs interventions auprès des élèves afin d'optimiser le temps d'attente (voir Figure 6-3 en haut à gauche). Enfin, des boutons accompagnés de compteurs (voir Figure 6-3 en bas à gauche) permettent le déclenchement manuel des différentes rétroactions conçues : « Obj » (objectif), « Fon » (fonction de contrôle), « Not » (notion), « Impl » (implémentation), « Sol » (solution), et « Aut » (autre).

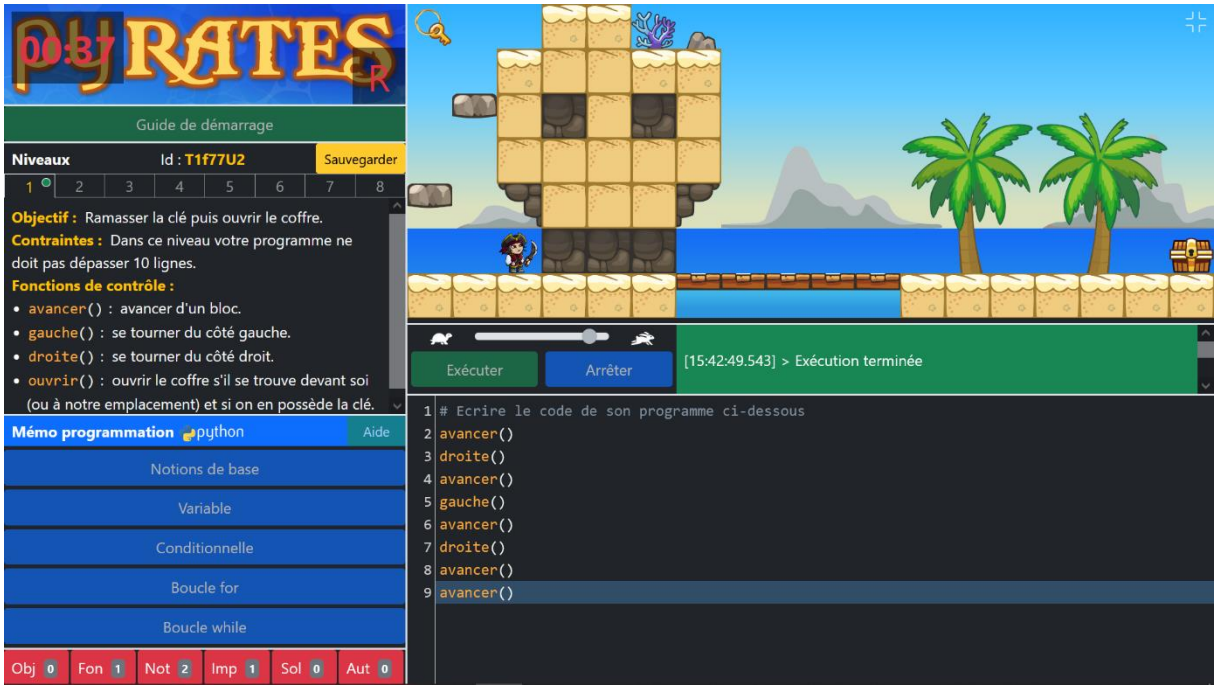


Figure 6-3 – Page principale de Pyrates dans le mode recherche lorsqu'une aide a été demandée ou proposée

En outre, nous amendons les traces générées par l'application (voir les éléments préfixés par {v2} en Annexe F) par l'ajout des éléments suivants :

- progression dans la map du niveau courant (en %) ;
- fonctions de contrôle exécutées (détection à postériori via l'arbre syntaxique abstrait des programmes) ;
- cycle de vie d'une rétroaction : demandée, reçue, lancée, relancée, quittée ;
- copie du code d'une rétroaction.

Nous rectifions également le schéma de la base de données afin de pouvoir sauvegarder les traces d'utilisations des apprenants entre deux séances. Nous ajoutons ainsi deux nouvelles tables permettant d'enregistrer des couples clé-valeur : « trace » et « activity » ainsi qu'un champ « help_mode » dans la table « game » (voir les éléments préfixés par {v2} en Annexe E).

Expérimentation 2

Cette seconde expérimentation se déroule comme la précédente en situation écologique dans des classes de seconde. Afin de constituer une nouvelle cohorte d'élèves, nous avons recontacté les enseignants volontaires lors de la première expérimentation : Yvan, Mickael, Rémi et Carole (voir section 5.5.2). Les critères de participation que nous avons fixés sont les suivants :

– élèves de seconde débutants en Python ;

– classe en demi-groupes ;

– deux séances de 55 minutes espacées d'une semaine maximum.

Les critères sont les mêmes que pour l'expérimentation initiale hormis l'espacement entre deux séances qui ne devait pas dépasser une semaine (contre deux auparavant) afin de limiter l'oubli d'une séance à l'autre (Ebbinghaus, 1913). Le Tableau 6-2 indique les données biographiques des enseignants impliqués dans cette nouvelle expérimentation.

Pseudo	Diplôme	Concours	Anc.	Disciplines
Yvan	Licence maths	CAPES Maths	17 ans	Maths, NSI
Mickael	Maitrise de maths pures	Agreg. Maths	20 ans	Maths
Rémi	Ingénieur électronique	CAPES Maths	7 ans	Maths, NSI, SNT
David	Maitrise de maths pures	Agreg. Maths	26 ans	Maths, SNT
Claude	Licence électronique	CAPET Génie élec.	28 ans	Maths, SNT
Victor	Licence Maths/Master MEEF	CAPES Maths	3 ans	Maths, SNT

Tableau 6-2 – Données biographiques des enseignants impliqués dans l'expérimentation 2, en italique, ceux ayant également participé à l'expérimentation 1

Trois enseignants sur quatre ont accepté de faire partie d'une nouvelle expérimentation. Carole n'étant pas en mesure de proposer deux séances en demi-groupe avec une semaine d'écart, elle n'a pas pu participer à cette deuxième phase. Trois nouveaux enseignants intègrent l'expérimentation (David, Claude et Victor), ce sont des collègues de Yvan et Mickael. Les séances ont donc eu lieu dans deux lycées que nous avions déjà investis la première fois : lycée E et lycée S (Voir section 5.5.2). Les enseignants participant à cette deuxième expérimentation ont des formations initiales en mathématiques et en électronique. À l'exception de Claude, ils sont titulaires d'un concours d'enseignement dans la discipline mathématiques. Ils sont globalement expérimentés voire très expérimentés (hormis Victor). Ils enseignent quasiment tous la NSI ou les SNT en complément des mathématiques (sauf Mickael qui développe par ailleurs des logiciels dans le cadre de projets personnels). Nous pouvons donc les considérer comme des experts du domaine de l'enseignement de la programmation.

Comme nous l'avons expliqué au début du chapitre, lors de cette nouvelle phase sur le terrain, l'objectif était de collecter des données permettant d'entrainer des modèles d'IA. Ces modèles doivent pouvoir choisir automatiquement la rétroaction à afficher en cas de demande en fonction de l'activité préalable de l'apprenant dans le système. Le mode opératoire a été présenté en amont aux enseignants lors de réunions en visioconférence ou en présentiel. Il consiste à présenter rapidement l'application, puis à laisser les élèves utiliser le jeu de manière autonome en évitant de communiquer entre eux. Les enseignants avaient pour consigne de n'intervenir qu'à la demande des élèves ou lors d'un blocage prolongé. Lorsqu'ils intervenaient, l'objectif était d'aider les élèves en utilisant autant que possible le déclenchement de rétroactions via les différents boutons qu'ils avaient à disposition (voir Figure 6-3). Le bouton « Aut »

permettait de signaler une aide apportée hors rétroaction, comme par exemple une indication donnée à l'oral. Notons que, comme la première fois, nous avons aidé l'enseignant de la classe dans cette tâche en agissant selon la même procédure.

Comme le récapitule le Tableau 6-3, nous avons pu récolter les données de 215 élèves répartis dans huit classes.

Experts	Lycée	Groupe	Modalité	Séances / écart
David, Chercheur	Lycée E	2nd J (28)	Demi groupe	2 x 1h / 6 jours
Yvan, Chercheur	Lycée E	2nd I (31)	Demi groupe	2 x 1h / 5 jours
Claude, Chercheur	Lycée E	2nd E (28)	Demi groupe	2 x 1h / 4 jours
Mickael, Chercheur	Lycée E	2nd F (31)	Demi groupe	2 x 1h /7 jours
Rémi, Chercheur	Lycée S	2nd 4 (30)	Demi groupe	2 x 1h / 7 jours
Rémi, Chercheur	Lycée S	2nd 10 (20)	Groupe entier	2 x 1h / 7 jours
Victor, Chercheur	Lycée S	2nd 2 (25)	Demi groupe	2 x 1h / 7 jours
Victor, Chercheur	Lycée S	2nd 9 (22)	Groupe entier	2 x 1h / 7 jours

Tableau 6-3 – Détails des séances de l'expérimentation 2

Ces données concernent l'activité des élèves : consultations et copiés-collés des contenus du mémo, lancement des programmes, erreurs commises, progression dans le niveau courant, concepts implémentés, fonctions de contrôle utilisées, etc. Elle concernent également l'activité des experts à travers les rétroactions déclenchées. Le Tableau 6-4 donne une vision synthétique de ces traces ponctuelles et horodatées (voir Annexe F pour plus de détails).

Code	Activité	Informations
GEN_COM	Généralités : commun à toutes les traces	Horodatage, identifiant de partie, niveau du jeu, temps écoulé depuis le début du niveau
CON_CON	Consultations de contenus : différents sous-concepts du mémo	Identifiant du contenu (base-*, var-*, condi-*, for-*, while-*), catégorie du contenu (base, var, condi, for, while), durée de consultation
COP_CON	Copie de contenus : éditeur de code, fonction de contrôle, sous-concepts du mémo, contenu des rétroactions	Identifiant du contenu (code-editor, control-functions, base-*, var-*, condi-*, for-*,while-*, help-content)
COL_CON	Collage de contenus	Code collé
LAN_PRO	Lancement de programmes	Code du programme, vitesse d'exécution, issue du programme (too-many-lines, syntactic-error, semantic-error, game-error, user-stopped, level-lost, level-completed, fully-executed), erreur dans le code, erreur dans le jeu (open-chest-location, open-chest-key, read-message-location , walk-location, not-allowed-function, function_parameters), raison de perte d'un niveau (spike-touch, barrel explosion, pirate-shot), progression dans le niveau, *concepts implémenté* (var-*, condi-*, etc.), *fonctions de contrôle utilisées* (walk, left, right, open, jump, etc.).

NAV_NIV	Navigation entre les niveaux	Commencé, terminé, recommencé, repris, quitté
RET_DEC	Déclenchement de rétroaction par l'enseignant ou le chercheur	Origine demande (student, teacher), type de rétroaction (control, notion, implementation, solution, other)
MOD_VIT	Modification du curseur de vitesse exécution	Nouvelle valeur de la vitesse

Tableau 6-4 – Traces générées par l'application Pyrates lors de l'expérimentation 2, en italique, détections réalisées à postériori à partir de l'arbre de syntaxe abstraite des programmes exécutés

Notons que de manière similaire à l'expérimentation 1, certaines informations qui apparaissent en italique sont détectées à postériori via l'analyse de l'arbre syntaxique des programmes exécutés (voir section 5.5.2).

Ces 73 189 traces ponctuelles et horodatées témoignent donc à la fois de l'activité des élèves dans les niveaux et des rétroactions déclenchées par les enseignants afin de leur venir en aide dans des situations de demande ou de besoin identifié. Nous les utilisons dans la suite comme des données d'apprentissage étiquetées à destination d'algorithmes d'IA.

Conception 2

Cette deuxième phase de conception vise à entraîner des modèles d'IA afin qu'ils puissent sélectionner la rétroaction à fournir aux utilisateurs de Pyrates lorsqu'ils demandent de l'aide (Q6.3).

Ayant à notre disposition des données ponctuelles portant sur l'activité des élèves et des enseignants, la première étape consiste à créer un jeu de données ayant pour *caractéristiques* (features) l'historique des actions des élèves depuis le début du niveau jusqu'au déclenchement d'une rétroaction, et pour *étiquette* (label) la rétroaction choisie par un des experts. Il s'agit ensuite d'entraîner un *classificateur multi-classes* (multi-class classifier) par *apprentissage automatique* (machine learning) (Géron, 2019). Ce classificateur aura pour tâche de prédire la rétroaction à fournir aux utilisateurs en fonction de leur activité préalable. Chaque niveau mettant en jeu des concepts particuliers, disposant d'une map et de fonctions de contrôle dédiées, nous choisissons d'entraîner un classificateur pour chacun d'eux. Cela doit nous permettre d'obtenir des prédictions plus pertinentes et d'avoir une meilleure explicabilité[38] de l'IA à travers une analyse de l'importance des caractéristiques par niveau (voir plus loin).

Afin de procéder à l'entrainement des modèles, nous employons la méthodologie en sept étapes proposée par Géron (2019) que nous détaillons ci-dessous. La mise en œuvre s'appuie sur la bibliothèque Python d'apprentissage automatique *Scikit-learn* (Pedregosa et al., 2011).

La première étape de cette méthodologie consiste à découvrir et à visualiser le jeu de données brut. Ainsi, pour chaque niveau du jeu, nous listons les caractéristiques

[38] La notion d'explicabilité pour un système d'IA signifie que les humains doivent pouvoir obtenir une explication factuelle, directe et claire des processus de prise de décision réalisées (Floridi et al., 2018).

pertinentes et étudions leur histogramme, donnons le nombre d'*observations* (instances), et analysons la distribution des étiquettes.

Lors de la deuxième étape, il s'agit de constituer aléatoirement un *jeu de test* (test set) qui sera mis de côté pour l'évaluation finale des modèles : « Pick some instances randomly, typically 20% of the dataset (or less if your dataset is very large), and set them aside » (Géron, 2019, p. 51). Comme l'illustre la Figure 6-4, nous choisissons, en suivant Géron, de mettre de côté 20% du jeu de données.

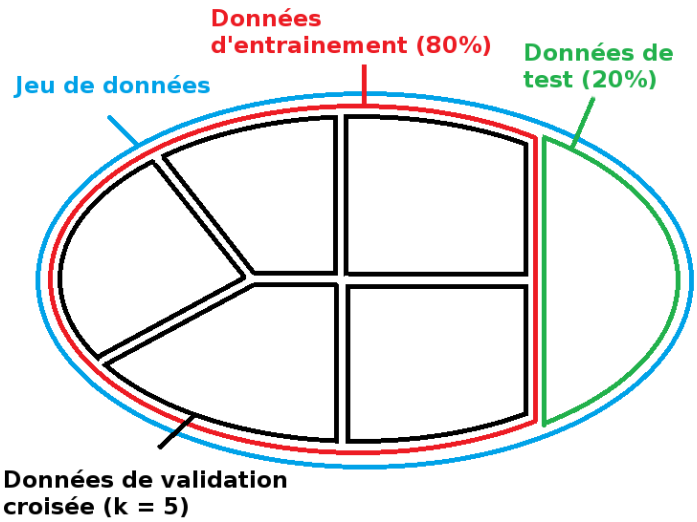


Figure 6-4 – Séparation du jeu de données pour l'entrainement et l'évaluation des modèles

Le reste des données constitue alors le *jeu d'entrainement* (train set) qui sera utilisé pour l'apprentissage des modèles.

Nous verrons plus loin que les données dont nous disposons ne sont pas équilibrées (la distribution des étiquettes, et donc des rétroactions, n'est pas homogène). Nous établissons donc le jeu de test par *échantillonnage stratifié* (stratified split) afin de préserver la distribution déséquilibrée des étiquettes : « the population is divided into homogeneous subgroups called strata, and the right number of instances are sampled from each stratum to guarantee that the test set is representative of the overall population » (Géron, 2019, p. 53). De plus, l'échantillonnage que nous effectuons évite que les données d'une même partie ne se retrouvent à la fois dans le jeu d'apprentissage et le jeu de test (split over groups). Cela prévient la contamination du jeu de test par des données proches de celles qui ont servi à l'entrainement des classificateurs, ce qui aurait pour conséquence d'obtenir des performances trop optimistes.

La troisième étape a pour objectif de préparer les données d'entrainement pour l'apprentissage des modèles. Pour cela, nous calculons d'abord le coefficient de Pearson (Cohen et al., 2009) entre les caractéristiques deux à deux afin de repérer les éventuelles corrélations linéaires. Nous supprimons ensuite une des deux caractéristiques lorsqu'une paire est très corrélée ($|r| > 0.8$) afin d'éviter la redondance d'information. En effet, deux caractéristiques très corrélées contiennent des informations proches, et supprimer l'une d'entre-elles entraine donc une faible perte d'information. Ce traitement permet de simplifier les modèles et donc de rendre

l'apprentissage et les prédictions plus rapides, mais également d'améliorer leur explicabilité (Nicodemus & Malley, 2009). Nous transformons ensuite les caractéristiques afin qu'elles prennent leurs valeurs dans le même intervalle (scaling) : « Machine Learning algorithms don't perform well when the input numerical attributes have very different scales. » (Géron, 2019, p. 69). Nous utilisons pour cela la normalisation min-max qui projette les données numériques sur une échelle de 0 à 1 : « Min-max scaling (many people call this normalization) is the simplest: values are shifted and rescaled so that they end up ranging from 0 to 1 » (Géron, 2019, p. 69). Enfin, afin d'encore simplifier les modèles, nous supprimons les caractéristiques qui ne varient pas (même valeur nulle pour toutes les observations), comme celles en lien avec les fonctions de contrôle non utilisables dans certains niveaux.

La quatrième étape consiste à choisir une méthode parmi celles permettant d'implémenter un classificateur multi-classes. Nous entrainons pour cela plusieurs modèles-candidats sur le jeu de données d'apprentissage avant de comparer leurs performances. Pour mesurer la performance de ces différents modèles, nous avons besoin d'utiliser une métrique adéquate. Notre jeu de données étant déséquilibré, nous n'utilisons pas la mesure classique d'*exactitude* (accuracy) qui est le taux de prédictions correctes parmi toutes les prédictions (« Quelle proportion des prédictions sont correctes ? ») : « accuracy is generally not the preferred performance measure for classifiers, especially when you are dealing with skewed datasets (i.e., when some classes are much more frequent than others) » (Géron, 2019, p. 90). Nous adoptons à la place trois autres indicateurs plus appropriés (Géron, 2019) que nous décrivons ci-dessous.

– La *précision* (precision) qui correspond à la proportion des prédictions correctes parmi les prédictions positives faites (« Quelle proportion de rétroactions prédites sont correctes ? »). Plus la valeur s'approche de 1, plus le nombre de faux positifs est faible.

$$precision = \frac{Vrais\ positifs}{Vrais\ positifs + Faux\ positifs}$$

– Le *rappel* (recall) qui correspond à la proportion des prédictions faites parmi les prédictions correctes (« Quelle proportion des rétroactions correctes sont prédites ? »). Plus la valeur s'approche de 1, plus le nombre de faux négatifs est faible.

$$recall = \frac{Vrais\ positifs}{Vrais\ positifs + \ Faux\ négatifs}$$

– Le *score F1* (F1-score) qui permet de combiner la précision et le rappel en calculant leur moyenne harmonique.

$$F_1\ Score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

Ces différentes métriques sont définies pour des tâches de classification binaire. Étant dans un contexte multi-classes (choix parmi quatre étiquettes), nous utilisons les moyennes pondérées par la fréquence des étiquettes (weighted). Nous entrainons donc plusieurs modèles à l'aide d'algorithmes différents puis nous sélectionnons le plus

performant au regard des métriques décrites précédemment. Parmi les méthodes permettant d'implémenter un classificateur multi-classes, nous retenons (Géron, 2019) :

- la *classification aléatoire* (retourne une rétroaction au hasard façon uniforme), qui nous sert de base afin de comparer les performances des autres modèles (Random baseline) ;
- la *classification naïve bayésienne* (Naive Bayes) ;
- la *régression logistique* (Logistic Regression) ;
- la *machine à support de vecteur* à noyau radial (RBF kernel Support Vector Machine) ;
- l'*arbre de décision* (Decision Tree) ;
- la *forêt aléatoire* (Random Forest).

Afin d'évaluer les performances de ces différents modèles candidats sur le jeu de données d'apprentissage sans solliciter les données du jeu de test, nous utilisons la méthode de la validation croisée (k-fold cross validation) (Fushiki, 2011). Comme le décrit la Figure 6-4, cette méthode consiste à diviser le jeu de données d'apprentissage aléatoirement en k blocs (fold), puis à sélectionner un des k blocs comme *jeu de validation* (validation set), alors que les k-1 autres blocs constituent le jeu d'apprentissage. Après l'entrainement du modèle sur le jeu d'apprentissage, les métriques sont calculées à partir du bloc de validation. L'opération est ensuite répétée en faisant tourner le rôle des blocs (on sélectionne un autre bloc de validation parmi les k blocs prédéfinis). À l'issue de la procédure nous obtenons ainsi k scores de performances :

> K-fold cross-validation [...] randomly splits the training set into [k] distinct subsets called folds, then it trains and evaluates the [...] model [k] times, picking a different fold for evaluation every time and training on the other [k-1] folds. The result is an array containing the [k] evaluation scores. (Géron, 2019, p. 73)

Nous calculons ensuite la moyenne et l'écart type de ces scores afin d'estimer les performances de chaque modèle. Nous choisissons un paramétrage standard de cinq blocs (k=5). Le fractionnement en blocs est réalisé à l'aide d'un échantillonnage aléatoire stratifié et tenant compte des identifiants de partie (stratified K-fold over groups) afin de respecter les proportions des étiquettes et de ne pas contaminer les jeux de validation avec des données des jeux d'apprentissage. Cette méthode permet donc de choisir un modèle parmi les six évalués pour chaque niveau du jeu.

La cinquième étape est consacrée à l'ajustement des hyperparamètres du modèle choisi dans l'objectif d'améliorer ses performances (hyperparameters tuning). Nous procédons pour cela à une *recherche par grille* (grid search) consistant à établir des listes de valeurs pour certains hyperparamètres puis à calculer le score F1 par validation croisée sur le jeu de données d'apprentissage pour toutes les combinaisons de valeurs possibles : « All you need to do is tell it which hyperparameters you want it to experiment with and what values to try out, and it will use cross-validation to evaluate all the possible combinations of hyperparameter values » (Géron, 2019, p. 76). Les valeurs des hyperparamètres menant aux meilleures performances seront conservées pour le modèle final pour chaque niveau du jeu.

Les hyperparamètres des modèles sélectionnés étant ajustés pour chaque niveau, il reste, lors de la sixième étape, à les évaluer sur des données auxquelles ils n'ont jamais été confrontés : le jeu de test. Nous représentons les résultats sous la forme de *matrices de confusion* ainsi qu'en donnant les différents scores obtenus : « A […] way to evaluate the performance of a classifier is to look at the confusion matrix. The general idea is to count the number of times instances of class A are classified as class B » (Géron, 2019, p. 90).

Enfin, la septième et dernière étape consiste à mesurer l'importance des caractéristiques lors des prédictions réalisées par les différents modèles d'IA par niveau. En effet, certaines méthodes d'apprentissage automatique comme les forêts aléatoires sont des procédures de type « boites noires » difficilement interprétables par les humains. Or, les recommandations pour une IA éthique au niveau mondial[39] poussent à la transparence et à l'explicabilité des discisions (Jobin et al., 2019). Dans cette optique, nous calculons l'importance des différentes caractéristiques pour chaque niveau afin de donner une idée des critères de sélection des rétroactions.

Production

À l'heure où nous écrivons ces lignes, la mise en production du système de rétroactions épistémiques de Pyrates n'est pas encore réalisée et fait partie des prolongements envisagés de la thèse. Néanmoins, lorsque cette évolution sera mise à disposition des utilisateurs finaux, un clic sur le bouton de demande d'aide entrainera l'interrogation du modèle d'IA du niveau courant qui prédira, à partir de l'historique des traces d'activités de l'utilisateur depuis le début du niveau, la rétroaction à fournir à l'élève demandeur.

Nous avons présenté la méthodologie utilisée dans ce chapitre, la section suivante donne les résultats concernant le diagnostic de la progression et de l'autonomie des élèves dans le jeu.

6.4 Évaluation de la progression et de l'autonomie

Commençons par examiner l'avancée des élèves au cours des deux séances de la première expérimentation ainsi que leur autonomie vis-à-vis de l'enseignant (Q6.1).

6.4.1 Avancée dans le jeu

La Figure 6-5 présente le nombre d'élèves de la première expérimentation ayant commencé et terminé les différents niveaux de Pyrates.

[39] En Europe, la proposition de règlement déposée par la commission européenne en 2021 (*Artificial Intelligence Act*, 2021) devrait entrer en application dans les prochaines années. Elle définit les « système IA » comme à « haut-risque » dans le domaine de l'éducation uniquement s'ils régissent l'accès aux études ou participent à l'évaluation des apprenants. Notre système entrerait lui dans la catégorie à « risque limité » dans la mesure où l'IA interagit directement avec les utilisateurs. Cela impliquerait de les notifier du fait qu'ils interagissent avec une IA et de préciser quelles données personnelles sont collectées et si elles sont utilisées pour les classifier.

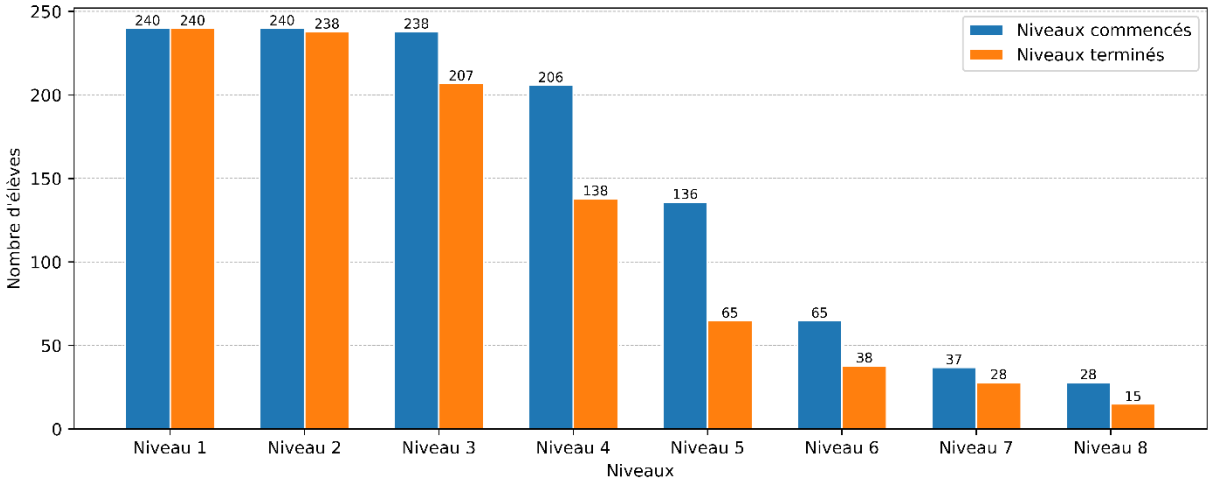


Figure 6-5 – Nombre d'élèves ayant commencé et terminé les différents niveaux du jeu

Nous pouvons constater que les trois premiers niveaux sont terminés par la grande majorité des élèves (86% des élèves parviennent à commencer le niveau 4). Cependant, à partir du niveau 4, les élèves ne sont pas tous en capacité de finir, et le taux d'achèvement décline niveau après niveau. Finalement, seuls 15 élèves parviennent à terminer complètement le jeu.

La Figure 6-6 indique le temps moyen en minutes nécessaire aux élèves pour terminer les différents niveaux du jeu.

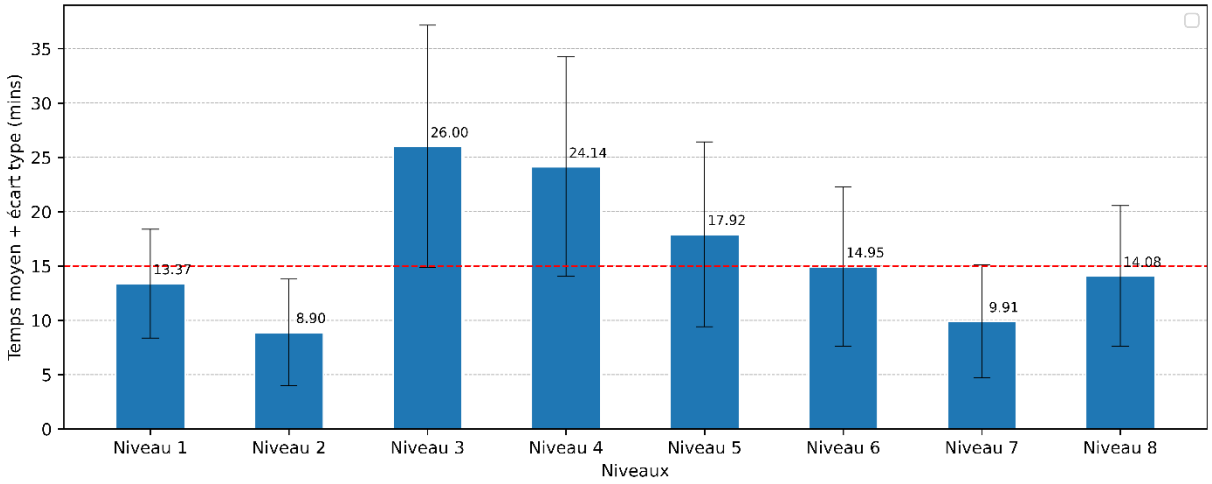


Figure 6-6 – Temps moyen et écart-type du temps passé par élève par niveau terminé

Rappelons que chaque niveau a été conçu pour une durée maximum de quinze minutes (ligne rouge pointillée sur la figure). Si les deux premiers niveaux respectent cette contrainte temporelle, les niveaux 3, 4 et 5 dépassent ce temps estimé à priori. Avec respectivement 26 et 24 minutes en moyenne, les niveau 3 et 4 sont particulièrement longs. Ajoutons que les niveaux 6, 7 et 8 connaissent des temps de jeu acceptables, gardons cependant à l'esprit qu'ils ne concernent que la minorité des élèves les plus en réussite.

6.4.2 Autonomie par rapport à l'enseignant

Intéressons-nous maintenant à l'autonomie des élèves. Commençons par rappeler que lors de l'expérimentation que nous avons menée, les élèves étaient en demi-groupes (environ 15 élèves) et étaient encadrés par deux experts (leur enseignant et le chercheur) qui se tenaient à leur disposition pour les aider en cas de sollicitation ou de blocage prolongé. En se basant sur ces informations, estimons le temps moyen d'une aide dans cette modalité d'encadrement. 889 aides ont été dispensées pendant les 32 séances de 55 minutes que comportait la première expérimentation. En considérant que, selon nos observations, l'enseignant et le chercheur ont passé au total environ 45 minutes à aider les élèves par séance, nous pouvons considérer qu'une aide dure en moyenne 3,2 minutes ($(32 \times 45) : (889 : 2) \approx 3{,}24$). Par conséquent, dans le contexte d'un enseignant seul pendant deux séances de 55 minutes (nous comptons 50 minutes de cours effectif), cet enseignant est en capacité de donner 31 aides ($50 \times 2 : 3{,}2 = 31{,}25$). En considérant le cas le plus défavorable d'une classe entière (30 élèves), cela correspond globalement à une aide par élève sur les deux séances. Étant donné que le jeu comporte huit niveaux, le nombre d'aides qu'un enseignant peut fournir par élève et par niveau est très réduit dans un contexte conventionnel (le calcul donnerait en moyenne 0,125 aide par élève et par niveau).

La Figure 6-7 présente le nombre d'aides effectivement reçues en moyenne par les élèves lors de l'expérimentation 1 (en demi-groupe et avec deux experts). La ligne rouge pointillée correspond aux 0,125 aides potentielles de la modalité d'enseignement classique (en groupe entier avec un enseignant).

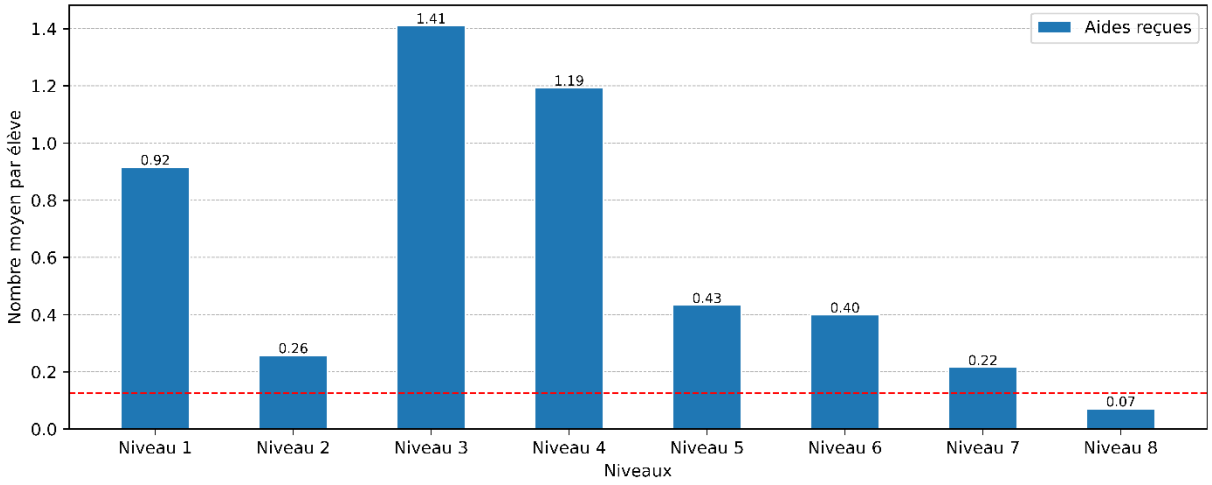


Figure 6-7 – Nombre moyen d'aides reçues par élève et par niveau

Rappelons que les niveaux 3 et 4 durent en moyenne respectivement 26 et 24 minutes, ce qui explique en partie le grand nombre d'aides fournies.

Nous constatons d'abord qu'à part pour le niveau 8, tous les niveaux dépassent la limite d'aides potentielles. Ensuite, dans le niveau 1 et plus encore dans les niveaux 3 et 4, les élèves requièrent énormément l'aide des experts, jusqu'à dix fois la limite.

6.4.3 Hypothèses explicatives

Tentons de proposer des hypothèses capables d'expliquer les constats que nous venons d'établir.

Dans le niveau 1, les élèves ont besoin de beaucoup d'aides, c'est sans doute la conséquence de la découverte du jeu et plus globalement de l'environnement de l'application (voir Figure 6-9 pour le détail des aides).

Les niveaux 3 et 4 sont trop longs et engendrent beaucoup d'interventions des experts. Ce constat est sans doute explicable par une densité conceptuelle trop forte. Ainsi dans le niveau 3, c'est la première fois que la map du niveau est dynamique et change aléatoirement avant chaque exécution. De plus, plusieurs concepts nouveaux sont introduits : les arguments d'une fonction, le retour d'une fonction, ainsi que l'affectation et l'utilisation d'une variable (voir Annexe G.3.2). En outre, Qian et Lehman (2017) montrent que le concept de variable est particulièrement difficile à appréhender pour les débutants car il peut être source de conceptions erronées : les élèves peuvent ne pas comprendre qu'une variable ne contient qu'une seule valeur à la fois, que l'ordre des énoncés de l'affection est important (a=6 ou 6=a), ils peuvent croire à tort que la signification du nom de la variable influence sa valeur, etc.

Le niveau 4 introduit le concept de structure conditionnelle (voir Annexe G.4.2) qui est un autre concept délicat qui peut faire l'objet de mécompréhensions : certains élèves imaginent que les énoncés des branches if et else d'une expression conditionnelle sont toutes les deux exécutées, d'autres pensent à tort que si la condition d'une branche if est fausse, l'exécution de tout le programme s'arrête, etc. (Qian & Lehman, 2017). Ajoutons que les concepts s'accumulent d'un niveau à l'autre, alors que les élèves n'ont pas forcément le recul nécessaire pour les articuler entre eux.

Enfin, d'une manière générale, même si les élèves parviennent à les accomplir en moins de 15 minutes, tous les niveaux à l'exception du dernier nécessitent trop d'interventions de l'enseignant pour un contexte d'enseignement classique.

6.4.4 Pistes d'améliorations

Afin de remédier aux limites que nous venons de pointer, nous proposons plusieurs pistes d'améliorations :

- il est souhaitable de découper le niveau 3 en deux sous-niveaux, l'un introduisant le concept de retour d'une fonction et l'autre la notion de variable ;
- il est possible d'ajouter un deuxième niveau portant spécifiquement sur les variables afin de consolider l'acquisition de ce concept avant de l'articuler avec une conditionnelle dans le niveau suivant ;
- il est nécessaire de développer un système d'aides automatisées capable d'épauler l'enseignant dans l'accompagnement des élèves pour tous les niveaux.

Les deux premiers points requièrent la restructuration d'un niveau existant et l'ajout d'un nouveau niveau. Cela nécessite une nouvelle itération de l'ingénierie didactique que nous n'avons pas eu le temps d'opérer lors de cette thèse et qui trouve naturellement sa place dans ses prolongements. Néanmoins, nous présentons dans la suite de ce chapitre la conception d'un système d'aides automatiques prenant la forme de rétroactions épistémiques sélectionnées par des modèles d'IA en fonction de l'activité des élèves dans l'application.

6.4.5 Synthèse de la section

Q6.1 : Quelle est la progression des élèves dans le jeu et leur autonomie vis-à-vis de l'enseignant lors des deux séances de la première expérimentation ? Quelles pistes proposer pour les améliorer ?

En analysant les données récoltées lors de notre première expérimentation dans les classes, nous sommes parvenu à montrer que :

- la **moitié** des élèves arrivent à atteindre le **niveau 5** puis le taux d'achèvement décline et seuls **6%** des élèves **terminent entièrement** le jeu ;
- les **niveaux 3, 4 et 5** prennent **trop de temps** aux élèves ;
- l'**autonomie** des élèves est **très insuffisante** dans un contexte de classe entière avec un seul enseignant pour 30 élèves, en particulier pour les niveaux 1, 3 et 4.

Nous avons ensuite proposé quelques pistes d'amélioration :

- le **niveau 3** doit être **découpé** en plusieurs sous-niveaux afin de diminuer sa densité conceptuelle ;
- un système d'**aides automatisées** doit être développé pour tous les niveaux afin de **soulager** les **enseignants** dans leur tâche d'accompagnement.

Suite à cette première étape de diagnostic, nous détaillons dans les sections suivantes la mise en œuvre de la deuxième piste, c'est à dire l'élaboration d'un ensemble de rétroactions épistémiques, puis le mécanisme permettant la sélection de la rétroaction à fournir lorsqu'un élève demande de l'aide.

6.5 Conception des rétroactions

Nous présentons dans cette section la conception des rétroactions que nous allons proposer aux utilisateurs de Pyrates (Q6.2). En reprenant la grille d'analyse issue de notre revue de littérature (voir section 6.1), nous exposons tour à tour les aspects relatifs au moment du déclenchement, au contenu, puis à la génération des rétroactions (données des élèves et actions des experts).

6.5.1 Moment du déclenchement

Comme nous l'avons évoqué dans la revue de littérature, il est déconseillé de fournir aux utilisateurs d'un système des rétroactions immédiates (IM) qui s'affichent directement en fonction de leur activité. Mieux vaut laisser les actants demander de l'aide (AD), ou leur proposer de visualiser une rétroaction lorsqu'elle est disponible (PR).

Nous faisons donc le choix d'ajouter un bouton de demande d'aide dans l'interface de la version standard[40] de Pyrates (voir Figure 6-8 au niveau du mémo programmation). Ce bouton, lorsqu'il est cliqué, déclenche l'apparition d'un *tuteur* prenant la forme d'un perroquet. Ce tuteur délivre des rétroactions en fonction de la décision retournée par notre système de sélection de rétroaction (voir section 6.6). Il fournit donc, soit une rétroaction parmi les quatre disponibles par niveau (voir section 6.5.2), soit un message indiquant qu'il ne peut pas aider l'utilisateur pour l'instant (voir Figure 6-8).

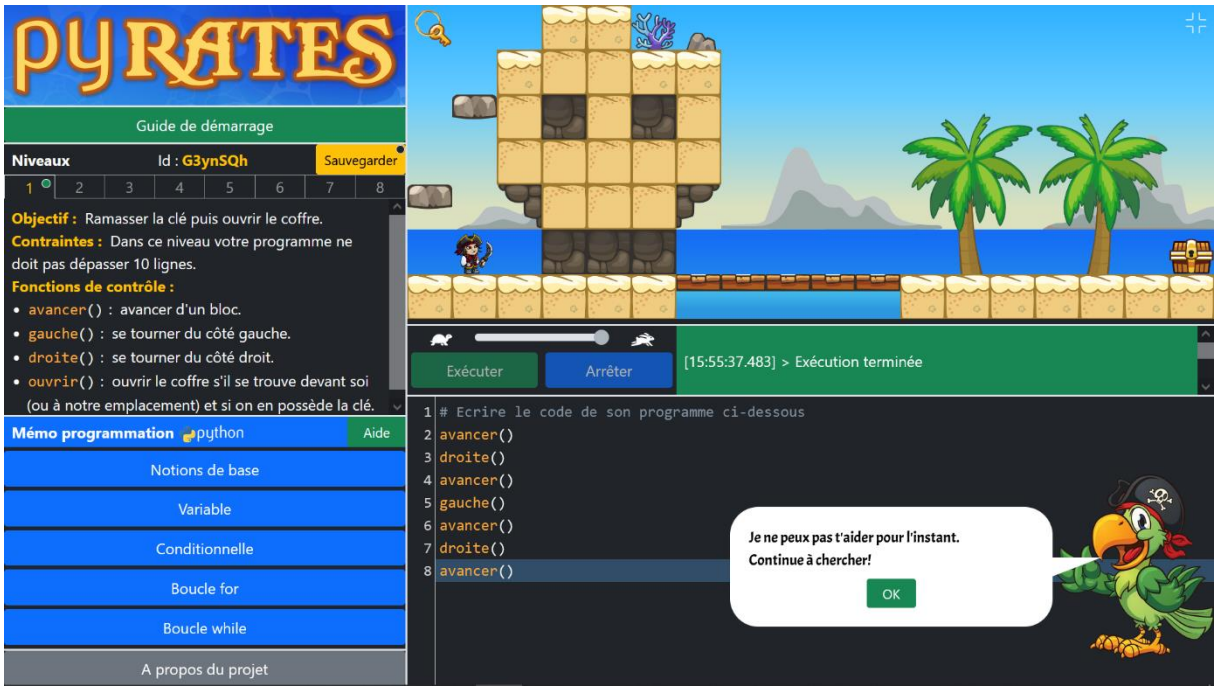


Figure 6-8 –Interface modifiée de l'application Pyrates : ajout d'un bouton de demande d'aide dans la zone du mémo et création d'un tuteur délivrant des rétroactions

Ainsi, lorsqu'un utilisateur clique sur le bouton d'aide, la *politique de déclenchement* des rétroactions est la suivante :

- le système de sélection de rétroaction est appelé, il retourne, parmi les quatre rétroactions existantes pour le niveau courant, celle qui correspond le mieux à l'activité de l'élève ;
- si la rétroaction retournée a déjà été déclenchée, le message affiché est : « Je ne peux pas t'aider pour l'instant. Continue à chercher ! » ;
- si la rétroaction retournée n'a pas encore été affichée, elle est délivrée par le tuteur.

6.5.2 Contenu des rétroactions

Ensuite, intéressons-nous au contenu des rétroactions qui peuvent être fournies par le tuteur. Comme nous l'avons expliqué dans la méthodologie (voir section 6.3.2),

[40] Rappelons que le mode standard de Pyrates est destiné aux utilisateurs finaux, et que le mode recherche comporte des fonctionnalités supplémentaires (traces d'utilisations, zone réservée aux enseignants). Ce mode est par ailleurs utilisé exclusivement lors des expérimentations que nous encadrons dans les classes.

nous nous sommes appuyé pour les concevoir sur les différents types d'aides qui ont été administrées aux élèves par les enseignants lors de notre première expérimentation. Rappelons que les enseignants avaient pour consigne de caractériser chaque aide qu'ils apportaient aux élèves selon six catégories que nous détaillons ci-dessous.

- *Application* : en rapport avec le fonctionnement de l'application elle-même. Par exemple : « Il faut écrire ton code ici », « Clique sur ce bouton pour arrêter ton programme ».
- *Jeu* : en rapport avec le jeu sérieux. Par exemple : « Utilise la fonction sauter() pour monter sur le tonneau », « Il faut d'abord prendre la clé avant d'ouvrir le coffre », « La hauteur des piles de caisses est aléatoire et change avant chaque exécution ».
- *Syntaxe* : en lien avec la syntaxe de Python. Par exemple :« Ajoute deux points ici », « Tu dois fermer cette parenthèse ».
- *Sémantique* : en lien avec le sens des programmes. Par exemple : « Si tu veux répéter cette instruction, il faut la décaler avec la tabulation », « Ici le else ne te permet pas de prendre en compte ce cas ».
- *Notion* : indication à l'élève de la notion en jeu dans le niveau courant. Par exemple : « Utilise une boucle pour faire répéter ces instructions », « Là, tu vas devoir utiliser une structure conditionnelle », « Crée une variable pour stocker le retour de cette fonction ».
- *Autre* : si aucun autre type d'aide ne convient.

La Figure 6-9 donne le détail des aides apportées par les enseignants par catégories selon les niveaux.

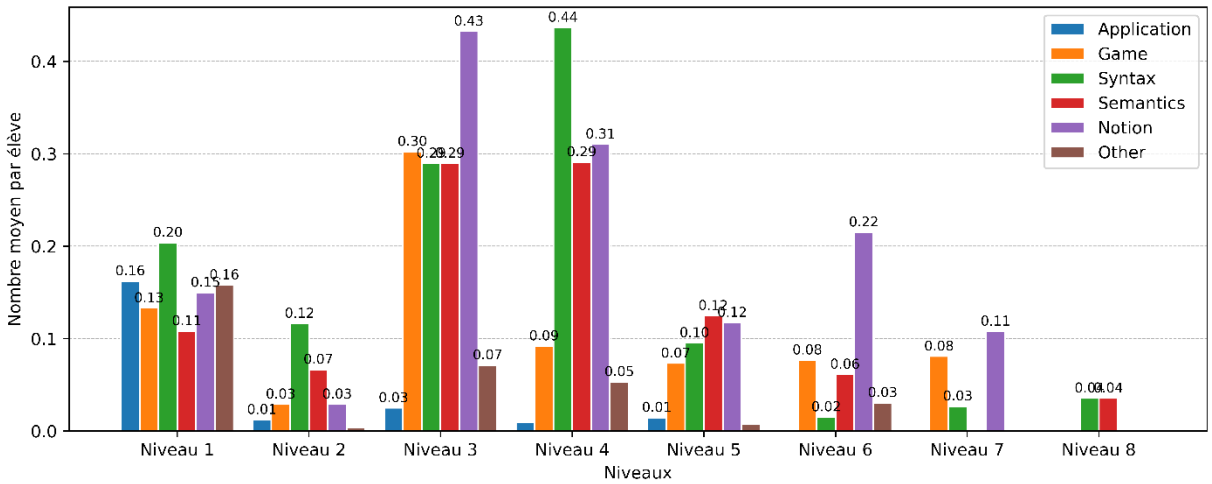
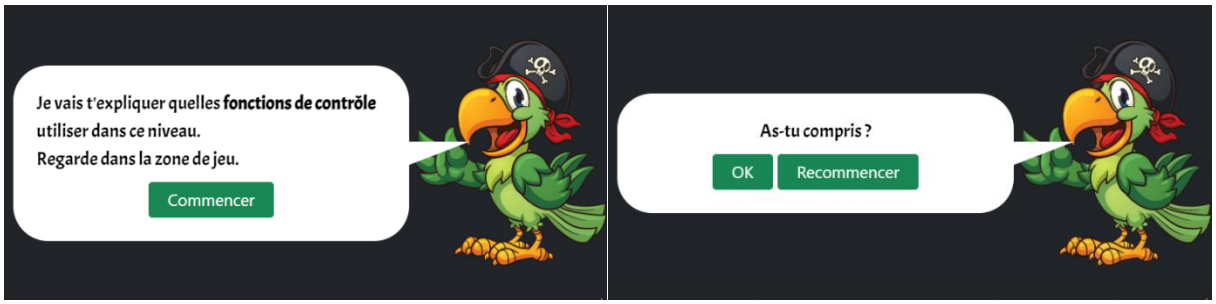


Figure 6-9 – Nombre moyen d'aides reçues dans les différentes catégories par élève et par niveau

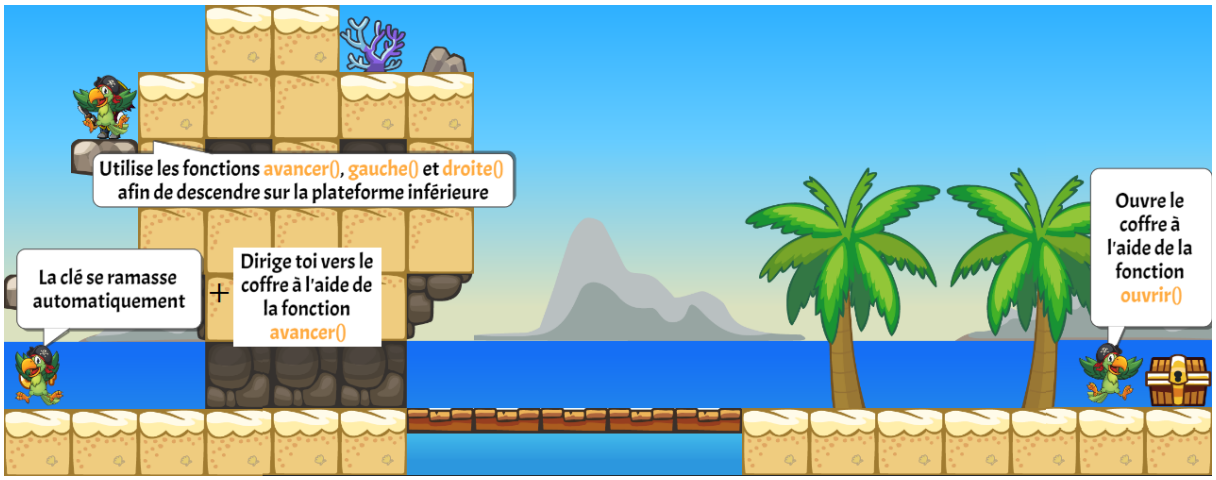
Les aides relatives au fonctionnement de l'application (Application) sont logiquement surtout présentes dans le premier niveau. Nous n'avons pas créé de rétroaction sur ce thème compte tenu de la présence d'un guide de démarrage vers lequel les enseignants peuvent diriger les élèves en début de séance (voir section 5.6.4).

Ensuite, concernant les aides en lien avec le jeu (Game), nous observons qu'elles ont été fournies assez fréquemment dans tous les niveaux excepté le dernier. Nous avons donc décidé de créer une rétroaction de type *jeu* dans laquelle le tuteur propose de regarder une animation dans la zone de jeu (voir Figure 6-10-a), puis parcourt la map du niveau afin d'expliquer ce qui est attendu tout en donnant des indications relatives aux fonctions de contrôle à utiliser (voir Figure 6-10-b). En fin d'animation, un dernier message permet de relancer l'animation ou de quitter la rétroaction (voir Figure 6-10-c).



(a)

(c)



(b)

Figure 6-10 – Rétroaction de type jeu pour le niveau 1 : invitation à lancer l'animation dans la zone de jeu (a), animation du tuteur qui parcourt la map du jeu en donnant des indications sur les fonctions de contrôle (b), et message de validation ou de relance de l'animation (c). Les rétroactions du même type pour les autres niveaux sont détaillées en Annexe L.1

Précisons que les informations contenues dans ce type de rétroactions ne diminuent pas le niveau d'adidacticité des situations étant donné qu'elles ne donnent aucun indice sur les concepts en jeu (voir section 5.2). En revanche, elles permettent d'aider les élèves à implémenter une procédure de base leur permettant de s'engager dans le problème posé (voir section 5.2). Ajoutons que le contenu de ce type de rétroactions se rapproche de la catégorie « règles de la tâche » (RT) dans le modèle proposé par Narciss (2013).

En revenant à l'analyse de la Figure 6-9, les aides de la catégorie notion (Notion), qui consistent à dévoiler le concept de programmation principalement visé dans le niveau courant, sont les plus fréquentes. Même si elles annihilent l'adidacticité des situations, et réduisent l'enjeu de la tâche à la mise en œuvre syntaxique de la notion

visée, c'est une manière de venir en aide aux élèves bloqués depuis un long moment et qui pourraient se décourager et abandonner. Ainsi, nous avons pris le parti de créer une rétroaction de type *concept* qui indique au joueur le concept principal en jeu dans le niveau en désignant la partie qui lui est consacrée dans le mémo programmation.

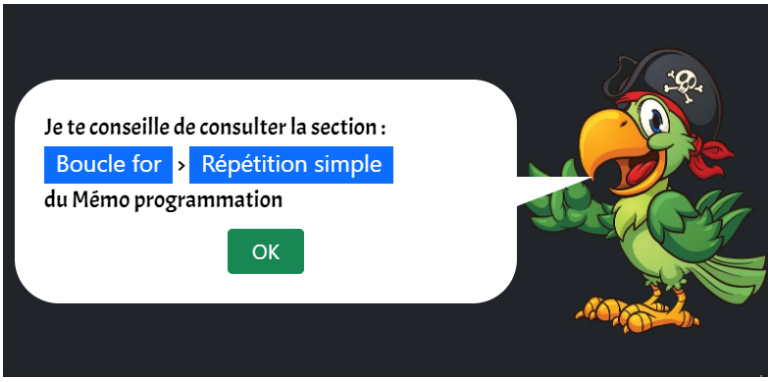


Figure 6-11 – Rétroaction de type concept pour le niveau 1, les rétroactions du même type pour les autres niveaux sont détaillées en Annexe L.2

Comme le montre la Figure 6-11, la rétroaction conseille pour le niveau 1 de consulter la partie « Répétition simple » du mémo Python. Le contenu de ces rétroactions peut être catégorisé comme « concept » (CO) selon la classification de Narciss (2013).

Toujours selon la Figure 6-9, les aides en lien avec la syntaxe (Syntax) et la sémantique (Semantics) sont également très courantes. Au sujet de la syntaxe, nous avons déjà vu que l'analyseur syntaxique que nous avons intégré à l'environnement de Pyrates semblait jouer son rôle dans la limitation des erreurs (voir section 5.9.2). Malgré cela, les enseignants doivent encore fréquemment intervenir à ce sujet. Nous avons donc choisi de concevoir en complément une rétroaction de type *implémentation* qui fournit un fragment de code facilitant l'implémentation du concept principal visé dans le niveau. Cependant, cet extrait de programme ne permet pas à lui tout seul de terminer le niveau. Il convient en effet de l'agencer avec les concepts secondaires ciblés. À titre d'exemple, la Figure 6-12 présente la rétroaction de type implémentation pour le niveau 3.

Figure 6-12 – Rétroaction de type implémentation pour le niveau 3, les rétroactions du même type pour les autres niveaux sont détaillées en Annexe L.3

Cette séquence d'instructions met en œuvre une variable afin de récupérer la hauteur des caisses à sauter (voir Annexe G.3), il est ensuite nécessaire de l'intégrer dans une

boucle for et de l'articuler dans le corps de la boucle avec d'autres déplacements du personnage pour terminer le niveau. Précisons que nous avons ajouté un bouton « Copier » ayant pour objectif de favoriser la pratique du copier-coller de la rétroaction vers l'éditeur de code afin de limiter les erreurs de syntaxe (sur le modèle de ce qui existe déjà dans le mémo). Selon le classement de Narciss (2013), le contenu de cette rétroaction peut être rangé dans la catégorie « indications procédurales » (IP). À propos des aides portant sur la sémantique des programmes (le programme ne fait pas ce que l'élève souhaite), nous ne sommes pas en mesure de concevoir des rétroactions capables de soutenir les élèves sur ce point étant donnée la grande diversité des problématiques rencontrées.

Enfin, Shute (2008) indique que les apprenants les moins performants ont besoin d'une forme de rétroaction concrète et directive qui peut aller jusqu'à donner la réponse complète au problème posé. Nous suivons cette recommandation et introduisons une dernière rétroaction de type *solution* qui fournit le code intégral permettant d'achever le niveau courant. Cette rétroaction est pensée pour être déclenchée en dernier recours, lorsque les élèves ont déjà beaucoup œuvré et sont proches de l'abandon. La Figure 6-13 reproduit la rétroaction de type solution pour le niveau 4.

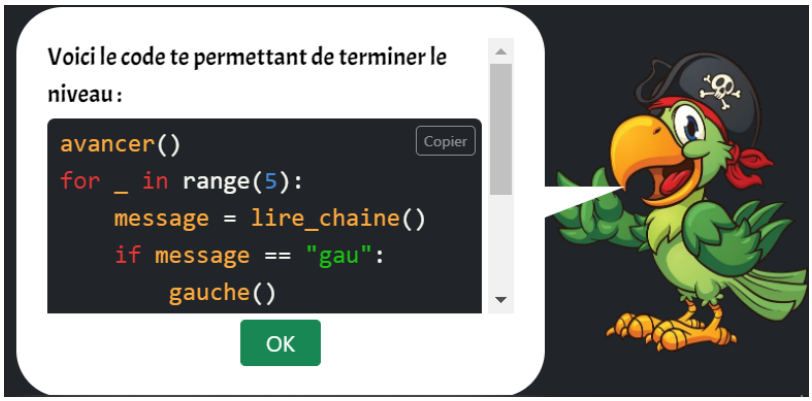


Figure 6-13 – Rétroaction de type solution pour le niveau 4, les rétroactions du même type pour les autres niveaux sont détaillées en Annexe L.4

Pour cette rétroaction, un ascenseur donne la possibilité de visualiser l'ensemble du programme-réponse, et un bouton « Copier » incite les élèves à copier-coller directement le code dans l'éditeur. Le contenu de cette rétroaction peut être classé dans le catégorie « réponse correcte » (RC) selon le modèle proposé par Narciss (2013).

Finalement, en comparaison avec les travaux présentés dans l'état de l'art (voir section 6.1.3), Pyrates offre des rétroactions variées et progressives, allant du rappel des règles de la tâche (RT), à la réponse correcte (RC), en passant par les concepts en jeu (CO) et des indictions procédurales (IP).

6.5.3 Génération des rétroactions

Relativement aux travaux présentés dans la revue de littérature, le mode de génération que nous avons choisi pour nos rétroactions allie l'exploitation des données des utilisateurs et l'intervention d'experts du domaine (chercheur et enseignant de la classe).

Données des élèves

Comme nous l'avons indiqué dans la méthodologie (voir section 6.3.2), les données des élèves que nous exploitons sont variées et ne se limitent pas au code exécuté. En référence à la classification proposée par Deeva et ses collèges (2021), il s'agit de :

- l'état du système (ET) avec l'avancée dans la map du niveau courant ;
- les réponses ouvertes (RO), c'est-à-dire les programmes lancés dont nous analysons les arbres de syntaxe abstraite afin d'en extraire les concepts implémentés et les fonctions de contrôle utilisées ;
- le comportement dans le système (CS), à travers les erreurs réalisées (dans le jeu ou le code), le temps de consultation des différentes parties du mémo Python, les copiés-collés de code, ainsi que le comportement vis-à-vis de l'exécution des programmes (lancement, arrêt, changement de vitesse).

En comparaison des travaux présentés dans l'état de l'art qui se basent généralement sur le code de l'utilisateur (RO), en exploitant parfois l'état du système (ET) ou le comportement des utilisateurs (CS), notre génération de rétroaction s'appuie sur les trois types de données à la fois (ET, RO et CS). Le fait d'avoir accès à cette variété de traces d'utilisations dans l'application doit permettre à notre système de sélection de rétroaction basé sur l'IA de performer davantage et d'avoir une meilleure explicabilité.

Experts du domaine

Tandis que certaines études se basent uniquement sur les données des utilisateurs pour générer des rétroactions, nous faisons le choix de faire appel à des experts du domaine de la programmation que sont les enseignants présentés dans la méthodologie (voir section 6.3.2), ainsi que nous même en tant que chercheur. Comme nous l'avons déjà évoqué dans la revue de littérature (voir section 6.1.3), cela permet, malgré un temps de développement supérieur et une flexibilité moindre, de proposer des rétroactions plus fiables, plus riches et plus variées. Ainsi, comme nous l'avons indiqué dans la méthodologie (voir section 6.3.2), notre solution fait intervenir des experts à deux moments :

- lors de la première expérimentation sur le terrain, les enseignants et le chercheur agissent de la même façon en donnant d'abord des aides orales aux élèves avant d'en catégoriser le contenu à l'aide des différents boutons présents dans l'interface (voir section 5.5.2).
- lors de la conception des rétroactions, le chercheur élabore différents types de rétroactions épistémiques à la lumière de son expertise didactique en se basant sur l'analyse des aides fournies lors de l'expérimentation 1 ;
- au cours de la deuxième expérimentation, les enseignants et le chercheur déclenchent des rétroactions préétablies en fonction des besoins diagnostiqués chez les élèves.

Les autres études de notre revue de travaux qui font intervenir des experts les mobilisent également pour concevoir les différentes rétroactions. Cependant, le choix du contexte qui nécessite ou pas une rétroaction ne revient pas à ces experts. Ces

contextes sont choisis par des méthodes de clustering faisant ressortir quelques cas représentatifs parmi des différents groupes de situations établis. De plus, cet étiquetage à postériori en laboratoire ne permet d'avoir accès qu'à une petite part du contexte (le programme des élèves et, pour une étude, l'état du jeu sérieux). Dans cette thèse, nous proposons de collecter les décisions des experts directement dans les salles de classe, en leur permettant de déclencher les différents types de rétroaction en ayant accès à l'ensemble du contexte. À notre connaissance, cette approche n'a pas été utilisée pour établir une politique de sélection de rétroaction basée sur l'IA dans un contexte d'enseignement de la programmation.

6.5.4 Synthèse de la section

Q6.2 : Quelles rétroactions épistémiques concevoir dans l'objectif de soutenir l'autonomie des élèves qui utilisent Pyrates ?

En analysant les données récoltées concernant les aides données lors de la première expérimentation, nous avons conçu un ensemble de rétroactions épistémiques :

- qui peuvent être déclenchées **à la demande** en cliquant sur le nouveau **bouton** « Aide » de l'interface ;
- qui sont délivrées par un **tuteur** prenant la forme d'un **perroquet** dans des **bulles de dialogue** ou via des **animations** dans la map du jeu ;
- avec une **politique de déclenchement** empêchant la diffusion répétée de la même rétroaction ;
- qui compte quatre rétroactions, une de type **jeu** qui **explique** le **parcours** à effectuer dans la map du niveau tout en indiquant les **fonctions de contrôle** à utiliser, une de type **concept** qui dévoile le **concept principal visé** dans le niveau en pointant vers un contenu du mémo, une de type **implémentation** qui fournit un **fragment de code** facilitant l'implémentation du **concept principal visé** dans le niveau, et une de type **solution** qui donne le **code intégral** permettant d'**achever le niveau** ;
- qui sont générées en s'appuyant sur les **données des élèves** (état du système, réponses ouvertes, et comportement dans le système) et sur les **décisions d'experts** prises **in situ** dans salles de classes.

Après avoir présenté la conception des rétroactions, nous exposons notre méthode de sélection des rétroactions au moment où un élève demande de l'aide.

6.6 Sélection de la rétroaction

Nous décrivons dans cette section le mécanisme de sélection de la rétroaction à fournir lorsqu'un utilisateur de Pyrates clique sur le bouton « Aide » de l'interface (Q6.3).

6.6.1 Analyse du problème

Cette tâche consiste donc à choisir dans chaque situation le bon type de rétroactions épistémiques parmi les quatre types que nous avons conçus. Ce choix doit se faire en fonction du comportement de l'élève dans l'application depuis le début du niveau courant. L'enjeu consiste donc à venir en aide aux élèves en difficulté tout en essayant de maintenir le niveau d'adidacticité des situations, autrement dit ne pas dévoiler trop rapidement les concepts en jeu (voir section 5.3). Comme évoqué dans la partie méthodologique (voir section 6.3.2), nous avons choisi de mettre en œuvre pour cela un classificateur multi-classes pour chaque niveau que nous entrainons par apprentissage automatique à partir des données récoltées lors de l'expérimentation 2. Lors de cette collecte de données sur le terrain, les enseignants et le chercheur avaient la charge de jauger le niveau d'information à délivrer aux élèves, en essayant de préserver autant que possible l'adidacticité des situations tout en leur permettant de progresser dans la tâche en cours.

Précisons également que lors de l'expérimentation 2, les experts étaient libres de déclencher les rétroactions dans l'ordre qu'ils souhaitaient. Ainsi, aucune limitation dans l'interface ne les contraignait dans leurs choix. Néanmoins, les rétroactions que nous avons conçues respectent, par construction, une certaine relation d'ordre : Jeu (1) < Notion (2) < Implémentation (3) < Solution (4). Par exemple, il n'est à priori pas pertinent de déclencher une rétroaction indiquant la notion à mettre en œuvre (2) si on a déjà donné son implémentation (3).

Nous verrons dans la suite que, pour les niveaux 2, 6, 7 et 8, nous disposons de trop peu d'observations pour pouvoir entrainer des classificateurs. Nous concentrons donc nos efforts sur les niveaux 1, 3, 4 et 5.

Le Tableau 6-5 donne la comparaison du niveau des rétroactions déclenchées par rapport à celle déclenchée juste avant selon les niveaux du jeu. En guise d'exemple, dans le niveau 3 du jeu, 85,9% des rétroactions déclenchées avaient un niveau supérieur à celle déclenchée précédemment (par exemple notion (2) puis solution (4)). En l'absence de déclenchement préalable (niveau 0), nous considérons que la rétroaction suivante est toujours de niveau supérieur.

Comparaison prec.	Niveau 1	Niveau 3	Niveau 4	Niveau 5
Niveau supérieur	86,7 %	85,9%	83,5%	89,6%
Niveau égal	8,9 %	11,6%	14,6%	9,3%
Niveau inférieur	4,4 %	2,5%	2%	1,1%

Tableau 6-5 – Comparaison du niveau des rétroactions déclenchées par rapport à celle déclenchée juste avant lors de l'expérimentation 2 selon les niveaux du jeu

Nous constatons que dans 83,5% à 89,6% des cas, les rétroactions déclenchées étaient d'un niveau supérieur à la précédente. Néanmoins, pour 8,9% à 14,6% des déclenchements, la rétroaction choisie est la même que la précédente. C'était généralement le fait d'élèves ayant fermé la bulle de dialogue du tuteur sans pouvoir exploiter pleinement la rétroaction et qui demandaient de l'afficher à nouveau. Enfin, de façon très marginale (1% à 4% des cas), les rétroactions déclenchées étaient d'un

niveau inférieur à la précédente. Il s'agissait la plupart du temps pour les enseignants d'indiquer une fonction de contrôle via la rétroaction de niveau jeu (1) malgré le fait que le concept visé (2) ou son implémentation (3) aient déjà été dévoilés. Illustrons en donnant l'exemple d'un élève qui demande comment ouvrir le coffre à la fin du niveau 1 alors qu'il a déjà mis en œuvre, suite à une rétroaction de niveau notion (2), une boucle for pour l'atteindre (voir Annexe G.1). Afin de simplifier la tâche des classificateurs, nous supprimons du jeu de données les traces des déclenchements de rétroactions de niveau égal ou inférieur pour ne garder que celles de niveau supérieur.

Bien que les rétroactions soient ordonnées, cela ne signifie pas qu'elles ont été déclenchées par les experts de façon incrémentale de la première à la dernière. Ainsi, la Figure 6-14 représente, sous forme de graphes, l'enchainement des rétroactions qui ont été données par les experts lors de l'expérimentation 2.

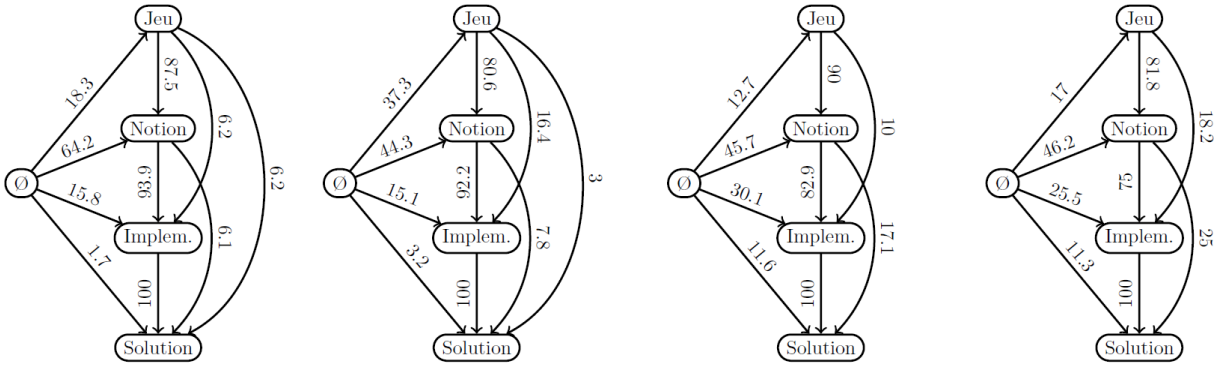


Figure 6-14 –Graphe d'enchaînement des rétroactions dans les différents niveaux du jeu lors de l'expérimentation 2. Pour un nœud donné, les arêtes sortantes représentent la proportion (en %) de chaque type de rétroactions suivantes choisies par les experts

Par exemple, pour le niveau 3 du jeu, lorsqu'une nouvelle rétroaction a été donnés suite une rétroaction de niveau jeu (1), il s'agit dans 80,6% des cas d'une rétroaction de niveau notion (2). Précisons que ces graphes sont trompeurs étant donné qu'ils représentent uniquement les rétroactions suivantes, et non l'absence de rétroactions suivantes (et donc le fait d'être la dernière rétroaction du niveau). Il ne faut par exemple pas conclure que tous les élèves ont été exposés à la dernière rétroaction (solution), mais plutôt que 100% des rétroactions déclenchées à la suite du type implémentation sont de type solution.

Comme nous pouvons le constater sur la Figure 6-14, les graphes sont orientés (les arêtes comportent des flèches) dans le sens des rétroactions croissantes. Cela signifie que les rétroactions sont toujours données en suivant la relation d'ordre. Cela est lié au fait nous n'avons gardé que les traces portant sur les déclenchements de niveau supérieur qui présentent 80% à 90% des rétroactions déclenchées. Nous remarquons également que les graphes sont quasiment complets (sans prendre en compte l'orientation, chaque sommet est relié directement à tous les autres sommets). Autrement dit, les experts ont choisi la première rétroaction parmi l'ensemble des possibles, puis peuvent décider de la suivante en « sautant » certains nœuds du graphe afin de proposer aux élèves le contenu le plus approprié pour répondre à leurs difficultés.

Dans ce contexte, l'enjeu pour les classificateurs est double lors d'une demande d'aide, il s'agit de :

– choisir la première rétroaction, qui n'est pas forcément de type contrôle (1), parmi les quatre existantes ;

– sélectionner correctement les rétroactions suivantes qui ne sont pas nécessairement celles d'ordre directement supérieur.

Rappelons que le classificateur prédit la rétroaction adéquate à la situation lorsqu'il est sollicité, mais c'est la politique de déclenchement décrite dans la section 6.5.1 qui décide de l'affichage ou non de la rétroaction en fonction des rétroactions précédemment fournies (pas de répétitions).

6.6.2 Création du jeu de données

Une étape préalable à l'entrainement des classificateurs est la création d'un jeu de données (caractéristiques + étiquette) à partir des traces d'activités que nous avons récoltées lors de l'expérimentation 2.

Ainsi, pour chaque déclenchement de rétroaction présent dans les traces, nous transcrivons l'historique de l'activité de l'élève depuis le début du niveau courant dans les caractéristiques. Afin de rendre ces éléments comparables entre les élèves, nous décidons de limiter l'influence du temps passé et du nombre de programmes lancés. Pour ce faire, nous ne conservons pas l'historique de manière cumulative mais nous déterminons des maximums, des moyennes par programme lancé, des temps moyens, et des taux en fonction du temps passé. Ces calculs qui diffèrent selon les activités tracées sont décrits dans le Tableau 6-6 qui donne le résumé des caractéristiques et de l'étiquette du jeu de données (le détail complet est disponible en Annexe M.1).

Cat.	Source	Description	Code	Card.
Jeu	LAN_PRO Progression niveau	Progression maximum dans la map du niveau (%)	GA_max_progression	1
	LAN_PRO Fonction de contrôle	Nombre moyen d'utilisations des fonctions de contrôle par programme exécuté	GA_avg_*_ctr_fun_used	14
	LAN_PRO Erreur dans le jeu	Nombre moyen d'erreurs dans le jeu par programme lancé	GA_avg_*_error	6
	LAN_PRO Raison perte niveau	Nombre moyen de pertes de niveau par programme lancé	GA_avg_*_lost	3
Concept	CON_CON Catégorie de contenu durée de consultation	Temps moyen de consultations par catégorie de contenu	CO_avg_*_disp_time	5
	COP_CON Identifiant du contenu	Taux de contenus copiés par rapport au temps passé	CO_rate_*_copied	17
	COL_CON	Taux de contenus collés par rapport au temps passé	CO_rate_pasted	1

	LAN_PRO Concepts implémentés	Nombre moyen de concepts implémentés par programme exécuté	CO_avg_*_impl	10
Exécution	LAN_PRO Issue du programme	Nombre moyen d'erreurs dans le code par programme lancé	EX_avg_*_errors	3
	LAN_PRO Issue du programme	Nombre moyen de programmes arrêtés par programme lancé	EX_avg_user_stopped	1
	LAN_PRO Issue du programme	Nombre moyen de programmes exécutés par programme lancé	EX_avg_executed	1
	MOD_VIT	Taux de changement de vitesse d'exécution par rapport au temps passé	EX_rate_speed_changer	1
Groupe	GEN_COM Identifiant de la partie	Identifiant de la partie permettant d'éviter la contamination des jeux de test	game_id	1
Étiquette	RET_DEC Type de rétroaction	Type de rétroaction déclenchée par l'enseignant	help_id	1

Tableau 6-6 – Résumé des caractéristiques et de l'étiquette du jeu de données

Précisons que les caractéristiques sont regroupées en trois catégories. Celles concernant le jeu comprennent la progression dans la map du niveau (18%, 76%, etc.), les fonctions de contrôle utilisées (avancer, sauter, etc.), les erreurs liées au jeu (déplacement interdit, ouverture du coffre sans clé, etc.), et les raisons en cas de perte du niveau (piques touchées, explosions, etc.).

La deuxième catégorie relève des concepts. Elle est constituée des temps de consultation des panneaux du mémo (var, condi, etc), des copiés-collés depuis les sous-parties du mémo (condi_1bran, condi_2bran ,etc.) et depuis l'éditeur de code, et des sous-concepts implémentés dans les programmes (var_affec, if_branch, etc.).

La dernière catégorie de caractéristiques est liée aux exécutions. Nous retrouvons les erreurs dans le code (too_many_lines, syntactic_error, etc.), les programmes exécutés ou arrêtés, et les changements au niveau du curseur de vitesse.

Précisons que les codes de ces 63 caractéristiques sont préfixés par leur catégorie (GA, CO, EX) puis par leur mode de calcul (max, avg, rate). Cela permettra une meilleure lisibilité lors de l'interprétation de l'importance relative des caractéristiques dans le processus de décision (voir plus loin section 6.6.9).

En plus des caractéristiques qui jouent le rôle de prédicteurs de l'étiquette, le groupe (identifiant de la partie) est une caractéristique d'ordre technique qui permet d'éviter la contamination des jeux de test et de validation au cours de l'entrainement des modèles (voir section 6.3.2).

Enfin, l'étiquette de chaque observation est le type de rétroaction qui a été déclenchée par l'expert en lien avec l'historique des activités décrites par les caractéristiques.

Précisons que même si cela est en mesure d'aider les classificateurs, nous ne tenons pas compte des rétroactions déclenchées dans le passé pour prédire la rétroaction courante (voir section précédente d'analyse du problème). Ces données risquent en effet de biaiser l'évaluation des modèles. Ainsi, en prenant l'exemple de l'évaluation finale sur le jeu de test (données nouvelles inconnues des classificateurs), cela reviendrait pour les classificateurs à s'appuyer sur des décisions forcément « correctes » qui ont été prises précédemment par les experts. Les modèles n'auront pas ces informations à disposition lors du déploiement en production étant donné que les décisions précédentes seront de leur fait, et comprendront inévitablement des erreurs.

Nous suivons maintenant les sept étapes de la méthodologie, détaillées dans la section 6.3.2, afin de procéder à l'entrainement et à l'évaluation des différents classificateurs.

6.6.3 Découverte et visualisation du jeu de données

Afin de donner une vision globale des différents jeux de données dont nous disposons selon les niveaux, la Figure 6-15 en résume les principales propriétés. Elle indique pour chaque niveau le nombre d'observations, le nombre total de caractéristiques, le nombre de caractéristiques pertinentes (qui varient), ainsi que la distribution des étiquettes (rétroactions déclenchées).

Commençons par constater que ces jeux de données ne sont pas équilibrés. Autrement dit, la distribution des étiquettes n'est pas homogène selon les classes. C'est par exemple la rétroaction portant sur les concepts (2) qui est majoritaire pour les niveaux 1, 3, 5, 6 et 7, tandis que le rétroaction donnant l'implémentation des concepts (3) est la plus fréquente dans les niveaux 2, 4 et 8. Le fait que les jeux de données soient déséquilibrés (imbalanced) a un impact sur le choix des métriques d'évaluation et sur les méthodes de sélection des jeux de test et de validation (voir section 6.3.2).

Ensuite, le nombre d'observations (qui correspondent à des déclenchements de rétroactions) varie d'un niveau à l'autre. Comme nous l'avons montré dans la section 6.4, certains niveaux, plus « faciles » que les autres, entrainent moins d'interventions des experts (niveau 2). D'autres niveaux ont été atteints seulement par la fraction des élèves les plus en réussite (niveau 6,7 et 8). Ils comportent donc proportionnellement moins de déclenchement de rétroactions. Il n'existe pas de règle clairement établie par la littérature concernant la taille minimum du jeu d'apprentissage pour une tâche de classification (Maxwell et al., 2018). Dans l'objectif d'avoir au moins une observation représentant chaque étiquette dans tous les blocs lors de la validation croisée, nous fixons le seuil à 150 observations. En effet, avec 24 observations par bloc ($150 \times 0{,}8 : 5 = 24$), et une proportion des étiquettes les moins fréquentes autour de 10%, cela devrait permettre de respecter cette contrainte. À partir de ce point, nous concentrons nos efforts sur les niveaux disposant d'au moins 150 observations, c'est-à-dire les niveaux 1, 3, 4 et 5. Pour les autres, nous serons contraints lors de la mise en production d'implémenter une politique de déclenchement indépendante de l'activité des élèves

basée sur une sélection incrémentale ou une marche aléatoire exploitant le graphe d'état reproduit dans la Figure 6-14.

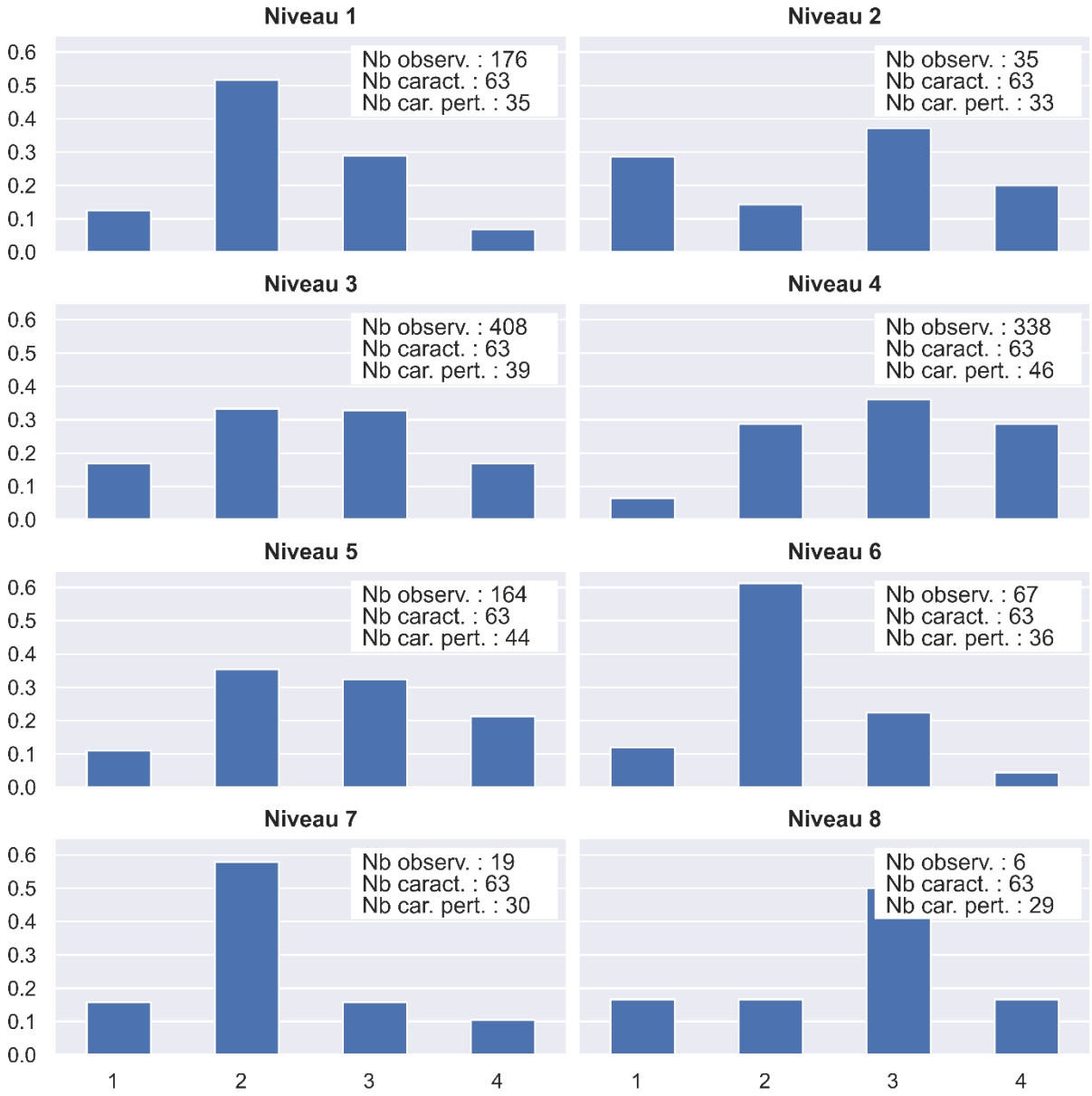


Figure 6-15 – Distribution des différentes étiquettes (barres bleues) et propriétés des différents jeux de données selon les niveaux du jeu

Comme nous l'avons vu précédemment, les 63 caractéristiques ne sont pas pertinentes pour tous les niveaux. Ainsi, certaines d'entre-elles, comme l'utilisation de certaines fonctions de contrôle qui ne sont pas autorisées, ont des valeurs nulles pour toutes les observations du jeu de données d'un niveau. Afin de réduire la dimension du jeu de données, nous supprimons ces caractéristiques et conservons entre 29 et 36 caractéristiques selon les niveaux.

Enfin, nous donnons en annexe M.2 les histogrammes de ces caractéristiques pertinentes par niveau. En parcourant ces représentations graphiques, nous ne remarquons pas de valeurs aberrantes (outliers) qu'il conviendrait de supprimer. Le choix de caractéristiques non cumulatives basées sur des maximums, des taux et des moyennes a l'avantage de lisser les traces induites par les comportements extrêmes

provoqués par les procédures de contournement didactique que nous avons décrites dans le chapitre précédent (voir section 5.8).

6.6.4 Constitution du jeu de test

Comme précisé dans la partie méthodologie (voir section 6.3.2), un jeu de test est constitué aléatoirement, il représente 20% du jeu de données. Il est mis de côté pour chaque niveau et servira à l'évaluation finale des modèles. Le reste des données forme le jeu d'apprentissage qui va être utilisé pour l'entraînement des différents modèles.

6.6.5 Préparation des données

Il s'agit ensuite de préparer les données à l'apprentissage automatique. Afin d'éviter la redondance d'information, nous supprimons d'abord une caractéristique par paire fortement corrélée ($|r| > 0.8$). Nous donnons en Annexe M.3 les matrices de corrélation entre les caractéristiques pour chaque niveau. Le Tableau 6-7 donne le détail des caractéristiques les plus corrélées par niveau.

Id	Niveau	Caractéristique 1 (supprimée)	Caractéristique 2
cor1	Niveau 1	CO_rate_var_modif_copied	CO_rate_var_creation_copied
cor2	Niveau 1	GA_avg_right_ctr_fun_used	GA_avg_walk_ctr_fun_used
cor3	Niveau 3	CO_rate_while_simple_copied	CO_rate_condi_1branch_copied
cor4	Niveau 3	CO_rate_pasted	CO_rate_code_editor_copied
cor5	Niveau 3	CO_rate_pasted	CO_rate_control_function_copied
cor6	Niveau 3	GA_avg_jump_ctr_fun_used	CO_avg_if_branch_impl
cor7	Niveau 3	GA_avg_attack_ctr_fun_used	CO_avg_if_branch_impl
cor8	Niveau 4	CO_avg_if_branch_impl	CO_avg_string_impl
cor9	Niveau 5	CO_rate_pasted	CO_rate_control_function_copied
cor10	Niveau 5	CO_avg_else_branch_impl	CO_avg_if_branch_impl

Tableau 6-7 – Caractéristiques fortement corrélées ($|r| > 8$) selon les niveaux

Certaines corrélations sont le fait de comportements atypiques concernant très peu d'élèves, comme l'utilisation de fonction de contrôle interdites[41] ou la copie du mémo concernant des concepts qui ne sont pas ciblés (cor1, cor3, cor6 et cor7). D'autres sont dues à l'implémentation conjointe de concepts et de fonctions de contrôle menant à la solution des niveaux (cor2, cor8 et cor10). Enfin, le collage de code est logiquement très corrélé à la copie de code (cor4,cor5 et cor9). Nous supprimons donc arbitrairement la première des caractéristiques de chaque couple dans le jeu de données.

Pour terminer, nous mettons les caractéristiques à l'échelle en utilisant la normalisation min-max qui réalise une projection des données entre 0 et 1. Cela permet d'améliorer les performances de certains algorithmes d'apprentissage automatique.

[41] Les fonctions de contrôle interdites ne pouvant pas être exécutées car elles provoquent des erreurs, elles devaient se trouver dans du code mort (donc non exécuté) mais cependant détecté à postériori dans l'arbre de syntaxe abstrait du programme

6.6.6 Choix du modèle

Les données étant préparées pour l'apprentissage automatique, nous entraînons maintenant plusieurs modèles par validation croisée sur le jeu d'apprentissage afin de sélectionner l'algorithme le plus efficace pour notre tâche de classification. Nous avons choisi cinq algorithmes classiques : Naive Bayses, Logistic Regression, SVM, Descision Tree et Random Forest. Nous testons également un classificateur aléatoire (Random Baseline) qui nous sert de base de comparaison (voir méthodologie section 6.3.2).

Nous représentons dans la Figure 6-16 les performances (précision, rappel et score F1) de ces algorithmes d'apprentissage automatique par niveau du jeu.

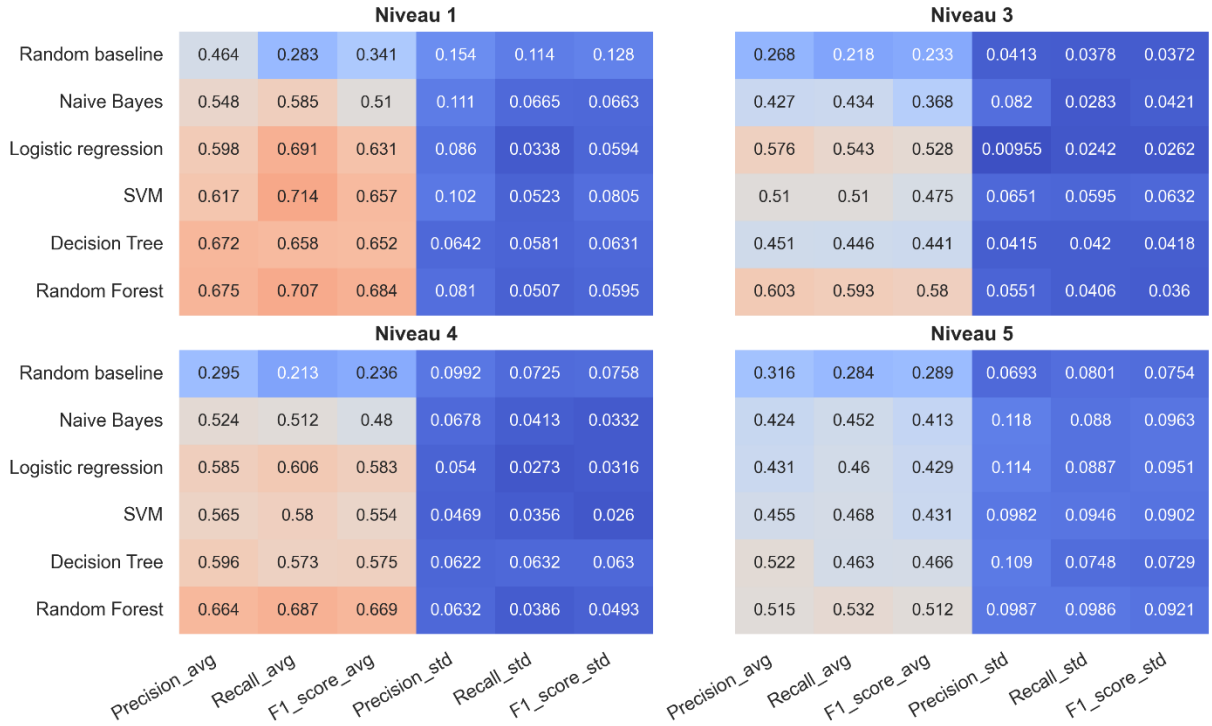


Figure 6-16 – Moyenne et écart-type des scores des différents modèles obtenus lors de la validation croisée sur le jeu d'apprentissage

Nous pouvons d'abord relever que tous les algorithmes performent mieux que le classificateur aléatoire. Cela valide la faisabilité de notre tâche de classification. Ensuite, la comparaison des scores révèle que les classificateurs Random Forest (RF) obtiennent les meilleures performances pour toutes les métriques, atteignant des scores F1 de 0,51 (niveau 5) à 0,68 (niveau 1) avec une balance précision-rappel équilibrée. Ces résultats sont par ailleurs nettement supérieurs aux performances de notre base aléatoire, nous retrouvons un facteur d'amélioration allant 1,8 à 2,8 selon les niveaux. Dans l'ensemble, ces résultats montrent que RF est le meilleur algorithme d'apprentissage automatique pour prédire les types de rétroactions dans tous niveaux du jeu Pyrates. Nous nous concentrons donc sur cet algorithme dans les sections suivantes.

6.6.7 Ajustement des hyperparamètres

L'algorithme RF étant choisi, il s'agit désormais d'ajuster au mieux ses hyperparamètres dans l'objectif d'améliorer ses performances. Les RF comportent de nombreux hyperparamètres (Probst et al., 2019). Après plusieurs essais infructueux,

nous avons focalisé nos ajustements sur deux d'entre eux : le nombre d'arbres de décision dans la foret aléatoire (n_estimator) et la profondeur maximale des arbres de décision (max_depth). Les valeurs testées et le rôle de ces paramètres sont précisés dans le Tableau 6-8.

Hyperparamètres	Rôles	Valeurs testées
n_estimators	Nombre d'arbres de décision dans la foret aléatoire	10, 50, *100*, 150, 200, 250, 300
max_depth	La profondeur maximale des arbres. Si None, alors les nœuds sont développés jusqu'à ce que toutes les feuilles soient pures ou jusqu'à ce que toutes les feuilles contiennent moins de deux échantillons	5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, *None*

Tableau 6-8 – Rôles et valeurs testées pour des hyperparamètres des RF, en italique la valeur par défaut

Nous avons procédé à l'ajustement des hyperparamètres en utilisant une recherche par grille (Grid Search) consistant à calculer le score F1 par validation croisée sur le jeu de données d'apprentissage pour toutes les combinaisons des valeurs testées. Les résultats de ces 91 validations croisées (7×13) sont présentés dans le Tableau 6-9 qui indique le meilleur score F1 obtenu ainsi que les valeurs des paramètres pour cet optimal. Nous indiquons également le gain obtenu par rapport aux paramètres par défaut utilisés dans l'étape de sélection de l'algorithme.

Niveau	Meilleur score F1	Score F1 initial	Gain score F1	n_estimators	max_depth
Niveau 1	0.72	0.68	0.04	50	15
Niveau 3	0.59	0.58	0.01	300	25
Niveau 4	0.69	0.67	0.02	250	20
Niveau 5	0.53	0.51	0.02	300	5

Tableau 6-9 – Scores et valeurs des hyperparamètres offrant les meilleures performances

Ces réglages fins des modèles offrent des gains non négligeables allant de un à quatre points de score F1. Nous retenons donc ces valeurs pour nos quatre modèles finaux.

6.6.8 Évaluation sur le jeu de test

Après avoir optimisé nos modèles, il nous reste à les tester sur des données auxquelles ils n'ont jamais été confrontés, celles contenues dans le jeu de test. Nous reproduisons dans la Figure 6-17 les matrices de confusion ainsi que les différents scores obtenus selon les niveaux en réalisant des prédictions à partir de ces nouvelles données.

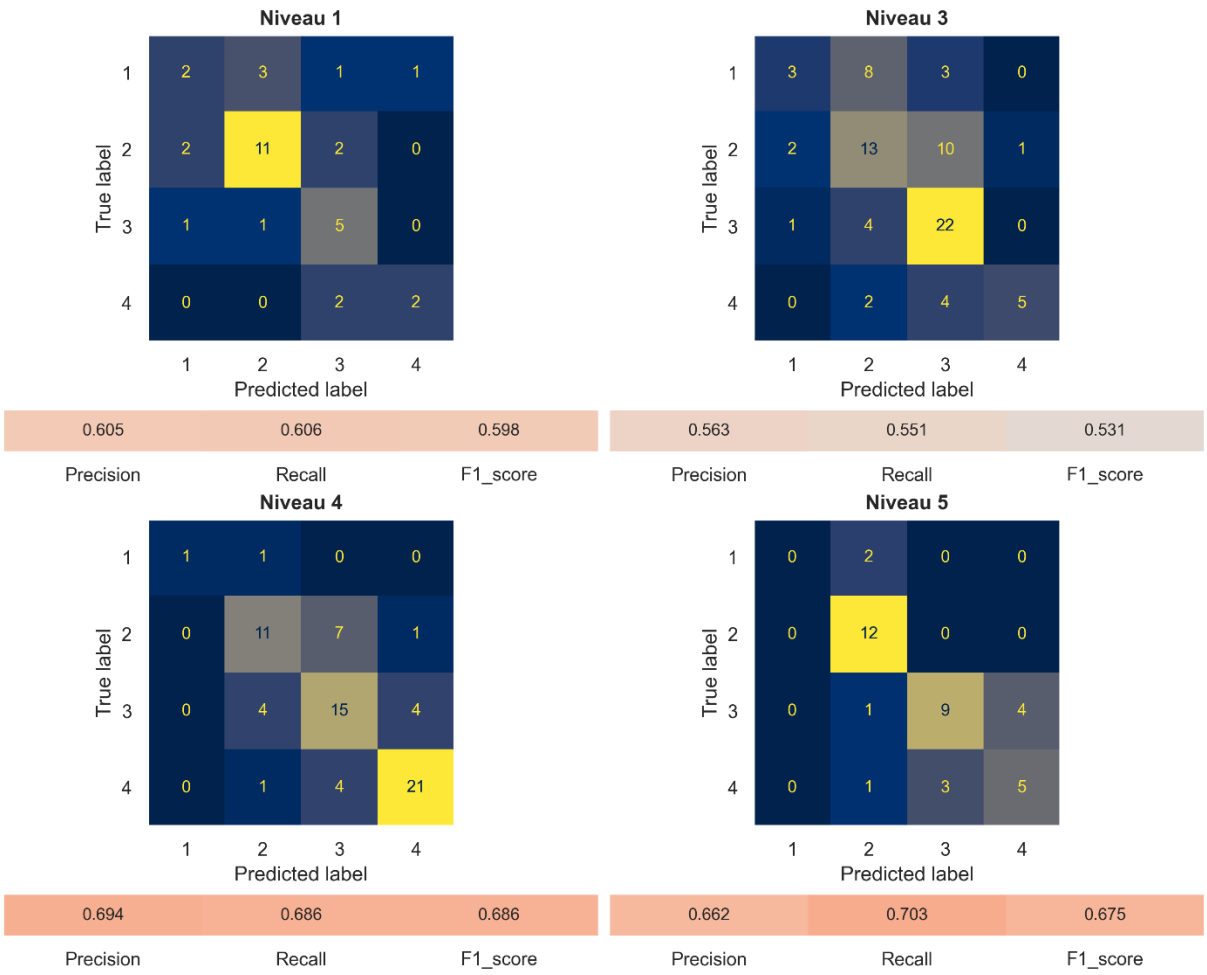


Figure 6-17 – Matrices de confusion et scores obtenus lors de l'évaluation des modèles finaux sur le jeu de test selon les niveaux du jeu

Remarquons d'abord que les différents scores calculés sont assez proches de ceux obtenus par validation croisée sur le jeu d'apprentissage. Ces résultats sont toutefois moindres pour le niveau 1 et meilleurs pour le niveau 5.

Ensuite, en examinant les matrices de confusion, nous pouvons affiner nos analyses concernant les performances des modèles pour chacun des quatre types de rétroactions.

Premièrement, nous donnons dans le Tableau 6-10 la précision (proportion des rétroactions prédites pertinentes) et le rappel (proportion des rétroactions pertinentes prédites) spécifiques pour chaque type de rétroactions pour tous les niveaux confondus.

	Rétroaction 1 Jeu	**Rétroaction 2 Concept**	**Rétroaction 3 Implémentation**	**Rétroaction 4 Solution**
Précision	0,24	0,65	0,72	0,66
Rappel	0,5	0,63	0,58	0,75

Tableau 6-10 – Précision et rappel spécifique par type de rétroactions tous niveaux confondus

Nous pouvons voir que les rétroactions 2 (concept), 3 (implémentation) et 4 (solution) sont globalement plus faciles à prédire avec une précision qui varie entre 65% et 72% et un rappel qui fluctue entre 58% et 75%. En revanche, la rétroaction 1 (jeu) génère davantage d'erreurs de classification et la précision des modèles est particulièrement mauvaise la concernant (24%). Cela veut dire que cette rétroaction pourrait être souvent déclenchée de façon inadéquate (faux positif). Ces erreurs n'ont à priori pas de conséquences néfastes sur l'apprentissage compte tenu du fait que cette rétroaction ne diminue pas l'adidacticité des situations (voir section 6.5.2), et qu'elle est également la moins fréquente (voir la distribution des étiquettes dans la Figure 6-15).

Deuxièmement, les erreurs de classification sont généralement peu sévères. Ainsi, lorsqu'un modèle ne prédit pas correctement une rétroaction, il recommande dans la plupart des cas une rétroaction proche de celle attendue. Ainsi, parmi les erreurs de classification, 84% concernent un décalage d'un niveau (par exemple 3 à la place de 2), 15% se rapportent à un écart de deux niveaux (par exemple 4 à la place de 2), et 1% relève d'une distance de 3 niveaux (par exemple 1 à la place de 4).

Troisièmement, nous évaluons l'impact des erreurs de prédiction sur le niveau d'adidacticité des situations proposées aux élèves. Le fait de prédire une rétroaction d'un plus haut niveau que celui nécessaire (ce qui correspond aux erreurs situées dans les demi-matrices supérieures) diminue inutilement l'adidacticité des situations, ce qui peut avoir un effet défavorable sur l'apprentissage des élèves. Ces prédictions erronées qui engendrent la diffusion de « trop » d'informations représentent 22% du total des prédictions faites sur le jeu de test (48/218). Nous pouvons donc considérer qu'un cinquième des prédictions peuvent être néfastes aux apprentissages des utilisateurs de Pyrates, ce qui est nettement plus performant qu'un classificateur aléatoire qui prédit 37,5% de rétroactions supérieures.

Au final, les rétroactions de niveau 2 (concept), 3 (implémentation) et 4 (solution) sont plus faciles à prédire que la rétroaction de niveau 1 (jeu) qui connait beaucoup d'erreurs ayant cependant peu d'impact. Enfin, les erreurs de classification sont peu sévères mais peuvent poser problème pour un cinquième des prédictions, ce qui est une performance significativement supérieure à celle que notre référence aléatoire.

6.6.9 Importance des caractéristiques

Afin de mettre en évidence les caractéristiques les plus prédictives pour la sélection des rétroactions, et ainsi améliorer l'explicabilité de nos modèles, nous présentons dans la Figure 6-18 les dix caractéristiques les plus importantes pour chaque niveau. Nous utilisons pour cela la méthode de diminution moyenne de l'impureté (MDI) propre à l'algorithme RF. Dans cette métrique, l'importance relative d'une caractéristique s'apprécie en évaluant dans quelle mesure les nœuds de l'arbre qui utilisent cette caractéristique réduisent leur impureté (coefficient de Gini) en moyenne à travers tous les arbres de la forêt (Géron, 2019, p. 198).

Rappelons que les noms des caractéristiques sont préfixés par l'une des trois catégories décrites dans la section 6.6.4 : jeu (GA), concept (CO) ou exécution (EX).

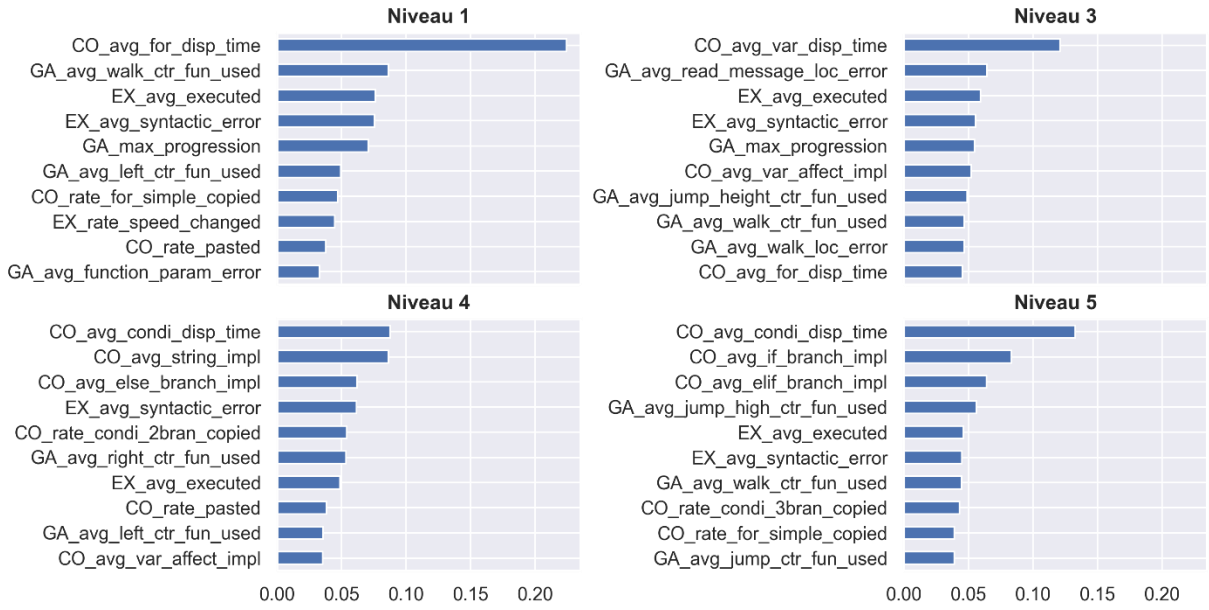


Figure 6-18 – Top 10 des caractéristiques les plus importantes d'après la méthode MDI selon les niveaux du jeu

Nous constatons d'abord que les trois catégories de caractéristiques apparaissent dans les quatre classements, ce qui indique qu'elles sont toutes utiles et complémentaires pour prédire le type de rétroaction à sélectionner.

Ensuite, les caractéristiques liées aux concepts (CO_*) sont toujours en rapport avec les concepts ciblés dans les niveaux (niveau 1 : for ; niveau 3 : variable ; niveau 4 : branches conditionnelles if-else ; niveau 5 : branches conditionnelles if-elif-else). De plus, toutes les interactions avec les concepts sont exploitées : affichage (CO_*_disp_time), copie (CO_*_copied), et mise en œuvre (CO_*_impl). L'évaluation de la consultation, de la copie et de l'utilisation des concepts visés dans un niveau est un aspect essentiel dans le choix des aides à apporter aux élèves. Nous pouvons affirmer que les classificateurs que nous avons entrainés en tiennent compte.

En ce qui concerne les caractéristiques liées au jeu (GA_*), nous pouvons remarquer dans les niveaux 1 et 3 que celles relatives aux erreurs de jeu (GA_*_error) et à la progression (GA_max_progession) ont un fort pouvoir prédictif. De surcroit, pour tous les niveaux, les caractéristiques relatives aux fonctions de contrôle utilisées (GA_*_fun_used) apparaissent dans le top 10. L'avancement dans le niveau, les erreurs de jeu et l'utilisation des fonctions de contrôle du personnage sont nécessairement des critères importants dans l'évaluation du niveau d'aide à accorder aux élèves. Nous pouvons constater que ces critères sont pris en considération par les modèles que nous avons conçus.

Enfin, bien que les caractéristiques d'exécution (EX_*) apparaissent moins dans les classements, le taux d'erreur syntaxique (EX_avg_syntactic_error) et le taux de programmes exécutés sans erreurs (EX_avg_executed) sont des bons prédicteurs pour tous les niveaux. Remarquons par ailleurs que ces deux caractéristiques sont assez fortement corrélées (niveau 1 : -0,52 ; niveau 3 : -0,46 ; niveau 4 :-0,52 ; niveau 5 : -0,34) (voir Annexe M.3). Cela a sans doute pour conséquence de minimiser leur importance relative dans ces classements. Les taux d'erreurs syntaxiques et de programmes corrects ont donc une forte influence sur la nature des aides à fournir aux

élèves. Ces informations sont probablement utiles pour prédire le déclenchement des rétroactions de niveau implémentation (3).

6.6.10 Synthèse de la section

Q6.3 : Quelle rétroaction épistémique fournir aux élèves lorsqu'ils demandent de l'aide afin de préserver autant que possible l'adidacticité des situations tout en leur permettant d'avancer ?

En nous basant sur les données étiquetées par des experts lors d'une deuxième expérimentation dans les classes, nous avons entraîné des modèles d'IA afin d'effectuer une tâche de classification.

Bien que les **rétroactions** respectent une certaine **relation d'ordre** de par leur conception, l'**enjeu** des **prédictions** consistait à :

- choisir la **première rétroaction** parmi les **quatre possibles** ;
- choisir les **rétroactions suivantes** de façon **non** nécessairement **incrémentale**.

Pour cela, en suivant la méthodologie proposée par Géron (2019) nous avons :

- créé un **jeu de données par niveau** comportant des **caractéristiques non cumulatives** transcrivant l'**historique** de l'activité de l'élève depuis le **début** du **niveau** selon trois catégories (**jeu**, **concept** et **exécution**), et ayant pour **étiquette** la **rétroaction** qui a été déclenchée par l'**expert** ;
- constaté que **seuls les niveaux 1, 3, 4 et 5** comportaient assez d'observations (>150) **pour l'apprentissages** des modèles ;
- entrainé **cinq** modèles à l'aide d'**algorithmes** de **classification** différents en plus d'une base aléatoire ;
- retenu l'algorithme **Random Forest**, ayant des scores F1 entre **2 et 3 fois supérieurs** à la **référence aléatoire**, pour les quatre niveaux ;
- ajusté les **hyperparamètres** des modèles afin de **gagner quelques points** de performance ;
- évalué les modèles finaux sur le jeu de test et constaté que les rétroactions de niveau 2 (**concept**), 3 (**implémentation**) et 4 (**solution**) étaient les plus **faciles à prédire** à la différence de celles de niveau 1 (**jeu**) qui connaissent **beaucoup** d'**erreurs** (non néfastes à l'apprentissage), et que les **erreurs** de classification sont peu **sévères** mais peuvent **poser problème** pour **un cinquième** des prédictions (réduction inutile de l'adidacticité) ce qui est **nettement plus performant** que la **référence aléatoire** ;
- évalué l'**importance relative** des caractéristiques (MDI) afin d'améliorer l'**explicabilité** des modèles et constaté que **toutes les catégories** de caractéristiques sont **utiles** aux prédictions, que celles en lien avec les **concepts** permettent d'évaluer la **consultation**, la **copie** et l'**utilisation** des concepts ciblés, que celles en rapport avec le **jeu**

donnent des informations sur l'**avancement** dans le niveau, les **erreurs de jeu** et l'utilisation des **fonctions de contrôle**, et enfin que celles ayant trait à l'**exécution** renseignent sur les **erreurs sémantiques** et les **programmes corrects** exécutés.

Dans ce chapitre, nous avons d'abord évalué l'avancée des élèves dans le jeu ainsi que leur autonomie vis-à-vis des enseignants lors de la première expérimentation. Nous avons ensuite proposé une évolution de l'application visant à accroitre cette autonomie par l'intermédiaire de rétroactions épistémiques fournies automatiquement en fonction de l'activité des élèves dans l'environnement. Nous avons conçu quatre types de rétroactions épistémiques par niveau en nous appuyant sur les données de la première expérimentation. Ces rétroactions, délivrées par un tuteur, sont sélectionnées par des modèles d'intelligence artificielle mettant en œuvre des classificateurs. Nous avons entrainé ces modèles à partir de nouvelles données qui ont été étiquetées par des experts lors d'une deuxième expérimentation sur le terrain. Les résultats indiquent que ces classificateurs prédisent les rétroactions deux à trois fois mieux que si elles étaient choisies au hasard, ce qui est prometteur dans l'objectif d'épauler les enseignants dans l'accompagnement des élèves. Néanmoins, dans 20% des cas, des prédictions erronées peuvent être défavorables aux apprentissages des élèves. Enfin, nous avons montré que les différents types de caractéristiques qui résument l'historique de l'activité des élèves ont tous une influence sur les prédictions, cela valide notre conception et permet d'améliorer l'explicabilité de notre système IA.

Nous présentons dans le chapitre suivant la conclusion générale de la thèse qui comprend un résumé des résultats et des contributions, les limitations des travaux menés, ainsi que des perspectives de recherches à mener à la suite de la thèse.

Chapitre 7

Conclusion et perspectives

Ce dernier chapitre conclut cette thèse et esquisse de nouvelles orientations de recherche. Nous rappelons d'abord les résultats obtenus en répondant aux six questions de recherche présentées en introduction (section 7.1), nous présentons ensuite les contributions de nos travaux pour la recherche et les praticiens sur le terrain (section 7.2). Par la suite, nous décrivons les limites dues à nos choix méthodologiques (section 7.3), et terminons par ouvrir quelques perspectives de recherche (section 7.4).

7.1 Rappel des résultats

Nous rappelons dans cette section les résultats que nous avons établis dans cette thèse pour chaque question de recherche présentée dans l'introduction.

Q1 : Qu'est-ce que l'informatique en général, et plus particulièrement la science informatique et la programmation informatique ? Quels sont les concepts fondamentaux de la programmation informatique ?

Nous avons montré que l'informatique comporte quatre niveaux hiérarchiques : la théorie informatique (concepts, principes et propriétés théoriques), les techniques informatiques (procédés méthodiques fondés sur la théorie), les outils informatiques (artefacts matériels ou logiciels mettant en œuvre les techniques), et les usages informatiques (utilisation des outils). La science informatique est composée de la théorie et des techniques informatiques.

Plus précisément, la science informatique est constituée de dix domaines de connaissance : calculabilité et complexité, mathématiques discrètes, architecture des machines, systèmes d'exploitation, réseaux, programmation, gestion des données, intelligence artificielle, sécurité, interactions homme-machine.

Le domaine de connaissance « programmation » comprend l'utilisation d'un langage de programmation. Un tel langage est défini comme un média qui permet de décrire des algorithmes calculables par des machines, et compréhensibles par des humains par le biais de programmes constitués d'instructions suivant une certaine syntaxe et ayant une certaine sémantique définies de façon formelle.

Les langages de programmation permettent différents niveaux d'abstraction de la machine : langage machine, langage d'assemblage, et langage de haut niveau. Ces langages sont classés selon leurs caractéristiques dans différents paradigmes : la programmation impérative (procédurale ou orientée-objet) et la programmation déclarative (logique ou fonctionnelle). Ils dérivent tous d'un ensemble de langages pionniers conçus dès les années 50 et 60 : Fortran, Simula ou Lisp.

Enfin, l'usage de la programmation impérative met en jeu neuf concepts fondamentaux : variable, types primitifs, structure conditionnelle, structures de

boucle, entrées-sorties des programmes, fonction et paramètres, récursivité, types composites simples, et types composites complexes.

Q2 : Quelle est la place de la programmation informatique dans les programmes scolaires français à la transition collège-lycée ? Quelles sont les modalités d'enseignement prescrites ? Quelles sont les conséquences pour les élèves qui passent du collège au lycée ?

La programmation informatique est enseignée en mathématiques en troisième et seconde, ainsi qu'en technologie en troisième, et en SNT en seconde. Cette présence est légitimée dans les textes officiels par les éléments suivants : les algorithmes sont historiquement liés aux mathématiques, les artefacts techniques utilisés dans la société sont les objets d'étude de la technologie, et les SNT portent sur la science informatique dont la programmation est un élément central.

La programmation informatique est implantée de manière différente dans les programmes d'enseignement de ces disciplines, elle occupe différents habitats (Artaud, 1997). Ainsi, en mathématiques en troisième, elle possède un domaine dédié n'ayant aucune interaction avec les autres domaines du programme, l'objectif est l'initiation aux concepts fondamentaux à travers des activités ludiques. En technologie, elle est incluse dans un domaine dédié et opère quelques interactions avec les autres domaines du programme, l'objectif est de programmer des systèmes techniques en réponse à un problème posé en termes de besoins. En mathématiques en seconde, elle fait l'objet d'un domaine dédié et connait d'importantes interactions avec tous les autres domaines, l'objectif est double, approfondir les concepts fondamentaux et fournir un outil au service des mathématiques. Enfin en SNT, elle est diluée dans les autres domaines du programme, elle a donc d'importantes interactions avec eux, l'objectif étant de proposer des activités qui illustrent les différents domaines de la science informatique.

Les programmes d'enseignement comportent une partie notionnelle qui répertorie les concepts fondamentaux ciblés. Il s'agit en troisième des variables, des structures conditionnelles, des boucles for et while, et de la programmation évènementielle. En seconde, s'ajoutent le typage des variables et les fonctions.

Ces concepts ont une certaine utilité dans les activités proposées dans les documents d'accompagnement, on parle de niche (Artaud, 1997). Ainsi, les variables de type primitif servent de façon utilitaire alors que les types composites et les objets permettent d'accéder aux fonctionnalités des bibliothèques externes. Les conditionnelles permettent d'effectuer des traitements contingents dans tous types d'activités. Les boucles for ont pour rôle de réitérer des traitements et de parcourir des collections indexées, et les boucles while permettent de détecter la fin d'un traitement itératif ou de faire une « boucle infinie » d'attente d'un évènement.

Les indications des textes officiels portent également sur les modalités d'enseignement. Nous avons classé ces modalités suivant quatre indicateurs : contenus, activités, dispositifs, et artefacts (Reuter, 2004). Au sujet des contenus, toutes les disciplines opèrent la distinction entre algorithme et programme à l'exception des mathématiques en troisième. Pour les activités, la démarche ciblée est plutôt la résolution d'exercices et de problèmes en mathématiques, alors que la technologie et les SNT proposent des démarches d'investigation et des projets plus

importants. Les types d'activités de programmation sont plus divers en mathématiques qu'en technologie et en SNT. Enfin, le mode de programmation est orienté vers les essais-erreurs en blocs en mathématiques en troisième, alors qu'il s'agit de passer par une phase de formalisation algorithmique dans les autres disciplines et niveaux. Concernant les dispositifs, les lieux de programmation sont le laboratoire pour la technologie, et la salle de classe ou la salle informatique pour les autres disciplines. L'organisation de l'espace varie entre la classe entière et les petits groupes d'élèves. Les élèves peuvent être amenés à interagir entre eux lors de tâches collaboratives. Enfin, au sujet des artefacts, les supports de la programmation (artefacts-supports) sont les langages de blocs en troisième et les langages textuels au lycée, et les machines d'édition sont les ordinateurs et tablettes pour toutes les disciplines. Les programmes mis au point doivent permettre de contrôler des objets programmables en technologie et SNT (artefacts-cibles).

Au bilan, les discontinuités que l'on retrouve dans les programmes de troisième et de seconde sont susceptibles d'engendrer des difficultés pour les élèves. Ainsi, l'étude des habitats révèle des différences, avec une programmation autonome permettant des activités ludiques et concrètes au collège, et des activités en lien avec les programmes, plus scolaires et abstraites au lycée. Cela est en mesure d'atténuer l'engagement et la motivation des élèves. Les concepts de programmation de variable, structure conditionnelle et boucles sont ciblés au collège comme au lycée, leur transposition de la modalité bloc vers la modalité texte peut être source de difficultés pour les élèves. En comparaison avec la troisième, les niches des concepts de structures de contrôle deviennent plus complexes en seconde, la programmation évènementielle et parallèle est abandonnée pour laisser la place à une méthode linéaire et séquentielle, et des bibliothèques externes sont employées afin d'étendre les possibilités des langages. Pour finir, les modalités d'enseignement diffèrent surtout concernant les artefacts à utiliser pour produire des programmes. Le passage de la programmation par blocs à la programmation textuelle cache une diversité de changements à même de poser des difficultés cognitives.

Q3 : Quelles sont les différences entre les langages de programmation Scratch et Python ? Quelles sont les conséquences pour les apprenants qui transitent de l'un à l'autre ?

Les langages de programmation Scratch et Python connaissent des différences inhérentes à la fois aux langages eux-mêmes, mais également à leur environnement d'édition.

Pour commencer, les paradigmes de programmation mis en œuvre ne sont pas les mêmes. Scratch permet de la programmation orientée objet, événementielle et parallèle, alors que Python est utilisé au lycée de manière procédurale et séquentielle. Le passage de l'un à l'autre nécessite des adaptations cognitives.

L'implémentation des différents concepts de programmation est en partie différente. Ainsi, l'étape de déclaration des variables nécessaire en Scratch n'est pas faite en Python. De plus, les variables ne sont pas typées en Scratch au contraire de Python qui distingue trois types primitifs différents. Au sujet des itérations, les boucles for disposent en Python de variables de boucles qui n'existent pas en Scratch, et la condition d'arrêt des boucles while est inversée entre les deux langages. Concernant les

fonctions, elles agissent uniquement par effet de bord en Scratch alors qu'elles permettent de retourner des valeurs en Python. Ces différences conceptuelles peuvent constituer des obstacles épistémologiques (Bachelard, 1938) en mesure de mettre les apprenants en difficulté.

On retrouve également des variations au niveau des registres sémiotiques (Duval, 1993). Ainsi les scripts Scratch sont constitués d'éléments graphiques de différentes formes et couleurs ainsi que de locutions françaises, alors que les programmes Python sont formés de mots-clés en anglais et de symboles typographiques. Le passage d'une forme à l'autre peut être particulièrement problématique pour les boucles (faible congruence). De plus, la forme textuelle des programmes Python est délicate à appréhender pour les novices dans la mesure où elle est dense, peu intuitive, éloignée du langage naturel dans ses formulations, et dans une langue étrangère (sauf pour les anglophones).

Les erreurs de programmation constituent une autre divergence entre les deux langages. Pour des raisons de simplification d'usage, Scratch ne produit aucun message d'erreur, et les variables y sont faiblement typées. De son côté, Python signale lors de l'interprétation tous les types d'erreurs (lexicales, syntaxique et sémantique) en maintenant un typage fort et dynamique pour les variables. Ces écarts mènent les débutants en Python à commettre de nombreuses erreurs, et à être confrontés à des messages d'erreurs mal formulés et déroutants qu'ils ont du mal à interpréter.

D'autres différences concernent plutôt les environnements d'édition des programmes. Ainsi, Scratch dispose d'un catalogue de commandes prenant la forme d'un menu persistant listant tous les blocs classés par thèmes et par concepts. Il permet aux débutants de découvrir ou de se souvenir des concepts et des fonctionnalités du langage. L'environnement standard fourni avec Python met à disposition des programmeurs la documentation technique complète du langage ainsi qu'une complétion automatique des mots-clés. Destinées aux utilisateurs expérimentés, ces fonctionnalités permettent difficilement de soutenir la découverte du langage par les débutants. Cela oblige par conséquent les apprenants à mémoriser les fonctionnalités du langage et leur syntaxe.

La composition des programmes est également différente. En Scratch, les scripts sont constitués en glissant des blocs depuis un catalogue puis en les assemblant de manière guidée et contrainte. En Python, les programmes sont rédigés au clavier en étant potentiellement assisté par la complétion et l'indentation automatique. La création et le maintien de la structure globale des programmes Python est source d'erreurs et le processus mécanique de saisie au clavier peut représenter un défi cognitif et moteur pour les jeunes apprenants.

Enfin, la visibilité et le contrôle de l'exécution sont distincts entre les deux modalités. La modalité Scratch met en valeur le code exécuté et affiche l'état des variables. Il rend également possible l'exécution directe des blocs à tout moment et permet de contrôler la vitesse d'exécution. Bien que plus difficile d'accès pour les débutants, la modalité Python offre également certaines de ces fonctionnalités. Le débogueur signale la ligne en cours d'exécution, affiche le contenu des variables et propose une exécution pas à pas. Ces fonctionnalités de suivi permettent aux novices de mieux comprendre comment les programmes sont exécutés, de plus, l'exécution

directe des blocs Scratch rend les exécutions plus concrètes et aide à se positionner dans le temps.

Q4 : Quelles sont les pratiques déclarées des enseignants concernant l'enseignement de la programmation informatique en troisième et en seconde ? Quelles répercussions pour la création d'une ressource d'enseignement ?

Lors d'une enquête en ligne que nous leur avons soumis, des enseignants ont pu fournir des informations sur leurs pratiques d'enseignement en lien avec les contenus, les activités, les dispositifs et les artefacts.

Concernant les contenus, la distinction entre les notions d'algorithme et de programme est davantage faite auprès des élèves de seconde que de ceux de troisième. Plus de la moitié des enseignants ne se concertent pas avec les professeurs de leur binôme disciplinaire au sujet de l'enseignement de la programmation, lorsque qu'une coordination est mise en place, les enseignants de mathématiques n'endossent pas particulièrement la charge de l'introduction conceptuelle.

Pour ce qui est des activités, quelle que soit leur discipline, les enseignants consacrent en moyenne environ dix séances par an à la programmation informatique. Lors de la préparation de leurs séances, les enseignants disent se référer beaucoup aux programmes d'enseignement mais peu aux ressources d'accompagnement. Ils mettent à profit les manuels scolaires et les cahiers d'activités (surtout en mathématiques), les ressources mises à disposition en ligne par leurs pairs, et des sites Internet offrant des activités « clés en main » (surtout en seconde). Les activités de programmation déclarées sont fortement en lien avec les autres domaines du programme d'enseignement dans toutes les disciplines, et ceci ne semble pas dépendre des préconisations officielles. Les démarches classiques de résolutions d'exercices et de problèmes sont majoritaires (surtout en mathématiques), alors que les démarches plus actives de type projets sont très présentes en technologie, modérément en SNT, et peu en mathématiques. Dans toutes les disciplines, les types d'activités de programmation sont très variés, il s'agit de tester, expliquer, compléter ou modifier un programme existant, la création complète d'un programme est également très présente (moins en seconde). Les activités collaboratives sont rares en mathématiques, et plus courantes en technologie et en SNT. Ces usages se concentrent sur le partage des tâches et la comparaison de solutions. Le mode de programmation mis en œuvre par les élèves de troisième en mathématiques est très majoritairement la programmation directe par essais-erreurs, ce mode est partagé dans les autres disciplines avec une formalisation préalable des algorithmes (organigramme ou pseudo-code), suivie d'une traduction en langage de programmation. Enfin, les enseignants de toutes les disciplines disent proposer peu d'activités débranchées.

Au sujet des dispositifs, les enseignements de programmation ont lieu majoritairement en salle informatique et en salle de classe ordinaire pour les mathématiques, quasi exclusivement dans le laboratoire de technologie pour la technologie, et en salle informatique ou dans des laboratoires de sciences pour les SNT. Les effectifs par classe pour nos répondants sont en moyenne de 26,2 en troisième et de 29,9 en seconde. La grande majorité des enseignants de seconde (surtout en mathématiques) ont la possibilité de donner des heures de cours dédoublées en demi-groupe, ce qui est plus rare en troisième (en particulier en technologie). Enfin,

concernant l'organisation de l'espace et les interactions, les élèves de mathématiques et de SNT travaillent majoritairement de manière individuelle ou en binôme, et les enseignants se trouvent au tableau devant le groupe classe. En technologie, les activités de groupe ou en binômes sont majoritaires pour les élèves, et les enseignants sont plus souvent à leur contact direct.

Du côté des artefacts, les machines de développement sont très majoritairement des ordinateurs de bureau dans toutes les disciplines. On retrouve également des classes mobiles constituées plutôt de tablettes en troisième, et d'ordinateurs portables en seconde. La calculatrice est également un support de programmation répandu en seconde, en particulier en mathématiques. Le langage de programmation utilisé en mathématiques au collège est presque exclusivement Scratch, on distingue néanmoins une forte diversité d'autres langages de blocs en technologie. En seconde, Python domine largement les usages et est complété en SNT par le couple HTML/CSS. Toujours en seconde, en mathématiques, les élèves utilisent des librairies logicielles de calcul, de géométrie et de visualisation de données. En SNT, les bibliothèques de traitement d'images, de cartographie et de traitement de données sont les plus employées. Les environnements de programmation les plus répandus sont les éditeurs pédagogiques simplifiés tels que EduPython, les éditeurs de texte basiques sont également utilisés (surtout en SNT), ainsi que les éditeurs en ligne comme les notebooks Jupiter.

Ces informations donnent des indications pour la conception d'une ressource d'enseignement de la programmation informatique à la transition-collège lycée. Une telle ressource devra pouvoir être utilisée en classe de mathématiques et de SNT pour une durée de deux séances d'une heure maximum. Elle pourra prendre la forme d'une application en ligne développée prioritairement pour les ordinateurs fixes et portables, et les navigateurs Internet Mozilla Firefox et Google Chrome. Elle devra permettre l'écriture complète de programmes, tout en proposant un dispositif d'édition à même de soutenir la transition Scratch-Python. Enfin, elle devra favoriser l'autonomie des élèves dans un contexte de classes chargées.

Q5 : Comment concevoir une ressource d'enseignement de la programmation informatique sous la forme d'une application en ligne basée sur un jeu sérieux ? De quelle manière les élèves s'emparent-ils de cette application sur le terrain ?

Les différents niveaux du jeu sérieux ont d'abord été conçus sur le modèle des situations adidactiques. Les concepts de programmation ciblés sont communs aux programmes du collège et du lycée : variable, conditionnelle, boucle for et boucle while. L'objectif étant d'accompagner le changement de registres sémiotiques des blocs vers le texte. Le jeu sérieux développé est un jeu de plateforme sur le thème des pirates. Cet univers n'étant pas neutre du point de vue du genre, les scènes de violence et de combat ont été limitées. De plus, afin de favoriser leur identification, les joueurs et les joueuses ont le choix d'incarner deux personnages mixtes.

L'objectif commun aux huit niveaux du jeu est de collecter une clé permettant d'ouvrir un coffre au trésor. Les différentes maps des niveaux ont ensuite été conçues pour rendre nécessaire l'implémentation des concepts visés : parcours répétitifs ou structurés avec limitation des lignes pour les boucles for, rétention d'information pour les variables, parcours aléatoires et présence de fonction de collecte de données pour

les conditionnelles, et parcours répétitifs et aléatoires avec fonction de collecte de données pour les boucles while.

Les différentes situations que constituent les huit niveaux du jeu ont ensuite été évaluées sur le terrain. Il en ressort qu'elles respectent globalement les conditions d'adidacticité. En effet, les joueurs ont la capacité de s'engager dans les niveaux par le biais de procédures de base constituées uniquement de fonctions de contrôle. De plus, la conception des niveaux est très efficace pour conduire à la mise en œuvre des concepts de boucle for et de variable. Cependant, elle n'est pas suffisante pour imposer systématiquement l'utilisation des concepts basés sur des tests (conditionnelle et boucle while). Cependant, un ajustement des variables didactiques ainsi que le bannissement du point-virgule dans les programmes devrait être en mesure de limiter l'usage des stratégies de contournement didactique et d'équivalence conceptuelle.

Un deuxième aspect de la conception est l'intégration dans l'environnement de l'application de certaines caractéristiques des logiciels d'édition de blocs afin de pouvoir profiter de leurs avantages tout en programmant en Python. Ainsi, un mémo Python qui présente de façon synthétique les différents concepts visés dans le jeu sérieux a été créé. Il donne des explications et fournit des modèles génériques d'implémentation ainsi que des exemples illustratifs. Ces programmes sont donnés dans les deux modalités Scratch et Python afin de faciliter la transition du point de vue sémiotique. Un bouton copier a été ajouté dans tous les extraits de code afin d'encourager la pratique du copier-coller et de limiter la saisie au clavier et les erreurs syntaxiques. L'intégration d'un analyseur syntaxique issu de la recherche fournissant des messages d'erreur adaptés aux débutants (vocabulaire moins technique et en français) a pour but de faciliter la mise au point des programmes. Enfin, la création d'un panneau de contrôle permet de maitriser finement l'exécution (lancer, arrêter, changer de vitesse). La mise en valeur de la ligne en cours d'exécution vise à aider les élèves à faire le lien entre leur programme et les actions dans la scène du jeu.

Les traces récoltées lors d'une première expérimentation dans les classes nous ont permis de d'évaluer l'utilisation des fonctionnalités que nous venons de décrire. Certains choix de conception ont des conséquences positives. Ainsi, le mémo Python est très fréquemment consulté par les étudiants, il est le support de la découverte et de la remémoration des concepts. Les traductions Scratch-Python incluses dans le mémo sont considérées comme « utiles » par une grande majorité d'élèves, elles doivent accompagner la transition d'un registre sémiotique à l'autre. Le copier-coller à partir du mémo de programmation est largement pratiqué, ce qui a pour effet de limiter la saisie au clavier. Enfin, les rétroactions fournies par l'analyseur syntaxique via des messages d'erreur jugés comme « clairs » permettent aux élèves de corriger leurs programmes en autonomie avec peu d'interventions des enseignants.

D'autre fonctionnalités n'ont pas produit les résultats escomptés. Ainsi le panneau de contrôle devait permettre aux élèves de mieux appréhender l'exécution des programmes. Dans les faits, le bouton de lancement des programmes est fréquemment utilisé et le curseur de contrôle de la vitesse est réglé très tôt au maximum afin de faciliter un mode de programmation incrémental et par essais-erreurs hérité de l'utilisation de Scratch. Le bouton permettant d'arrêter les exécutions est peu utilisé. Lorsqu'il l'est, c'est surtout pour essayer de réussir certains niveaux basés sur le hasard en utilisant des procédures de contournement didactique.

Pour finir, nous avons interrogé les élèves sur leur perception globale de l'application. Il en ressort que la prise en main de l'application est considérée comme bonne par les élèves qui se déclarent motivés par une application jugée ludique, même si le jeu est perçu comme difficile.

Q6 : Comment améliorer l'autonomie des élèves vis-à-vis des enseignants par le biais de rétroactions automatisées lors de l'utilisation de notre application ?

Un premier constat sur l'avancée et l'autonomie des élèves lors de la première expérimentation a été réalisé. Il indique que la moitié des élèves arrivent à atteindre le niveau 5, puis le taux d'achèvement décline, et seuls 6% des élèves terminent entièrement le jeu. Le constat a été également fait que les niveaux 3, 4 et 5 prennent trop de temps aux élèves. De plus, leur autonomie est très insuffisante dans un contexte de classe entière avec un seul enseignant, en particulier pour les niveaux 1, 3 et 4. Plusieurs pistes permettant d'améliorer cette situation ont été évoquées. Il s'agit par exemple de scinder le niveau 3 en plusieurs sous-niveaux afin de diminuer sa densité conceptuelle. Il est également possible de développer un système d'aides automatisées pour tous les niveaux du jeu afin de soulager les enseignants dans leur tâche d'accompagnement.

La seconde piste d'amélioration a été mise en œuvre en créant d'abord un ensemble de rétroactions épistémiques pouvant être déclenchées à la demande en cliquant sur le nouveau bouton « Aide » de l'interface. Ces rétroactions sont délivrées par un tuteur prenant la forme d'un perroquet dans des bulles de dialogue ou via des animations dans la map du jeu. La politique de déclenchement implémentée empêche la diffusion répétée de la même rétroaction. Les rétroactions sont de quatre sortes. Une première de type « jeu », qui explique le parcours à effectuer dans la map du niveau tout en indiquant les fonctions de contrôle à utiliser. Une deuxième de type « concept » qui dévoile le concept principal visé dans le niveau en pointant vers un contenu du mémo. Une troisième de type « implémentation » qui fournit un fragment de code facilitant l'implémentation du concept principal visé dans le niveau. Et enfin, une quatrième de type « solution » qui donne le code intégral permettant d'achever le niveau en cours. Ces rétroactions sont générées en s'appuyant sur les données des élèves (état du système, réponses ouvertes, et comportement dans le système) et sur les décisions d'experts prises in situ dans les classes.

Afin de choisir la rétroaction à déclencher lorsqu'un élève demande de l'aide, la politique de déclenchement fait appel aux prédictions de modèles d'intelligence artificielle qui s'appuient sur l'historique d'activité de l'élève dans le niveau courant. Ces modèles ont été entrainés, à partir de données étiquetées par des experts lors d'une deuxième expérimentation dans les classes, dans le but d'effectuer une tâche de classification. Bien que les rétroactions respectent une certaine relation d'ordre du fait de leur conception, l'enjeu des prédictions consistait à choisir la première rétroaction parmi les quatre possibles, puis à sélectionner les rétroactions suivantes de façon non nécessairement incrémentale.

Pour cela, un jeu de données d'apprentissage a été créé par niveau. Il comporte des caractéristiques non cumulatives transcrivant l'historique de l'activité de l'élève depuis le début du niveau courant selon trois catégories (jeu, concept et exécution). Les étiquettes sont les rétroactions qui ont été déclenchées par les experts sur le terrain.

Seuls les niveaux 1, 3, 4 et 5 comportaient assez d'observations pour l'apprentissage des modèles, les autres ayant été mis de côté. Après avoir entrainé cinq modèles à l'aide d'algorithmes de classification différents, en plus d'une base aléatoire, l'algorithme Random Forest a été retenu. Il propose des performances deux à trois fois supérieures à la référence aléatoire pour les quatre niveaux. Les modèles finaux optimisés ont ensuite été évalués sur le jeu de test. Il en ressort que les rétroactions de niveau 2 (concept), 3 (implémentation) et 4 (solution) sont les plus faciles à prédire à la différence de celles de niveau 1 (jeu) qui connaissent beaucoup d'erreurs (non néfastes à l'apprentissage). Enfin, les erreurs de classification sont peu sévères mais peuvent poser problème pour un cinquième des prédictions (réduction inutile de l'adidacticité) ce qui reste deux fois mieux que la référence aléatoire.

Pour finir, l'importance relative des caractéristiques (MDI) a été évaluée afin d'améliorer l'explicabilité des modèles. Le constat a pu être fait que toutes les catégories de caractéristiques sont utiles aux prédictions. Celles en lien avec les concepts permettent d'évaluer la consultation, la copie et l'utilisation des concepts ciblés. Celles en rapport avec le jeu donnent des informations sur l'avancement dans le niveau, les erreurs de jeu, et l'utilisation des fonctions de contrôle. Enfin, celles ayant trait à l'exécution renseignent sur les erreurs sémantiques et les programmes corrects exécutés.

7.2 Contributions

Au-delà de ces réponses aux questions initiales, cette thèse a des implications dans plusieurs champs de recherche : celui de la didactique de l'informatique, et celui des Environnements Informatiques pour l'Apprentissage Humain (EIAH). Les résultats que nous avons établis et l'application développée peuvent également être utiles aux praticiens que sont les enseignants du secondaire, en poste ou en formation, ainsi qu'aux élèves dans les classes.

Contributions à la recherche en didactique de l'informatique

Nous avons constitué dans le Chapitre 1 un modèle épistémologique du « savoir savant » lié à la programmation informatique. Cet élément fondamental des analyses didactiques (Artigue, 1989) pourra être réutilisé dans d'autres travaux en didactique de l'informatique, en servant par exemple de base à des analyses de curriculums ou de pratiques enseignantes. De plus, un ensemble de praxéologies (Chevallard, 1999) à enseigner dans les quatre institutions que nous avons étudiées (mathématiques et technologie en troisième, et mathématiques et SNT en seconde) sont issues des analyses curriculaires fines que nous avons menées dans le Chapitre 2. Ces ensembles peuvent servir de base à la constitution de *modèles praxéologiques de référence* (Chaachoua et al., 2017) pour les concepts fondamentaux de la programmation informatique dans ces institutions. Ces modèles permettent l'analyse critique des complexes praxéologiques mis en œuvre dans une institution donnée (Doukhan, 2021). Ils pourront ainsi servir de référentiel de comparaison afin d'analyser d'autres choix de transposition dans d'autres institutions.

Ensuite, l'étude fine des différences entre les langages Scratch et Python dont rend compte le Chapitre 3 est en partie généralisable au passage de la programmation par blocs à la programmation textuelle. Ces éléments ont vocation à contribuer au

champ de recherche international sur l'introduction à la programmation (CS1). En effet, en plus de synthétiser les éléments issus de la littérature, nous avons produit de nouveaux résultats concernant l'analyse des erreurs, les concepts fondamentaux implémentables, et la forme des instructions (registres sémiotiques) dans chaque modalité.

D'un point de vue méthodologique, les praxéologies ont déjà été utilisées pour modéliser des activités de programmation informatique. Cependant, dans les travaux existants, ces activités sont toujours prescrites dans un contexte mathématique. Autrement dit, des activités qui consistent à résoudre des problèmes en mobilisant des connaissances mathématiques et des connaissances en programmation. Afin de modéliser ces activités « mixtes » (Chevallard, 2001), Couderette (2016) distingue les praxéologies d'ordre mathématique (par exemple « calculer une moyenne »), d'ordre algorithmique (par exemple « répéter une instruction »), et d'ordre informatique (par exemple « tester un programme »). Elle les articule ensuite les unes aux autres par imbrication et juxtaposition afin de modéliser les activités. Dans notre cas, puisque les activités de programmation que nous avons modélisées sont prescrites dans des contextes très variés (jeux, objets programmables, mathématiques, SNT, etc.), nous avons axé nos analyses sur le dénominateur commun que constituent les concepts de programmation. Nous avons ainsi réalisé un découpage en ingrédients de techniques (Chaachoua, 2018) allant jusqu'à la granularité minimale de l'atome que nous avons défini comme un type de tâches atomiques directement liées aux concepts de la programmation (par exemple « modifier la valeur d'une variable », « utiliser une structure conditionnelle à deux branches »). Plus récemment, Chaachoua et ses collègues (2022) proposent, dans le même ordre d'idées, de découper les techniques en ingrédients de techniques de plusieurs types dans des contextes mixtes de programmation au service d'autres domaines. Ils distinguent les ingrédients relevant de « l'algorithmique et de la programmation » (2022, p. 367) (par exemple « utiliser une boucle répéter »), et ceux concernant le « domaine dont relève le type de tâche » (par exemple « déterminer le quotient de deux nombres » pour le domaine mathématique). Il semble clair, compte tenu du développement de la programmation informatique dans de nombreuses disciplines scolaires au lycée (mathématiques, SNT, sciences physiques, SVT, NSI, etc.), que la modélisation praxéologique de ces tâches mixtes est une question ouverte pour la recherche en didactique de l'informatique.

Notre méthodologie d'ingénierie didactique (Artigue, 1988) comprend également quelques singularités. Ayant connu un âge d'or dans les années 80 et 90 pour les recherches en didactique des mathématiques, elle se trouva questionnée par la suite, notamment en raison d'une adaptation difficile au lycée et à l'université, eu égard à des savoirs en jeu plus complexes et à une souplesse moindre du système éducatif (Artigue, 2002). Néanmoins, nous sommes parvenu à adapter cette méthodologie à notre corpus conceptuel lié à la programmation informatique en créant des « situations fondamentales » (Brousseau, 1998) de ces savoirs (au sens où il y est optimal). Nous avons également mis à profit les formes scolaires plus flexibles permises par l'utilisation d'applications en ligne afin de contrôler les interactions avec les différents éléments du milieu didactique, et en particulier les interventions des enseignants. Enfin, nous proposons une mise en œuvre particulière de l'analyse à priori à travers la détection automatique de stratégies via l'analyse de l'arbre syntaxique des programmes. Cela rend possible le traitement d'un très grand nombre de procédures

(plus de 250 000 dans cette thèse, voir section 5.5.2), ce qui a pour effet d'augmenter la robustesse des résultats.

Contributions à la recherche en EIAH

Nos résultats concourent également à la recherche sur la conception et l'analyse des environnements numériques pour l'éducation et la formation.

D'abord, les analyses épistémologiques et curriculaires que nous avons menées dans les premiers chapitres de la thèse forment un référentiel en capacité de nourrir des ontologies du domaine informatique au service des EIAH.

Ensuite, l'application Pyrates que nous avons conçue puis évaluée prend en charge plusieurs dimensions de la transition Scratch-Python. Relativement aux autres solutions analysées dans l'état de l'art, elle offre en plus, dans le même environnement, des contenus pédagogiques et des situations d'apprentissage permettant de mobiliser une partie des concepts fondamentaux de la programmation en Python. De surcroit, ces concepts sont mis en jeu à travers le modèle des situations adidactiques dans la lignée de l'approche constructiviste piagétienne qui conduit à des niveaux plus élevés de compréhension et de capacité d'analyse (Richardson, 2005).

Ajoutons que cette application fonctionne correctement en production et est capable de collecter les traces d'utilisations des élèves de manière anonyme et dans le standard xAPI. Cela offre la possibilité à d'autres chercheurs du domaine EIAH de s'en emparer comme support d'expérimentation. La dépôt du code source auprès de l'Agence pour la Protection des Programmes (APP), et sa diffusion prochaine sous licence libre devrait permettre d'encore faciliter l'appropriation de Pyrates, en donnant la possibilité de modifier son fonctionnement ainsi que les traces générées.

Ajoutons que la démarche méthodologique complémentaire que nous avons adoptée, alliant analyses didactiques approfondies et développement d'un outil opérationnel, pourrait servir d'exemple à d'autres projets associant de manière féconde didacticiens et informaticiens.

Enfin, l'évolution qui vise à favoriser l'autonomie des élèves, que nous avons présentée dans le Chapitre 6, est une démarche originale en comparaison des travaux connexes présentés dans la revue de littérature. En effet, les classificateurs que nous avons entrainés s'appuient sur une combinaison de caractéristiques basées à la fois sur les programmes soumis mais également sur les interactions des élèves avec l'environnement de l'application. L'analyse des caractéristiques les plus prédictives indique d'ailleurs que ces deux sources d'informations sont utiles pour inférer les types de rétroactions à fournir aux élèves. Notons que les traces d'utilisations sont particulièrement pertinentes dans les environnements hautement interactifs et exploratoires que sont les jeux sérieux. Ajoutons que cela rend possible la fourniture de rétroactions même lorsque les élèves ne produisent pas de code mais explorent simplement l'interface et les contenus pédagogiques.

La singularité de la méthodologie tient également au fait que les étiquettes des données ont été apposées par des experts en situation écologique dans les salles de classe. Le contenu de chaque type de rétroaction a été prédéfini en amont par niveau du jeu en fonction d'analyses de données antérieures. Cela a permis de concevoir un ensemble de rétroactions variées et progressives.

Étant donné que Pyrates est basé sur un principe de jeu couramment utilisé qui consiste à contrôler un personnage dans un environnement, il est possible de généraliser notre approche en la portant dans d'autres jeux sérieux similaires.

Contributions pour les praticiens et les élèves

Pour finir, évoquons l'impact de nos travaux auprès des enseignants et des élèves. Les enseignants en formation, en particulier les étudiants en master MEEF NSI, pourrait s'approprier une version simplifiée de notre méthodologie de modélisation praxéologique afin d'analyser des ressources d'enseignements de la programmation. De la même manière, une adaptation de notre méthodologie d'analyse à priori peut également être employée en formation afin de guider la production de nouvelles ressources d'enseignement.

Ensuite, l'enquête que nous avons menée auprès des enseignants de troisième et de seconde au sujet de leurs pratiques d'enseignement de la programmation informatique est, à notre connaissance, la seule portant sur ce thème. Les résultats qui en ressortent peuvent contribuer à mieux cerner la constitution et les pratiques de ce groupe professionnel émergeant. Ces résultats pourraient éclairer le pilotage des politiques publiques en matière d'éducation en offrant un premier retour d'informations concernant l'introduction de la programmation en fin de collège et au début de lycée. Les résultats spécifiques aux activités proposées et aux artefacts employés sont également en mesure de nourrir les pratiques des enseignants sur le terrain ou en formation.

Enfin, au cours de ses 18 premiers mois en ligne, l'application Pyrates a déjà enregistré plus de 80 000 parties jouées en France, et environ 1 500 à l'international. Cette adoption relativement importante par les enseignants et les élèves contribue directement à l'enseignement-apprentissage de la programmation informatique à la transition collège-lycée, ce qui était un des objectifs de la thèse.

7.3 Limitations

Les résultats que nous venons de présenter doivent être considérés au regard des limites de la méthodologie employée.

Une première série de limitations porte sur les analyses préalables qui marquent la première phase de l'ingénierie didactique. D'abord, notre analyse de transposition didactique s'arrête à la constitution d'un modèle épistémologique lié à la programmation informatique qui sert de base à notre étude de la transposition didactique dans les programmes scolaires. Ce modèle est uniquement fondé sur le savoir universitaire, or la nature intrinsèquement technique et appliquée de la science informatique aurait pu nous amener à prendre également en compte les pratiques sociales de référence (Martinand, 1989). Ces pratiques à l'œuvre dans le monde académique et industriel reflètent la diversité des activités sociales autour de l'informatique. Elles devraient être étudiées afin de mesurer leur transposition dans la sphère scolaire.

Ensuite, notre étude de l'enseignement usuel, s'appuie essentiellement sur les textes officiels et les déclarations des enseignants. Elle aurait pu être approfondie, notamment en direction des élèves. Ainsi, nous aurions pu aller dans les classes

observer les pratiques des enseignants et analyser les ressources d'enseignement qu'ils utilisent (manuels scolaires, ressources numériques, etc.). De même, les conceptions des élèves n'ont pas pu être évaluées. Il aurait été souhaitable d'aller à leur contact afin de jauger leurs connaissances préalables en Scratch, ainsi que les difficultés et les obstacles qu'ils peuvent rencontrer dans l'apprentissage usuel du langage Python. Malheureusement, comme nous l'avons évoqué précédemment, l'accès au terrain a été fortement limité lors de la première année de cette thèse en raison des restrictions sanitaires liées à la pandémie de COVID-19.

La phase d'évaluation de l'application Pyrates atteint également certaines limites. Ainsi, les élèves étant dans un contexte écologique (cours conventionnels de maths ou de SNT), il était difficile de maintenir des conditions expérimentales totalement similaires entre les différents demi-groupes. Cela concerne en particulier l'activité des enseignants et la distance temporelle entre les différentes séances. De plus, raisonner sur des moyennes lors des analyses nous permet certes de dégager des tendances, mais masque les disparités de niveaux et de pratiques entre les élèves que nous avons pu observer dans les classes. Ensuite, la mise à jour manuelle de la liste des stratégies de l'analyse à priori à partir des procédures mises en œuvre par les élèves lors de la première expérimentation était assez fastidieuse. Elle n'était par ailleurs pas applicable aux très nombreux programmes soumis par les utilisateurs finaux. Une piste pourrait être d'utiliser des algorithmes de fouilles de données de type clustering afin de regrouper automatiquement les programmes similaires et potentiellement détecter de nouvelles stratégies. Enfin, nous n'avons pas eu l'occasion d'évaluer les apprentissages effectifs des élèves en Python suite aux deux séances d'utilisation de Pyrates.

Pour finir, la conception du système de rétroactions automatiques connait également des limitations. Premièrement, comme nous l'avons vu, la conception manuelle des rétroactions pour chaque niveau et le besoin de données étiquetées par des experts sur le terrain engendrent une faible flexibilité de l'application. La modification ou l'ajout d'un nouveau niveau implique par exemple de mobiliser à nouveau des experts et des élèves. Deuxièmement, dans le contexte de rétroactions ordonnées qui nous occupe, la métrique que nous avons employée pour évaluer nos modèles ne tient pas compte de l'écart entre la rétroaction prédite et la rétroaction effective. Une métrique sensible à cet ordre pourrait permettre de créer des modèles plus adaptés à notre tâche de sélection de rétroaction. Troisièmement, nous disposons d'un volume de données limité qui ne nous permet pas d'entrainer des modèles pour tous les niveaux du jeu. De plus, les caractéristiques que nous calculons synthétisent l'activité des élèves à travers des moyennes et des taux depuis le début du niveau. Cela a pour effet d' « écraser » les données en perdant l'aspect dynamique des traces. Nous ne sommes par exemple pas en mesure de prendre en compte certaines informations sur les élèves, comme le style d'édition, la vitesse de progression, etc. Quatrièmement, donner la bonne réponse en dernier recours permet à certains étudiants de passer au niveau suivant sans avoir nécessairement compris les concepts en jeu dans le niveau courant. Étant donné que l'utilisation des concepts est accumulative d'un niveau à l'autre, cela peut conduire à un cercle vicieux menant au déclenchement de la rétroaction-solution pour tous les niveaux, sans apprentissages pour l'élève. Ainsi, Shute (2008) conseille d'éviter que la séquence de rétroactions termine toujours par la bonne réponse et de prévoir des dispositions limitant leur utilisation abusive : « Avoid

using progressive hints that always terminate with the correct answer. Although hints can be facilitative, they can also be abused, so if they are employed to scaffold learners, provisions to prevent their abuse should be made » (2008, p. 179).

7.4 Perspectives

Enfin, nous terminons ce manuscrit en évoquant quelques prolongements dans plusieurs directions qui pourront faire l'objet de futurs travaux selon les opportunités dont nous disposerons.

Transposition didactique de la programmation informatique

Nous avons entamé dans cette thèse une étude de la transposition didactique de la programmation informatique. Comme nous l'avons évoqué dans la section précédente, cette étude pourrait être complétée, notamment en utilisant une approche en termes de pratiques sociales de référence (Martinand, 1989). En effet, la programmation informatique est mise en œuvre dans différents groupes sociaux sur lesquels pèsent un certain nombre de normes et de « bonnes pratiques ». Cela peut par exemple concerner la manière de mettre en forme les programmes au-delà de la syntaxe du langage : langue et forme des identifiants de variables (camel_case, snakeCase, kebab-case, etc.), formalisme et langue employée pour les commentaires accompagnant le code, manière d'indenter les programmes, style et recherche d'une certaine « élégance » dans les solutions développées. La culture disciplinaire (Becher & Trowler, 1989) des toutes récentes matières informatiques dans le secondaire (SNT et NSI) est en cours de constitution, et nul doute que l'identification de ces pratiques de référence serait utile pour en étudier la constitution.

Application Pyrates : analyse des usages et évolutions possibles

Ensuite, il serait possible de s'intéresser à la manière dont les enseignants s'emparent de l'application Pyrates dans les classes en questionnant les dispositifs mis en place et l'articulation de cette première approche avec les autres phases de l'enseignement comme l'institutionnalisation des connaissances. Nous pourrions utiliser pour cela l'approche instrumentale (Rabardel, 1995) en étudiant plus particulièrement les orchestrations instrumentales (Drijvers et al., 2010; Trouche, 2003) mises en œuvre. Il serait également utile d'évaluer les apprentissages des élèves dans le langage Python en mettant en œuvre des pré-tests et post-tests sur deux groupes, l'un utilisant Pyrates et l'autre suivant un enseignement traditionnel sur deux séances.

L'application pourrait également évoluer en proposant un « outil auteur » donnant l'opportunité aux enseignants et aux chercheurs de créer facilement de nouveaux niveaux répondant à leur problématique propre sans avoir à modifier son code source. Il serait également imaginable de laisser le choix de l'univers du jeu via la fourniture ou la sélection d'un tile set et d'un sprite sheet respectant certaines conditions. La conception d'une interface de type « tableau de bord » permettant aux enseignants de suivre l'avancement individuel de tous les élèves d'une classe serait également une piste à suivre.

Rétroactions épistémiques : déploiement, évaluation et améliorations

Enfin, notre mécanisme de rétroactions épistémiques automatisées pourrait également faire l'objet d'améliorations et de prolongements. Il s'agirait premièrement de déployer en production les modèles d'IA que nous avons entrainés et de finaliser leur intégration dans Pyrates. Par la suite, nous pourrions étudier l'effet des différentes rétroactions sur les élèves lorsqu'elles sont déclenchées par l'IA. Cela consisterait par exemple à mener une étude contrôlée visant à évaluer les effets des rétroactions sur l'activité des élèves et leurs apprentissages. Il serait également possible d'interroger les élèves afin de se faire une idée de leur expérience utilisateur.

Une autre piste serait d'améliorer les performances des modèles en utilisant des algorithmes d'apprentissage profond basés sur les réseaux neuronaux (en gardant en tête que cela nécessite beaucoup de données). En complément, il serait sans doute profitable de créer de nouvelles caractéristiques capables d'aider les classificateurs dans leurs prédictions. Il s'agirait d'exploiter les données des élèves que nous n'avons pas ou peu exploitées lors de cette thèse : le comportement dans la vie réelle (par exemple accessible via l'expression faciale ou le suivi du regard), et les caractéristiques de l'élève (par exemple les connaissances acquises dans les niveaux précédents) (Deeva et al., 2021).

Concernant les rétroactions elles-mêmes, il serait possible d'augmenter leur granularité en fournissant une ou plusieurs rétroactions intermédiaires entre celles donnant l'implémentation du concept visé et celles fournissant la solution complète du niveau. Nous pourrions ainsi donner des indications au sujet des concepts impliqués dans les niveaux précédents, ou sur l'articulation des différents concepts entre eux. Une autre piste consisterait à encore diversifier le contenu des rétroactions sur le thème de la performance ou de la métacognition (Narciss, 2013).

Enfin, dans la solution que nous avons développée, les élèves doivent cliquer sur un bouton lorsqu'ils souhaitent obtenir de l'aide. Le fait de pouvoir prédire si les élèves ont besoin d'aide nous permettrait d'offrir une modalité de déclenchement de rétroaction du type « proposé » (Deeva et al., 2021). Shute (2008) va dans le sens de cette approche en affirmant que les élèves engagés dans une tâche d'apprentissage difficile (ce qui d'après nos résultats est le ressenti global des élèves qui utilisent Pyrates) ont besoin de rétroactions immédiates afin qu'ils ne s'enlisent pas et ne se sentent pas découragés : « When a student is learning a difficult new task [...], it is better to use immediate feedback, at least initially. This provides a helpful safety net for the learner so she does not get bogged down and frustrated » (2008, p. 179). Il s'agirait ainsi d'interroger à intervalles réguliers un classificateur binaire prédisant le besoin d'aide. Cela nous permettrait, en cas de prédiction positive, d'activer le bouton d'aide afin de notifier l'élève qu'une aide est disponible, tout en limitant la distraction générée par une rétroaction immédiate. Pour ce faire, nous pourrions utiliser les données dont nous disposons en simulant des étiquettes "pas besoin d'aide" à des moments bien choisis selon la technique du point pivot (Lallé et al., 2016; Tan & Berger, 1999). Nous pourrions aussi, à la manière de Moresi et ses collègues (2021), demander à des experts d'étiqueter les données (besoin d'aide ou non) lorsqu'un élève est bloqué ou rencontre une difficulté dans la tâche qui l'occupe.

Le champ de recherche des EIAH est fortement pluridisciplinaire. Dans cette thèse, nous avons combiné au sein de ce champ une contribution relevant de la didactique de l'informatique, et une contribution plus spécifiquement informatique à travers la conception et l'évaluation d'un outil. Chacun de ces deux aspects correspond à un domaine en plein essor au niveau international. La didactique de l'informatique se développe notamment en lien avec les évolutions curriculaires que nous avons évoquées. Les technologies éducatives progressent fortement, notamment à travers le développement de fonctionnalités qui s'appuient sur les possibilités de l'IA. Notre contribution porte sur l'enseignement-apprentissage de la programmation informatique, nous espérons pouvoir l'élargir par la suite car cela constitue un enjeu social majeur pour l'avenir dont la recherche doit se saisir.

Bibliographie

Abiteboul, S., & Cattan, J. (2022). *Nous sommes les réseaux sociaux*. Odile Jacob.

Abiteboul, S., & Dowek, G. (2017). *Le temps des algorithmes*. Le pommier.

Akre-Aas, C., Kindem, I., & Divitini, M. (2022). Fighting the Gender Gap in ICT. In Ó. Mealha, M. Dascalu, & T. Di Mascio (Éds.), *Ludic, Co-design and Tools Supporting Smart Learning Ecosystems and Smart Education* (p. 27-39). Springer Singapore.

Albarello, L., Bourgeois, É., & Guyot, J.-L. (2010). *Statistique descriptive. Un outil pour les praticiens chercheurs*. De Boeck Supérieur.

Altadmri, A., & Brown, N. C. C. (2015). 37 Million Compilations : Investigating Novice Programming Mistakes in Large-Scale Student Data. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 522-527. https://doi.org/10.1145/2676723.2677258

Alvarez, J. (2007). *Du jeu vidéo au serious game. Approches culturelle, pragmatiques et formelle*. [Thèse de doctorat, Université Toulouse III - Paul Sabatier]. https://hal.science/tel-01240683/

Alves, M., Neves, S., & Mendonça, T. M. (2019). La notion de fonction : Les praxéologies prescrites pour être enseignées lors de l'articulation entre le collège et le lycée à São Paulo. *Educação Matemática Pesquisa*, *21*(4), 298-310.

Amadieu, F., & Tricot, A. (2014). *Apprendre avec le numérique : Mythes et réalités*. Retz.

Andrews, E., Bau, D., & Blanchard, J. (2021). From Droplet to Lilypad : Present and Future of Dual-Modality Environments. *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 1-2. https://doi.org/10.1109/vl/hcc51201.2021.9576355

Artaud, M. (1997). Introduction à l'approche écologique du didactique. L'écologie des organisations mathématiques et didactiques. In M. Bailleul (Éd.), *Actes de la IXe École d'été de didactique des mathématiques* (p. 101-139). IUFM de l'académie de Caen.

Artigue, M. (1988). Ingénierie didactique. *Recherches en didactique des mathématiques*, *9*(3), 281-308.

Artigue, M. (1989). *Épistémologie et didactique*. (I. de Paris, Éd.). IREM de Paris. https://hal.archives-ouvertes.fr/hal-02138030

Artigue, M. (2002). Ingénierie didactique : Quel rôle dans la recherche didactique aujourd'hui? *Les dossiers des sciences de l'éducation*, *8*(1), 59-72.

Bachelard, G. (1938). *La formation de l'esprit scientifique. Contribution à une psychanalyse de la connaissance objective*. (Vrin).

Barnes, T., & Stamper, J. (2010). Automatic Hint Generation for Logic Proof Tutoring Using Historical Data. *Journal of Educational Technology & Society*, *13*(1), 3-12.

Baron, G.-L., & Bruillard, É. (2011). L'informatique et son enseignement dans l'enseignement scolaire général français : Enjeux de pouvoir et de savoirs. In

Recherches et expertises pour l'enseignement scientifique: Vol. 1re éd. (p. 79-90). De Boeck Supérieur; Cairn.info. https://doi.org/10.3917/dbu.lebea.2011.01.0079

Baron, G.-L., & Drot-Delange, B. (2016). L'informatique comme objet d'enseignement à l'école primaire française ? Mise en perspective historique. *Revue française de pédagogie*, *2*, 51-62.

Baron, G.-L., Drot-Delange, B., Grandbastien, M., & Tort, F. (2014). Computer Science Education in French Secondary Schools : Historical and Didactical Perspectives. *ACM Trans. Comput. Educ.*, *14*(2). https://doi.org/10.1145/2602486

Baron, G.-L., Drot-Delange, B., Grandbastien, M., & Tort, F. (2015). L'enseignement de l'informatique dans l'enseignement secondaire en France : Un retour de balancier ? In *Informatique en éducation : Perspectives curriculaires et didactiques* (p. 83-101). Presses Universitaires Blaise-Pascal. https://hal.science/hal-01136340

Bart, A., Tibau, J., Tilevich, E., Shaffer, C. A., & Kafura, D. (2017). BlockPy : An Open Access Data-Science Environment for Introductory Programmers. *Computer*, *50*(05), 18-26. https://doi.org/10.1109/mc.2017.132

Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code : Block Code for a Text World. *Proceedings of the 14th International Conference on Interaction Design and Children*, 445-448. https://doi.org/10.1145/2771839.2771875

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable Programming : Blocks and Beyond. *Commun. ACM*, *60*(6), 72-80. https://doi.org/10.1145/3015455

Baudé, J. (2014a). L'expérience des 58 lycées. *1024 - Bulletin de la SIF*, *4*, 105-115.

Baudé, J. (2014b). L'option informatique dans les lycées dans les années 80 et 90. *1024 - Bulletin de la SIF*, *2*, 85-97.

Baudé, J. (2017). Le séminaire de Sèvre. *1024 - Bulletin de la SIF*, *11*, 115-127.

Baudé, J. (2019). Le mariage du siècle : Éducation et informatique. *1024 - Bulletin de la SIF*, *13*, 71-78.

Baudé, J. (2021). Naissance d'une discipline Une gestation d'un demi siècle pas encore achevée. *EpiNet - La revue électronique de l'EPI*.

Becher, T., & Trowler, P. (1989). *Academic tribes and territories. Intellectual enquiry and the culture of disciplines*. Open University Press.

Becker, B. A. (2016). An Effective Approach to Enhancing Compiler Error Messages. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 126-131. https://doi.org/10.1145/2839509.2844584

Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B., Kamil, A., Karkare, A., McDonald, C., Osera, P.-M., Pearce, J. L., & Prather, J. (2019). Compiler Error Messages Considered Unhelpful : The Landscape of Text-Based Programming Error Message Research. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, 177-210. https://doi.org/10.1145/3344429.3372508

Ben Nejma, S. (2019). Les difficultés langagières au centre des pratiques algébriques : L'exemple de la transition collège/lycée en Tunisie. In *Formation et*

enseignement des mathématiques et des sciences. Didactique, TIC et innovation pédagogique (p. 102-113). CRMEF.

Bennedsen, J., & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. *SIGCSE Bull.*, *39*(2), 32-36. https://doi.org/10.1145/1272848.1272879

Berndt, A. E. (2020). Sampling Methods. *Journal of Human Lactation*, *36*(2), 224-226. https://doi.org/10.1177/0890334420906850

Berry, G. (2017). *L'hyperpuissance de l'informatique : Algorithmes, données, machines, réseaux*. Odile Jacob.

Berry, G., Dowek, G., Abiteboul, S., Archambault, J.-P., Balagué, C., Baron, G.-L., de la Higuera, C., Nivat, M., Tort, F., & Viéville, T. (2013). L'enseignement de l'informatique en France. Il est urgent de ne plus attendre. *Rapport de l'Académie des sciences*.

Bessot, A. (2003). Une introduction à la théorie des situation didactiques. *Les cahiers du laboratoire Leibniz*, *91*, 1-28.

Branthôme, M. (2021). Apprentissage de la programmation informatique à la transition collège-lycée. *STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation)*, *28*(3), 1-35. https://doi.org/10.23709/sticef.28.3.1

Breton, P. (1990). *Une histoire de l'informatique* (2ème). Edition du Seuil.

Brousseau, G. (1981). Problèmes de didactiques des décimaux. *Recherches en Didactique des Mathématiques*, *2*(1), 37-125.

Brousseau, G. (1982). Les objets de la didactique des mathématiques. In *Actes de la Troisième école d'été de didactique des mathématiques* (p. 5-17). ARDM.

Brousseau, G. (1986a). Fondements et méthodes de la didactique des mathématiques. *Recherches en Didactique des Mathématiques*, *7*(2), 33-115.

Brousseau, G. (1986b). *Théorisation des phénomènes d'enseignement des mathématiques* [Thèse de doctorat d'état]. Université de Bordeaux I.

Brousseau, G. (1998). *Théorie des situations didactiques : Didactique des mathématiques 1970-1990*. La Pensée Sauvage.

Brousseau, G. (2002). Les doubles jeux de l'enseignement des mathématiques. *Revue du Centre de Recherches en Education, Université de Saint Etienne*, *22-23*, 83-155.

Brousseau, G. (2010). *Glossaire de quelques concepts de la théorie des situations didactiques en mathématiques*. Site personnel. http://guy-brousseau.com/wp-content/uploads/2010/09/Glossaire_V5.pdf

Brousseau, G. (2012). Des dispositifs piagétiens… aux situations didactiques. *Éducation et Didactique*, *6*(2), 103-129.

Bruillard, E. (2016). Quelle informatique à repenser et à construire pour les élèves de l'école primaire ? In F. Villemonteix, G.-L. Baron, & J. Béziat (Éds.), *L'école primaire et les technologies informatisées. Des enseignants face aux TICE.* (Presses Universitaires du Septentrion, p. 29-38).

Candy, J. (2020). *Etude de la transposition didactique du concept d'idéal : Écologie des savoirs et problématique de l'entrée dans la pensée structuraliste, en France*

et en Suisse romande [Thèse de doctorat, Université de Montpellier]. https://hal.science/tel-03119093

Cassell, J. (2002). Genderizing human-computer interaction. In *The human-computer interaction handbook : Fundamentals, evolving technologies and emerging applications* (Lawrence Erlbaum Associates Publishers, p. 401-412).

Chaachoua, H. (2018). T4TEL, un cadre de référence didactique pour la conception des EIAH. In J. Pilet & C. Vendeira (Éds.), *Actes du séminaire national de l'ARDM* (p. 8-25). IREM de Paris.

Chaachoua, H., Crisci, R., & Tchounikine, P. (2022). Un modèle praxéologique de référence pour des praxéologies mixtes dans des tâches de programmation. *Proceeding of CITAD7*, 361-372.

Chaachoua, H., Ferraton, G., & Desmoulins, C. (2017). Utilisation du modèle praxéologique de référence dans un EIAH. *Évolutions contemporaines du rapport aux mathématiques et aux autres savoirs à l'école et dans la société*, 301-324.

Chevallard, Y. (1985). *La transposition didactique : Du savoir savant au savoir enseigné*. La Pensée sauvage.

Chevallard, Y. (1992). Concepts fondamentaux de la didactique : Perspectives apportées par une approche anthropologique. *Recherches En Didactique Des Mathématiques*, *12*(1), 73-112.

Chevallard, Y. (1999). L'analyse des pratiques enseignantes en théorie anthropologique du didactique. *Recherches en didactique des mathématiques*, *19*(2), 221-265.

Chevallard, Y. (2001). Les mathématiques et le monde : Dépasser «l'horreur instrumentale». *Quadrature*, *41*, 25-40.

Chevallard, Y. (2002). Organiser l'étude. 3. Ecologie & régulation. In J.-L. Dorier (Éd.), *Actes de la XIème Ecole d'Eté de Didactique des Mathématiques* (p. 41-56). La Pensée Sauvage.

Chevallard, Y. (2003). Approche anthropologique du rapport au savoir et didactique des mathématiques. In S. Maury & M. Caillot (Éds.), *Communication aux 3es Journées d'étude franco-québécoises. Rapport au savoir et didactiques.* (p. 81-104). Fabert.

Chow, S., Yacef, K., Koprinska, I., & Curran, J. (2017). Automated Data-Driven Hints for Computer Programming Students. *Adjunct Publication of the Conference on User Modeling, Adaptation and Personalization*, 5-10.

Church, A. (1936). A note on the Entscheidungsproblem. *The journal of symbolic logic*, *1*(1), 40-41.

CodeMonkey home page. (2023). https://www.codemonkey.com

CodingPark home page. (2023). https://codingpark.io

Cohen, I., Huang, Y., Chen, J., Benesty, J., Benesty, J., Chen, J., Huang, Y., & Cohen, I. (2009). Pearson correlation coefficient. *Noise reduction in speech processing*, 1-4.

Couderette, M. (2016). Enseignement de l'algorithmique en classe de seconde : Une introduction curriculaire problématique. *Annales de Didactique et de Sciences Cognitives. Revue internationale de didactique des mathématiques*, *21*, 267-296.

CREM. (2003). *L'algorithmique au lycée—Annexe au rapport informatique et enseignement des mathématiques*. Commission de reflexion sur l'enseignement de l'éducation. http://www-old.irem.univ-paris-diderot.fr/up/Rapport%20informatique-Annexe-Algorithmique%20au%20lycee.pdf

Crisci, R. (2020). *Etude des conditions de viabilité d'une approche basée sur l'algorithmique et la programmation pour l'apprentissage de la division euclidienne à l'école primaire* (Numéro 2020GRALM046) [Thèse de doctorat, Université Grenoble Alpes]. https://theses.hal.science/tel-03116813

Deeva, G., Bogdanova, D., Serral, E., Snoeck, M., & Weerdt, J. D. (2021). A review of automated feedback systems for learners : Classification framework, challenges and opportunities. *Computers & Education*, *162*, 104094.

Delmas-Rigoutsos, Y. (2020). *Variables, grandeurs et types*. Colloque Didapro-Didastic 8, Lille, France. https://hal.science/hal-02477000

Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011). Understanding the Syntax Barrier for Novices. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 208-212. https://doi.org/10.1145/1999747.1999807

DEPP. (2021a). *Les effets des choix des élèves en lycée général et technologique sur les services des enseignants*. Direction de l'évaluation, de la prospective et de la performance. https://www.education.gouv.fr/media/96049/download

DEPP. (2021b). *Repères et références statistiques 2021*. Direction de l'évaluation, de la prospective et de la performance. https://www.education.gouv.fr/media/92540/download

DEPP. (2022a). *Panorama statistique des personnels de l'enseignement scolaire 2021-2022*. Direction de l'évaluation, de la prospective et de la performance. https://www.education.gouv.fr/media/118135/download

DEPP. (2022b). *Repères et références statistiques 2022*. Direction de l'évaluation, de la prospective et de la performance. https://www.education.gouv.fr/media/116557/download

de Singly, F. (2020). *Le questionnaire: Vol. 5e éd.* Armand Colin. https://www.cairn.info/le-questionnaire--9782200626877.htm

Djaouti, D., Alvarez, J., & Jessel, J.-P. (2011). Classifying serious games : The G/P/S model. In *Handbook of research on improving learning and motivation through educational games : Multidisciplinary approaches* (p. 118-136). IGI global.

Djaouti, D., Alvarez, J., Jessel, J.-P., Methel, G., & Molinier, P. (2008). A Gameplay Definition through Videogame Classification. *International Journal of Computer Games Technology*, *2008*. https://doi.org/10.1155/2008/470350

Djelil, F. (2016). *Conception et évaluation d'un micromonde de Programmation Orientée-Objet fondé sur un jeu de construction et d'animation 3D* [Thèse de doctorat, Université Blaise Pascal - Clermont II]. https://hal.science/tel-01487039

Donnard, E. (2021). *La lutte pour l'enseignement de l'informatique. Une histoire particulière de l'intégration d'une innovation à l'Education Nationale*. [Mémoire de master]. SciencesPo.

Dorier, J.-L. (2010). L'analyse a priori : Un outil pour la formation d'enseignants– exemple d'un jeu issu des manuels suisses romands de première année primaire. In *Actes du XXXVIème colloque international des formateurs de professeurs des écoles en mathématiques* (p. 80-92). COPIRELEM.

Douady, R. (1984). *Jeux de cadres et dialectiques outil-objet dans l'enseignement des Mathématiques. Une réalisation dans tout le cursus primaire.* [Thèse de doctorat d'état]. Université de Paris VII.

Doukhan, C. (2021). *Modèles praxéologiques dans la transition secondaire-supérieur : Le cas des probabilités en filière biologie* (Numéro 2021BRES0094) [Thèse de doctorat, Université de Bretagne occidentale - Brest]. https://theses.hal.science/tel-03632311

Dowek, G. (2008). *Les principes des langages de programmation*. Editions Ecole Polytechnique.

Dowek, G. (2011). Les quatre concepts de l'informatique. In G.-L. Baron, E. Bruillard, & V. Komis (Éds.), *Sciences et technologies de l'information et de la communication en milieu éducatif : Analyse de pratiques et enjeux didactiques. Actes du colloque international DIDAPRO 4—24-26 octobre 2011, Université de Patras* (p. 21-29). New Technologies Editions.

Dowek, G., Archambault, J.-P., Baccelli, E., Boldo, S., Bouhineau, D., Cegielski, P., Clausen, T. H., Guessarian, I., Lopes, S., & Mounier, L. (2011). *Une introduction à la science informatique pour les enseignants de la discipline en lycée* (CRDP Académie de Paris).

Drijvers, P., Doorman, M., Boon, P., Reed, H., & Gravemeijer, K. (2010). The teacher and the tool : Instrumental orchestrations in the technology-rich mathematics classroom. *Educational Studies in Mathematics*, *75*(2), 213-234. https://doi.org/10.1007/s10649-010-9254-5

Drot-Delange, B. (2016). *Internet en éducation : Objets de savoirs* [Synthèse pour l'habilitation à diriger des recherches]. Université Blaise Pascal.

Drot-Delange, B. (2019). Interactions entre gestion personnelle de l'information et connaissances professionnelles des enseignants. Le cas de l'informatique et sciences du numérique (ISN). *Recherches en éducation*, *35*. https://doi.org/10.4000/ree.1726

Drot-Delange, B., & Tort, F. (2018). *Concours Castor, ressource pédagogique pour l'enseignement de l'informatique ? Étude exploratoire auprès d'enseignants*. 199-218.

Drot-Delange, B., & Tort, F. (2022a). Éducation aux données ou enseignement des données : Quelles humanités numériques au lycée? *Humanités numériques*, *5*.

Drot-Delange, B., & Tort, F. (2022b). *Les datasprints, un dispositif d'éducation aux données ? Une étude exploratoire via le cas de «Traces de Soldats»*. Didapro 22 - L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche?

Duval, R. (1993). Registres de représentation sémiotique et fonctionnement cognitif de la pensée. *Annales de didactique et de sciences cognitives*, *5*, 37-65.

Ebbinghaus, H. (1913). *Memory : A contribution to experimental psychology*. Teachers College, Columbia University.

Edublocks home page. (2023). https://app.edublocks.org/

Edwards, S. H. (2004). Using Software Testing to Move Students from Trial-and-Error to Reflection-in-Action. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 26-30. https://doi.org/10.1145/971300.971312

Elimelech, O. C., Ferrante, S., Josman, N., Meyer, S., Lunardini, F., Gómez-Raja, J., Galán, C., Cáceres, P., Sciama, P., Gros, M., Vurro, C., & Rosenblum, S. (2022). Technology use characteristics among older adults during the COVID-19 pandemic : A cross-cultural survey. *Technology in Society*, *71*, 102080. https://doi.org/10.1016/j.techsoc.2022.102080

Artificial Intelligence Act, (2021) (testimony of European Commission). https://eur-lex.europa.eu/legal-content/FR/TXT/?uri=CELEX%3A52021PC0206

European Commission. (2022). *Eurydice report : Informatics education at school in Europe*. Publications Office of the European Union.

Floridi, L., Cowls, J., Beltrametti, M., Chatila, R., Chazerand, P., Dignum, V., Luetge, C., Madelin, R., Pagallo, U., Rossi, F., Schafer, B., Valcke, P., & Vayena, E. (2018). AI4People—An Ethical Framework for a Good AI Society : Opportunities, Risks, Principles, and Recommendations. *Minds and Machines*, *28*(4), 689-707. https://doi.org/10.1007/s11023-018-9482-5

Floyd, R. W. (1978). The paradigms of programming. *Communications of the ACM*, *22*(8), 455-460.

Fluckiger, C. (2019). *Une approche didactique de l'informatique scolaire*. Presses universitaires de Rennes.

Forum NSI home page. (2023). https://mooc-forums.inria.fr/moocnsi/

Fossati, D., Di Eugenio, B., Ohlsson, S., Brown, C., & Chen, L. (2015). Data driven automatic feedback generation in the iList intelligent tutoring system. *Technology, Instruction, Cognition and Learning*, *10*(1), 5-26.

Froidevaux, C. (2023). Enseignement de l'informatique : 10 ans d'actions. *1024 - Bulletin de la SIF*, *21*, 47-55.

Fudenberg, D., & Tirole, J. (1991). *Game theory*. MIT press.

Fushiki, T. (2011). Estimation of prediction error by using K-fold cross-validation. *Statistics and Computing*, *21*, 137-146.

Gabbrielli, M., & Martini, S. (2010). *Programming Languages : Principles and Paradigms*. Springer. https://doi.org/10.1007/978-1-84882-914-5

Galesic, M. (2006). Dropouts on the web : Effects of interest and burden experienced during an online survey. *Journal of official statistics*, *22*(2), 313-328.

Garneli, V., Giannakos, M. N., & Chorianopoulos, K. (2015). Computing education in K-12 schools : A review of the literature. *2015 IEEE Global Engineering Education Conference (EDUCON)*, 543-551. https://doi.org/10.1109/EDUCON.2015.7096023

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.

Ghezzi, C., & Jazayeri, M. (2008). *Programming language concepts*. John Wiley & Sons.

Ginat, D., Shifroni, E., & Menashe, E. (2011). Transfer, Cognitive Load, and Program Design Difficulties. In I. Kalaš & R. T. Mittermeir (Éds.), *Informatics in Schools. Contributing to 21st Century Education* (p. 165-176). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-24722-4_15

Gingras, M., & Belleau, H. (2015). *Avantages et désavantages du sondage en ligne comme méthode de collecte de données : Une revue de la littérature* (INRS Centre-Urbanisation Culture Société). https://espace.inrs.ca/id/eprint/2678

Gopalan, V., Bakar, J. A. A., Zulkifli, A. N., Alwi, A., & Mat, R. C. (2017). A review of the motivation theories in learning. *AIP Conference Proceedings*, *1891*(1), 020043. https://doi.org/10.1063/1.5005376

Gueudet, G., & Trouche, L. (2008). Du travail documentaire des enseignants : Genèses, collectifs, communautés. Le cas des mathématiques. *Education et didactique*, *2*(3), 7-33.

Gueudet, G., & Trouche, L. (2010). Des ressources aux documents, travail du professeur et genèses documentaires. In G. Gueudet & L. Trouche (Éds.), *Ressources vives. Le travail documentaire des professeurs en mathématiques.* (p. 57-74). Presses Universitaires de Rennes et INRP. https://hal.science/hal-00497305

Gueudet, G., & Vandebrouck, F. (2022). Transition secondaire-supérieur : Ce que nous apprend la recherche en didactique des mathématiques. *epiDEMES*, *1 | 2022*. https://doi.org/10.46298/epidemes-7486

Guide pédagogique Pyrates. (2023). https://py-rates.fr/guide/FR/

Guo, P. J. (2018). Non-Native English Speakers Learning Computer Programming : Barriers, Desires, and Design Opportunities. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1-14. https://doi.org/10.1145/3173574.3173970

Hattie, J., & Timperley, H. (2007). The Power of Feedback. *Review of Educational Research*, *77*(1), 81-112. https://doi.org/10.3102/003465430298487

Head, A., Glassman, E., Soares, G., Suzuki, R., Figueredo, L., D'Antoni, L., & Hartmann, B. (2017). Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. *Conference on Learning@Scale*, 89-98.

Hicks, A., Peddycord, B., & Barnes, T. (2014). Building games to learn from their players : Generating hints in a serious game. *International Conference on Intelligent Tutoring Systems*, 312-317.

Hubwieser, P., Armoni, M., Giannakos, M. N., & Mittermeir, R. T. (2014). Perspectives and Visions of Computer Science Education in Primary and Secondary (K-12) Schools. *ACM Trans. Comput. Educ.*, *14*(2). https://doi.org/10.1145/2602482

Hunter, J. D. (2007). Matplotlib : A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(03), 90-95. https://doi.org/10.1109/MCSE.2007.55

Jobin, A., Ienca, M., & Vayena, E. (2019). The global landscape of AI ethics guidelines. *Nature Machine Intelligence*, *1*(9), 389-399.

Jolivet, S. (2018). *Modèle de description didactique de ressources d'apprentissage en mathématiques, pour l'indexation et des services EIAH* (Numéro 2018GREAM074) [Thèse de doctorat, Université Grenoble Alpes]. https://theses.hal.science/tel-02079412

Jouët, J. (2000). Retour critique sur la sociologie des usages. *Réseaux. Communication-Technologie-Société*, *18*(100), 487-521.

Journault, M., Lafourcade, P., Poulain, R., & More, M. (2020). *Une preuve pour le lycée de l'indécidabilité du probleme de l'arrêt*. Didapro 20 - L'informatique, objets d'enseignements enjeux épistémologiques, didactiques et de formation.

Kazemitabaar, M., Chyhir, V., Weintrop, D., & Grossman, T. (2022). CodeStruct : Design and Evaluation of an Intermediary Programming Environment for Novices to Transition from Scratch to Python. *Interaction Design and Children*, 261-273. https://doi.org/10.1145/3501712.3529733

Kevan, J. M., & Ryan, P. R. (2016). Experience API: Flexible, Decentralized and Activity-Centric Data Collection. *Technology, Knowledge and Learning*, *21*(1), 143-149. https://doi.org/10.1007/s10758-015-9260-x

Khazaei, B., & Jackson, M. (2002). Is there any difference in novice comprehension of a small program written in the event-driven and object-oriented styles? *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, 19-26. https://doi.org/10.1109/HCC.2002.1046336

Klieve, H., Beamish, W., Bryer, F., Rebollo, R., Perrett, H., & Van Den Muyzenberg, J. (2010). Accessing practitioner expertise through online survey tool LimeSurvey. *Knowledge in Technology Education (TERC 2010)*, *2*.

Knuth, D. E. (1968). *The Art of Computer Programming, vol 1 : Fundamental Algorithms*. Addison-Wesley.

Kohn, T. (2017). *Teaching Python Programming to Novices : Addressing Misconceptions and Creating a Development Environment* [PhD Thesis, ETH Zurich]. https://doi.org/10.3929/ethz-a-010871088

Kölling, M., Brown, N. C. C., & Altadmri, A. (2015). Frame-Based Editing : Easing the Transition from Blocks to Text-Based Programming. *Proceedings of the Workshop in Primary and Secondary Computing Education*, 29-38. https://doi.org/10.1145/2818314.2818331

Krishnamurthi, S., & Fisler, K. (2019). Programming paradigms and beyond. *The Cambridge Handbook of Computing Education Research*, *37*.

Kroustalli, C., & Xinogalos, S. (2021). Studying the effects of teaching programming to lower secondary school students with a serious game : A case study with Python and CodeCombat. *Education and Information Technologies*, *26*(5), 6069-6095. https://doi.org/10.1007/s10639-021-10596-y

Kuhn, T. S. (1970). *The structure of scientific revolutions*. University of Chicago press.

Kumar, E. (2011). *Natural language processing*. IK International Publishing House Pvt Ltd.

Kyfonidis, C., Weill-Tessier, P., & Brown, N. (2021). Strype : Frame-Based Editing Tool for Programming the Micro:Bit through Python. *The 16th Workshop in Primary and Secondary Computing Education*, 1-2. https://doi.org/10.1145/3481312.3481324

Lakens, D. (2022). Sample size justification. *Collabra: Psychology*, *8*(1). https://doi.org/10.1525/collabra.33267

Lallé, S., Conati, C., & Carenini, G. (2016). Predicting Confusion in Information Visualization from Eye Tracking and Interaction Data. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2529-2535.

Lang, B. (1998). *L'Informatique : Science, Techniques et Outils*. LexiPraxi 98, journée de réflexion sur le thème «Former des citoyens pour maîtriser la société de l'information», Maison de l'Europe (Paris).

Larini, M., & Barthes, A. (2018). *Statistiques et traitement de données quantitatives en éducation : De la collecte au traitement des données* (Vol. 2). ISTE Group.

Lazard, E., & Mounier-Kuhn, P.-E. (2019). *Histoire illustrée de l'informatique* (2ème). EDP sciences.

Leavens, G. (1997). *Programming languages courses : Major programming paradigms*. Iowa State University. http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html

Léonard, M., Peter, Y., Secq, Y., & Fluckiger, C. (2022). Computational Thinking : Focus on Pattern Identification. In I. Hilliger, P. J. Muñoz-Merino, T. De Laet, A. Ortega-Arranz, & T. Farrell (Éds.), *Educating for a New Future : Making Sense of Technology-Enhanced Learning Adoption* (p. 187-200). Springer International Publishing.

Li, N., & Zhang, B. (2021). The Research on Single Page Application Front-end development Based on Vue. *Journal of Physics: Conference Series*, *1883*(1). https://doi.org/10.1088/1742-6596/1883/1/012030

Libert, C., & Vanhoof, W. (2020). *Introduire la concurrence en début de secondaire?* Didapro 20 - L'informatique, objets d'enseignements enjeux épistémologiques, didactiques et de formation.

Limone, P., & Toto, G. A. (2021). Psychological and Emotional Effects of Digital Technology on Children in COVID-19 Pandemic. *Brain Sciences*, *11*(9). https://doi.org/10.3390/brainsci11091126

Lin, Y., & Weintrop, D. (2021). The landscape of Block-based programming : Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages*, *67*, 101075. https://doi.org/10.1016/j.cola.2021.101075

Lindberg, R. S. N., Laine, T. H., & Haaranen, L. (2019). Gamifying programming education in K-12 : A review of programming curricula in seven countries and programming games. *British Journal of Educational Technology*, *50*(4), 1979-1995. https://doi.org/10.1111/bjet.12685

Loisel Decque, M. (2002). *La transition du collège au lycée : Socialisation lycéenne et réussite scolaire* [Thèse de doctorat]. Université de Lille 3.

Luengo, V. (2009). *Les rétroactions épistémiques dans les Environnements Informatiques pour lApprentissage Humain* [Habilitation à Diriger des Recherches, Université Joseph Fourrier]. https://hal.science/hal-00699802

Lutz, M. (2013). *Learning python : Powerful object-oriented programming*. O'Reilly Media, Inc.

Majaj, M. (2011). *L'enseignement de l'arithmétique en France au collège et à la transition collège / lycée* (Numéro 2011LYO10062) [Thèse de doctorat, Université Claude Bernard - Lyon I]. https://theses.hal.science/tel-00598426

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.*, *10*(4). https://doi.org/10.1145/1868358.1868363

Margolinas, C. (1992). Eléments pour l'analyse du rôle du maître : Les phases de conclusion. *Recherches en Didactique des Mathematiques*, *12*(1), 113-158.

Martelli, A., Ravenscroft, A. M., Holden, S., & McGuire, P. (2023). *Python in a Nutshell*. O'Reilly Media, Inc.

Martinand, J.-L. (1989). Pratiques de référence, transposition didactique et savoirs professionnels en sciences et techniques. *Les sciences de l'éducation pour l'ère nouvelle*, *2*, 23-29.

Marwan, S., Akram, B., Barnes, T., & Price, T. W. (2022). Adaptive Immediate Feedback for Block-Based Programming : Design and Evaluation. *IEEE Transactions on Learning Technologies*, *15*(3), 406-420.

Masse, M. (2011). *REST API design rulebook : Designing consistent RESTful web service interfaces*. O'Reilly Media, Inc.

Matsuzawa, Y., Ohata, T., Sugiura, M., & Sakai, S. (2015). Language Migration in Non-CS Introductory Programming through Mutual Language Translation Environment. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 185-190. https://doi.org/10.1145/2676723.2677230

Maxwell, A. E., Warner, T. A., & Fang, F. (2018). Implementation of machine-learning classification in remote sensing : An applied review. *International Journal of Remote Sensing*, *39*(9), 2784-2817.

McCall, D., & Kölling, M. (2014). Meaningful categorisation of novice programmer errors. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, 1-8. https://doi.org/10.1109/FIE.2014.7044420

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, 125-180. https://doi.org/10.1145/572133.572137

McKinney, W. (2011). Pandas : A Foundational Python Library for Data Analysis and Statistics. *Python for high performance and scientific computing*, *14*(9), 1-9.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). Habits of Programming in Scratch. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, 168-172. https://doi.org/10.1145/1999747.1999796

MEN. (1999). Programme de mathématiques de la classe de seconde. In *Bulletin officiel n°6 du 12 août 1999*. Ministère de l'Éducation nationale. https://www.education.gouv.fr/bo/BoAnnexes/1999/hs6/hs6vol2.pdf

MEN. (2009a). Programme de mathématiques de seconde générale et technologique. In *Bulletin officiel n° 30 du 23 juillet 2009*. Ministère de l'Éducation nationale.

https://cache.media.education.gouv.fr/file/30/52/3/programme_mathematiques_seconde_65523.pdf

MEN. (2009b). *Ressources pour la classe de seconde : Algorithmique*. Éduscol. http://mathematiques.ac-bordeaux.fr/txtoff/prog/lycee/2de_09/doc_ress_algo_v25.pdf

MEN. (2013). Organisation des cycles d'enseignement. In *Bulletin officiel n° 32 du 5 septembre 2013*. Ministère de l'Éducation nationale. https://www.education.gouv.fr/bo/13/Hebdo32/MENE1318869D.htm?cid_bo=73449

MEN. (2014). *Lettre de saisine du CSP relative à l'introduction de connaissances et de compétences en informatique et en sciences du numérique dans les programmes de la scolarité obligatoire (cycles 3 et 4)—19 décembre 2014*. https://www.education.gouv.fr/media/73439/download

MEN. (2015). Programme d'enseignement cycle des approfondissements (Cycle 4). In *Bulletin officiel spéciale n° 11 du 26 novembre 2015*. MEN. https://cache.media.education.gouv.fr/file/MEN_SPE_11/76/0/Programme_cycle_4_pour_B.O._1424760.pdf

MEN. (2016a). *Guide pédagogique et didactique d'accompagnement du nouveau programme de technologie*. Éduscol. https://cache.media.eduscol.education.fr/file/Techno/97/1/RA16_C4_TECH_0_Guide_peda_didac_tech_550971.pdf

MEN. (2016b). *Outils d'aide à l'élaboration de la progression pédagogique en technologie*. Éduscol. https://cache.media.eduscol.education.fr/file/Techno/50/8/RA16_C4_TECH_Outil_progression_pedagogique_547508.xlsx

MEN. (2016c). *Ressources d'accompagnement en mathématiques cycle 4 : Algorithmique et programmation*. Éduscol. https://bit.ly/3dnhyBk

MEN. (2017a). Programme de mathématiques de seconde générale et technologique. In *Bulletin officiel n° 18 du 4 mai 2017*. Ministère de l'Éducation nationale. https://cache.media.education.gouv.fr/file/18/95/3/ensel512_maths_757953.pdf

MEN. (2017b). *Ressources pour le lycée général : Algorithmique et programmation*. Eduscol. https://cache.media.eduscol.education.fr/file/Mathematiques/73/3/Algorithmique_et_programmation_787733.pdf

MEN. (2018). Organisation et volumes horaires de la classe de seconde des lycées d'enseignement général et technologique. In *Journal officiel RF n° 0162 du 17 juillet 2018*. Ministère de l'Éducation nationale. https://www.legifrance.gouv.fr/loda/id/JORFTEXT000037202776

MEN. (2019a). *Documents d'accompagnement en algorithmique et programmation pour la classe de seconde et de première*. Éduscol. https://cache.media.eduscol.education.fr/file/Mathematiques/34/0/RA19_Lycee_GT_2-1_MATH_preambule-algorithmique-programmation_1172340.pdf

MEN. (2019b). Programme de mathématiques de seconde générale et technologique. In *Bulletin officiel spécial n° 1 du 22 janvier 2019*. Ministère de l'Éducation

nationale. https://cache.media.eduscol.education.fr/file/SP1-MEN-22-1-2019/95/7/spe631_annexe_1062957.pdf

MEN. (2019c). Programme de mathématiques de seconde générale et technologique. In *Bulletin officiel spécial n°1 du 22 janvier 2019*. MEN. https://cache.media.education.gouv.fr/file/SP1-MEN-22-1-2019/95/7/spe631_annexe_1062957.pdf

MEN. (2019d). Programme de sciences numériques et technologie de seconde générale et technologique. In *Bulletin officiel spécial n°1 du 22 janvier 2019*. MEN. https://cache.media.education.gouv.fr/file/SP1-MEN-22-1-2019/08/5/spe641_annexe_1063085.pdf

MEN. (2019e). Programme de sciences numériques et technologique de seconde générale et technologique. In *Bulletin officiel spécial n° 1 du 22 janvier 2019*. Ministère de l'Éducation nationale. https://cache.media.education.gouv.fr/file/SP1-MEN-22-1-2019/08/5/spe641_annexe_1063085.pdf

MEN. (2019f). Programme de spécialité de numérique et sciences informatiques de première générale. In *Bulletin officiel spécial n° 1 du 22 janvier 2019*. Ministère de l'Éducation nationale. https://eduscol.education.fr/document/30007/download

MEN. (2019g). Programme de spécialité de numérique et sciences informatiques de terminale générale. In *Bulletin officiel spécial n° 8 du 25 juillet 2019*. Ministère de l'Éducation nationale. https://eduscol.education.fr/document/30010/download

MEN. (2019h). *Ressources d'accompagnement pour la mise en œuvre du programme de SNT*. Éduscol. https://eduscol.education.fr/cid143713/snt-bac-2021.html

MEN. (2020a). *Filles et garçons, sur le chemin de l'égalité de l'école à l'enseignement supérieur*. https://www.education.gouv.fr/media/51746/download

MEN. (2020b). Programme d'enseignement du cycle des approfondissements (cycle 4). In *Bulletin officiel n°31 du 30 juillet 2020*. Ministère de l'Éducation nationale. https://www.education.gouv.fr/media/70285/download

MEN. (2022a). *Programme de la session 2023 du concours externe de l'agrégation du second degré—Section informatique*. Ministère de l'Éducation nationale. https://media.devenirenseignant.gouv.fr/file/agreg_externe/50/6/p2023_agreg_ext_informatique_1426506.pdf

MEN. (2022b). *Rapport du jury du concours de l'agrégation externe d'informatique—Session 2022*. Ministère de l'Éducation nationale. https://media.devenirenseignant.gouv.fr/file/agreg_externe/84/0/rj-2022-agregation-externe-informatique_1428840.pdf

MEN. (2023). *Les programmes du collège*. Ministère de l'Education Nationale et de la Jeunesse. https://www.education.gouv.fr/les-programmes-du-college-3203

Merceron, A., & Yacef, K. (2005). *Educational Data Mining : A Case Study*. 467-474.

Meyer, A., & Modeste, S. (2022). *Situation didactique autour d'un jeu de recherche : Expérimentation en classes de NSI*. Didapro 22.

Miljanovic, M. A., & Bradbury, J. S. (2018). A Review of Serious Games for Programming. In S. Göbel, A. Garcia-Agundez, T. Tregel, M. Ma, J. Baalsrud

Hauge, M. Oliveira, T. Marsh, & P. Caserman (Éds.), *Serious Games* (p. 204-216). Springer International Publishing.

Mirabail, M. (1990). La culture informatique. *ASTER*, *11*, 11-28.

Mitchell, J. C. (2003). *Concepts in programming languages*. Cambridge University Press.

Modeste, S. (2012). *Enseigner l'algorithme pour quoi ? Quelles nouvelles questions pour les mathématiques ? Quels apports pour l'apprentissage de la preuve?* [Thèse de doctorat]. Université de Grenoble.

Moresi, M., Gomez, M. J., & Benotti, L. (2021). Predicting Students' Difficulties From a Piece of Code. *IEEE Transactions on Learning Technologies*, *14*(3), 386-399. https://doi.org/10.1109/TLT.2021.3092998

Muratet, M. (2010). *Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation* [Thèse de doctorat]. Université Toulouse III - Paul Sabatier.

Muratet, M., Delozanne, E., Torguet, P., & Viallet, F. (2012). Serious Game and Students' Learning Motivation : Effect of Context Using Prog&Play. In S. A. Cerri, W. J. Clancey, G. Papadourakis, & K. Panourgia (Éds.), *Intelligent Tutoring Systems* (p. 123-128). Springer Berlin Heidelberg.

Narciss, S. (2013). Designing and evaluating tutoring feedback strategies for digital learning. *Digital Education Review*, *23*, 7-26.

Nicodemus, K. K., & Malley, J. D. (2009). Predictor correlation impacts machine learning algorithms : Implications for genomic studies. *Bioinformatics*, *25*(15), 1884-1890.

Nijenhuis-Voogt, J., Bayram-Jacobs, D., Meijer, P. C., & Barendsen, E. (2021). Teaching algorithms in upper secondary education : A study of teachers' pedagogical content knowledge. *Computer Science Education*, 1-33. https://doi.org/10.1080/08993408.2021.1935554

Nørmark, K. (2013). *Functional Programming in Scheme : Overview of the four main programming paradigms*. Aalborg University, Denmark. http://people.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html

Oda, M., Noborimoto, Y., & Horita, T. (2021). International Trends in K–12 Computer Science Curricula Through Comparative Analysis : Implications for the Primary Curricula. *International Journal of Computer Science Education in Schools*, *4*(4), 24-58. https://doi.org/10.21585/ijcses.v4i4.102

Oliphant, T. E. (2007). Python for Scientific Computing. *Computing in Science & Engineering*, *9*(3), 10-20. https://doi.org/10.1109/MCSE.2007.58

O'Regan, G. (2021). *A Brief History of Computing*. Springer International Publishing. https://doi.org/10.1007/978-3-030-66599-9

Page d'accueil Pyrates. (2023). https://py-rates.fr

Papert, S. (1980). *Mindstorms : Children, computers, and powerful ideas*. Basic Books, Inc.

Paris, A. (2021). *Les parcours de formation des enseignants des 1er et 2nd degrés en France*. Cnesco.

Parzen, E. (1962). On estimation of a probability density function and mode. *The annals of mathematical statistics*, *33*(3), 1065-1076.

Passey, D. (2017). Computer science (CS) in the compulsory education curriculum : Implications for future research. *Education and Information Technologies*, *22*(2), 421-443. https://doi.org/10.1007/s10639-016-9475-z

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., & Dubourg, V. (2011). Scikit-learn : Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830.

Pérez, F., Granger, B. E., & Hunter, J. D. (2011). Python : An Ecosystem for Scientific Computing. *Computing in Science & Engineering*, *13*(2), 13-21. https://doi.org/10.1109/MCSE.2010.119

Perugini, S. (2022). *Programming languages : Concepts and implementation*. Jones & Bartlett Learning.

Piech, C., Huang, J., Nguyen, A., Phulsuksombati, M., Sahami, M., & Guibas, L. (2015). Learning Program Embeddings to Propagate Feedback on Student Code. *International Conference on Machine Learning*, 1093-1102.

Price, T. W., Dong, Y., & Barnes, T. (2016). Generating Data-Driven Hints for Open-Ended Programming. *International Conference on Educational Data Mining*, 1914-198.

Price, T. W., Dong, Y., & Lipovac, D. (2017). ISnap : Towards Intelligent Tutoring in Novice Programming Environments. *SIGCSE Technical Symposium on Computer Science Education*, 483-488.

Probst, P., Wright, M. N., & Boulesteix, A. (2019). Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: data mining and knowledge discovery*, *9*(3), e1301.

Programming paradigm. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Programming_paradigm&oldid=991270690

Qian, Y., & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming : A Literature Review. *ACM Trans. Comput. Educ.*, *18*(1). https://doi.org/10.1145/3077618

Qian, Y., & Lehman, J. D. (2016). Correlates of success in introductory programming : A study with middle school students. *Journal of Education and Learning*, *5*(2), 73-83. https://doi.org/10.5539/jel.v5n2p73

Rabardel, P. (1995). *Les hommes et les technologies; approche cognitive des instruments contemporains*. Armand Colin.

Ravel, L. (2003). *Des programmes à la classe : Etude de la transposition didactique interne. Exemple de l'arithmétique en Terminale S spécialité mathématique*. [Thèse de doctorat, Université Joseph-Fourier - Grenoble I]. https://theses.hal.science/tel-00162790

Reeborg's World home page. (2023). https://reeborg.ca

Rennie, L. J. (2003). "Pirates Can Be Male or Female" : Investigating Gender-Inclusivity in a Years 2/3 Classroom. *Research in Science Education*, *33*(4), 515-528. https://doi.org/10.1023/B:RISE.0000005253.72282.f4

Resnick, M., Kafai, Y., Maeda, J., Rusk, N., & Maloney, J. (2003). A networked, media-rich programming environment to enhance technological fluency at after-school centers in economically-disadvantaged communities. *Proposal to National Science Foundation (Project funded 2003-2007)*.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch : Programming for All. *Commun. ACM*, *52*(11), 60-67. https://doi.org/10.1145/1592761.1592779

Reuter, Y. (2004). Analyser la discipline : Quelques propositions. In É. Falardeau, C. Fisher, C. Simard, & N. Sorin (Éds.), *Le français, discipline singulière, plurielle ou transversale, Actes du 9e colloque de l'Association internationale pour la recherche en didactique du français, Québec 26 au 28 août 2004*. Université de Laval.

Richard-Bossez, A., Cornand, R., Hache, C., Lorcerie, F., Audren, G., Lopez, E., Olympio, N., Pavie, A., & Richit, N. (2020). Accompagner la transition collège-lycée : Les enseignements des parcours d'excellence en REP+. *Les Cahiers d'Education & Devenir*, *35/36*, 53-63.

Richardson, V. (2005). Constructivist teaching and teacher education : Theory and practice. In *Constructivist teacher education : Building a World of New Understandings* (p. 3-14). Falmer Press.

Rivers, K., & Koedinger, K. R. (2017). Data-Driven Hint Generation in Vast Solution Spaces : A Self-Improving Python Programming Tutor. *International Journal of Artificial Intelligence in Education*, *27*(1), 37-64.

Robertson, P. K. (1991). A Methodology for Choosing Data Representations. *IEEE Computer Graphics and Applications*, *11*(03), 56-67. https://doi.org/10.1109/38.79454

Robinson, W. (2016). From Scratch to Patch : Easing the Blocks-Text Transition. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*, 96-99. https://doi.org/10.1145/2978249.2978265

Royal Society. (2012). *Shut down or restart ? The way forward for computing in UK schools*. Royal Society. https://royalsociety.org/~/media/education/computing-in-schools/2012-01-12-computing-in-schools.pdf

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in Secondary school : A pedagogical content knowledge perspective. *Informatics in education*, *10*(1), 73-88.

Scott, M. L. (2015). *Programming language pragmatics*. Morgan Kaufmann.

Scratch statistics. (2023). https://scratch.mit.edu/statistics/

Sebesta, R. W. (2012). *Concepts of programming languages-tenth edition*. Pearson.

Shahid, M., Wajid, A., Haq, K. U., Saleem, I., & Shujja, A. H. (2019). A Review of Gamification for Learning Programming Fundamental. *2019 International Conference on Innovative Computing (ICIC)*, 1-8. https://doi.org/10.1109/ICIC48496.2019.8966685

Shute, V. J. (2008). Focus on Formative Feedback. *Review of Educational Research*, *78*(1), 153-189. https://doi.org/10.3102/0034654307313795

Sobral, S. R. (2021). The Old Question : Which Programming Language Should We Choose to Teach to Program? In T. Antipova (Éd.), *Advances in Digital Science* (p. 351-364). Springer International Publishing.

Stefik, A., & Siebert, S. (2013). An Empirical Investigation into Programming Language Syntax. *ACM Trans. Comput. Educ.*, *13*(4). https://doi.org/10.1145/2534973

Storte, D., Webb, M., Bottino, R., Passey, D., Kalas, I., Bescherer, C., Smith, J., Angeli, C., Katz, Y., & Micheuz, P. (2019). Coding, programming and the changing curriculum for computing in schools. *Report of UNESCO/IFIP TC3 Meeting at OCCE*.

Sweller, J. (1988). Cognitive load during problem solving : Effects on learning. *Cognitive science*, *12*(2), 257-285.

Tan, F. E. S., & Berger, M. P. F. (1999). Optimal allocation of time points for the random effects model. *Communications in Statistics - Simulation and Computation*, *28*(2), 517-540. https://doi.org/10.1080/03610919908813563

Task force ACM & IEEE-CS. (2013). *Computer Science Curricula 2013 : Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery. https://doi.org/10.1145/2534860

Tchounikine, P. (2009). *Précis de recherche en ingénierie des EIAH*. http://membres-liglab.imag.fr/tchounikine/Precis.html.

Tiled home page. (2023). https://www.mapeditor.org/

Trouche, L. (2003). *Construction et conduite des instruments dans les apprentissages mathématiques : Nécessité des orchestrations* [Habilitation à Diriger des Recherches]. Université Paris VII.

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society*, *2*(1), 230-265.

Vahldick, A., Mendes, A. J., & Marcelino, M. J. (2014). A review of games designed to improve introductory computer programming competencies. *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, 1-7. https://doi.org/10.1109/FIE.2014.7044114

Vandevelde, I., & Fluckiger, C. (2020, février). L'informatique prescrite à l'école primaire. Analyse de programmes, ouvrages d'enseignement et discours institutionnels. *Colloque Didapro-Didastic 8*. https://hal.univ-lille.fr/hal-02462385

van Laar, E., van Deursen, A. J. A. M., van Dijk, J. A. G. M., & de Haan, J. (2017). The relation between 21st-century skills and digital skills : A systematic literature review. *Computers in Human Behavior*, *72*, 577-588. https://doi.org/10.1016/j.chb.2017.03.010

van Reijmersdal, E. A., Jansz, J., Peters, O., & van Noort, G. (2013). Why girls go pink : Game character identification and game-players' motivations. *Computers in Human Behavior*, *29*(6), 2640-2649. https://doi.org/10.1016/j.chb.2013.06.046

Vilatte, J.-C. (2007). *Méthodologie de l'enquête par questionnaire*. Université d'Avignon.

Waskom, M. L. (2021). Seaborn : Statistical data visualization. *Journal of Open Source Software*, *6*(60), 1-4.

Watson, S., & Richter Lipford, H. (2019). Motivating Students Beyond Course Requirements with a Serious Game. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 211-217. https://doi.org/10.1145/3287324.3287364

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century : Why, what and when? *Education and Information Technologies*, *22*(2), 445-468. https://doi.org/10.1007/s10639-016-9493-x

Węglarczyk, S. (2018). Kernel density estimation and its application. *ITM Web Conf.*, *23*. https://doi.org/10.1051/itmconf/20182300037

Weill-Tessier, P., Kyfonidis, C., Brown, N., & Kölling, M. (2022). Strype : Bridging from Blocks to Python, with Micro:Bit Support. *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 2*, 585-586. https://doi.org/10.1145/3502717.3532155

Weintrop, D. (2019). Block-Based Programming in Computer Science Education. *Commun. ACM*, *62*(8), 22-25. https://doi.org/10.1145/3341221

White, G., & Sivitanides, M. (2005). Cognitive Differences Between Procedural Programming and Object Oriented Programming. *Information Technology & Management*, *6*(4).

Wing, J. M. (2006). Computational Thinking. *Commun. ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Yoon, D.-M., & Kim, K.-J. (2015). Challenges and Opportunities in Game Artificial Intelligence Education Using Angry Birds. *IEEE Access*, *3*, 793-804. https://doi.org/10.1109/ACCESS.2015.2442680

Yücel, Y., & Rızvanoğlu, K. (2019). Battling gender stereotypes : A user study of a code-learning game, "Code Combat," with middle school children. *Computers in Human Behavior*, *99*, 352-365. https://doi.org/10.1016/j.chb.2019.05.029

Zhan, Z., He, L., Tong, Y., Liang, X., Guo, S., & Lan, X. (2022). The effectiveness of gamification in programming education : Evidence from a meta-analysis. *Computers and Education: Artificial Intelligence*, *3*. https://doi.org/10.1016/j.caeai.2022.100096

Zimmermann, P., Flavier, É., & Méard, J. (2012). L'identité professionnelle des enseignants en formation initiale. *Spirale-Revue de recherches en éducation*, *49*(1), 35-50.

COLLEGE DOCTORAL BRETAGNE

EDUCATION, LANGAGES
INTERACTIONS, COGNITION
CLINIQUE, EXPERTISE

Université de Bretagne Occidentale

Titre : Apprentissage de la programmation informatique : analyses et ressources pour accompagner la transition collège-lycée

Mots clés : enseignement-apprentissage de la programmation, transition blocs-texte, jeux sérieux, rétroactions automatiques, analytique des apprentissages, apprentissage automatique.

Résumé : L'objectif de cette thèse est d'abord d'analyser la transition collège-lycée afin de mettre au jour les discontinuités que peuvent rencontrer les élèves lorsqu'ils apprennent à programmer, puis de concevoir une ressource d'enseignement à même de les accompagner dans ces différents changements.
Pour ce faire, nous commençons par réaliser une analyse épistémologique portant sur la programmation informatique, puis effectuons une étude approfondie des curriculums s'appuyant sur la Théorie anthropologique du didactique, et examinons en détails les différences entre les langages Scratch et Python. Nous menons ensuite une enquête auprès de 480 enseignants visant à connaitre leurs pratiques d'enseignement.
En nous basant sur ces analyses préalables, nous concevons l'application en ligne Pyrates sous la forme d'un jeu sérieux d'apprentissage du langage Python. Son environnement est aménagé afin de faciliter la transition blocs-texte, et ses huit niveaux mettent en jeu les concepts fondamentaux de la programmation à travers des situations ludiques conçues selon la Théorie des situations didactiques. Nous évaluons ensuite cette application dans les classes auprès de 240 élèves de seconde. L'étude des traces d'activités générées nous permet de valider globalement cette conception. Enfin, nous créons un système de rétroactions automatiques sélectionnées par des modèles d'IA afin d'améliorer l'autonomie des élèves. Ces modèles, qui font des prédictions en fonction de l'activité des élèves, sont entrainés à partir de données étiquetées par des enseignants lors d'une seconde expérimentation dans les classes de seconde impliquant 215 élèves.

Title : Computer programming education: analyses and resources to support the transition from middle school to high school

Keywords : computer programming education, block-to-text transition, serious games, automatic feedback, learning analytics, machine learning

Abstract : The aim of this thesis is first to analyse the teaching and learning of programming at the transition between lower and upper secondary school (from grade 9 to grade 10) in France in order to enlighten the discontinuities that students may encounter when they learn to program, and then to design teaching materials that can support them through these various changes.
We began with an epistemological analysis of computer programming, followed by an in-depth study of curricula based on the anthropological theory of didactics, and a detailed investigation of the differences between the Scratch and Python programming languages. We then conduct a survey with 480 teachers to find out about their teaching practices.
Based on these preliminary analyses, we designed the Pyrates online application as a serious game for learning the Python language. Its environment was tailored to facilitate the transition from blocks to text, and its eight levels involve the fundamental concepts of programming through playful situations conceived according to the Theory of didactical situations. We then evaluated this application in the field with 240 tenth-grade students. By studying the generated activity traces, we were able to validate the overall design.
Finally, we created an automatic feedback system based on AI models to improve students' autonomy. These models make predictions relying on student activity, and were trained on data labelled by teachers during a second field experiment involving 215 students.