Approche didactique et instrumentale de la pensée informatique : focus sur le concept de motif

Marielle Léonard

Université de Lille

CIREL

CRIStAL
Centre de Recherche en Informatique, Signal et Automatique de Lille

Université de Lille

École doctorale **Sciences de l'Homme et de la Société**

Centres de recherche
Centre Interuniversitaire de Recherche en Éducation de Lille
(CIREL - ULR 4354)
Centre de Recherche en Informatique, Signal et Informatique de Lille
(CRIStAL - UMR 9189)

Thèse présentée par

Marielle Léonard

Soutenue le **13 novembre 2024**

Discipline **Sciences de l'éducation et de la formation**

Spécialité **Didactique de l'informatique**

Approche didactique et instrumentale de la pensée informatique : focus sur le concept de motif

Composition du jury

Rapporteurs	Béatrice Drot-Delange	professeure à l'université Clermont Auvergne
	Sébastien George	professeur à l'université du Mans
Présidente	Clarisse Dhaenens	professeure à l'université de Lille
Examinatrices	Mariam Haspekian	professeure à l'université Paris Cité
	Sandra Nogry	MCF HDR au Cergy-Paris Université
Invitée	Janine Rogalski	université Paris Diderot
Directeurs de thèse	Cédric Fluckiger	professeur à l'université de Lille
	Yvan Peter	professeur à l'université de Lille
	Yann Secq	MCF à l'université de Lille

Colophon

Mémoire de thèse intitulé « Approche didactique et instrumentale de la pensée informatique : focus sur le concept de motif », écrit par Marielle Léonard, achevé le 20 juillet 2024, composé au moyen du système de préparation de document LaTeX et de la classe yathesis dédiée aux thèses préparées en France.

Approche didactique et instrumentale de la pensée informatique : focus sur le concept de motif

Résumé

En France, depuis 2016, l'initiation à la programmation informatique est présente dans les curricula scolaires de l'école obligatoire. L'objectif de cette thèse est de comprendre le processus de conceptualisation lors de la résolution de puzzles de programmation par des sujets âgés de 7 à 15 ans. À cette fin, nous combinons les apports respectifs de la théorie des champs conceptuels (Vergnaud, 1991) et de l'analyse de traces d'interaction dans un EIAH.

Nous nous concentrons sur le concept de motif, en particulier lors des premières confrontations avec la notion de boucle en programmation par blocs. Nous définissons un motif comme « une entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles » et mettons en évidence la place essentielle de ce concept lors de l'initiation à la pensée algorithmique.

L'approche didactique adoptée vise à positionner le concept de motif au sein d'un champ conceptuel des notions de base de l'algorithmique, champ conceptuel qui a pour périmètre la programmation impérative en langage Scratch au niveau de l'école obligatoire. Au sein de ce champ conceptuel, nous approfondissons l'étude des situations de programmation d'un robot virtuel sur une grille qui requièrent l'utilisation d'une boucle.

Notre protocole expérimental est adossé au concours en ligne de programmation Algoréa. Nous construisons un outillage méthodologique incluant un dispositif de collecte de données à trois échelles, des analyses statistiques sur de larges échantillons, une automatisation du traitement de traces d'interaction avec l'EIAH, et des analyses qualitatives d'enregistrements vidéo d'écran. Cet outillage méthodologique, qui permet de combiner précision des analyses qualitatives et robustesse statistique, constitue l'un des apports de la thèse.

Avec cette approche, nous réalisons d'abord une étude instrumentale de l'EIAH telle que la définit Rabardel (1995). Son but est de distinguer ce qui, dans l'activité, relève de la maîtrise conceptuelle et ce qui relève de la maîtrise instrumentale d'un environnement de programmation particulier. Nous nous concentrons ensuite sur la conceptualisation-en-acte au sens de Vergnaud (1991). Nous identifions les schèmes mis en œuvre par le sujet lors de l'activité de programmation étudiée, notamment les invariants opératoires sous-jacents.

Nos analyses nous permettent ainsi d'identifier et de documenter des paliers de difficulté et des erreurs récurrentes lors des premiers apprentissages de la boucle. Une de nos perspectives de recherche est de reproduire cette démarche pour mener des investigations sur l'ensemble des concepts abordés lors de l'initiation à la programmation informatique au niveau de l'école obligatoire. Ces résultats constituent une contribution de nature à outiller les enseignants de l'école élémentaire et du collège pour accompagner leurs élèves et les aider à surmonter les difficultés rencontrées lors de l'apprentissage des concepts fondamentaux de l'algorithmique.

Mots clés : apprentissage de la programmation, eiah, pensée informatique, pensée algorithmique, motif, répétition, boucle

Abstract

In France, since 2016, introduction to computer programming has been included in compulsory school curricula. The objective of this thesis is to understand the conceptualization process when solving programming puzzles by subjects aged 7 to 15 years old. To this end, we combine the respective contributions of conceptual field theory (Vergnaud, 1991) and the analysis of pupils activity in a TEL environment.
We focus on the concept of pattern, in particular during the first confrontations with the loop notion in block programming. We define a pattern as "an entity identifiable within a set because it is repeated identically or with predictable variations" and highlight the essential place of this concept when initiating algorithmic thinking.
The didactic approach adopted aims to position the concept of pattern within a conceptual field of basic notions of algorithms, a conceptual field which has as its scope imperative programming in Scratch language at compulsory school level. Within this conceptual field, we deepen the study of programming situations of a virtual robot on a grid which require the use of a loop.
Our experimental protocol is backed by the Algoréa online programming competition. We are building methodological tools including a data collection device at three scales, statistical analyzes on large samples, automation of the processing of interaction traces with the EIAH, and qualitative analyzes of screen video recordings. This methodological tool, which makes it possible to combine the precision of qualitative analyzes and statistical robustness, constitutes one of the contributions of this thesis.
With this approach, we first carry out an instrumental study of the TEL environment as defined by Rabardel (1995). Its goal is to distinguish what, in the activity, relates to conceptual mastery and what relates to instrumental mastery of a particular programming environment. We then focus on conceptualization-in-act in the sense of Vergnaud (1991). We identify the schemes implemented by the subject during the programming activity studied, in particular the underlying operational invariants.
Our analyzes allow us to identify and document levels of difficulty and recurring errors during the first learning of the loop. One of our research perspectives is to reproduce this approach to carry out investigations on all the concepts covered during the introduction to computer programming at compulsory school level. These results constitute a contribution likely to help elementary and middle school teachers to support their pupils and help them overcome the difficulties encountered when learning fundamental concepts of algorithms.

Keywords: programming learning, technology enhanced learning, computational thinking, algorithmic thinking, pattern, repetition, loop

Avant-propos

Début 2013

ARTHUR: Maman, Gaëtan a trouvé un concours de programmation sur internet, est-ce que je peux le faire ?

10 novembre 2022

ARTHUR: On est sixièmes !!!

À tout domaine ses premiers pas... qui me fascinent. Pour appréhender le travail qui va suivre, il me semble important de situer la genèse de cette thèse.

On ne se hisse pas directement sur les épaules de géants, on se hisse d'abord sur un de leurs orteils. Ces premières marches constituent le socle d'une potentielle future ascension. Les premiers pas sont la source de mon questionnement, de ma motivation et de mon action. Qu'est-ce qui, dans les premiers pas, porte en germe les bases du développement futur ?

J'ai eu l'occasion d'accompagner des premiers pas dans plusieurs domaines : premiers pas en tennis de table entre 1992 et 2018, premiers pas dans le langage oral pour de jeunes élèves de 2 ans en 2010-2011, premiers pas en lecture notamment entre 2011 et 2015.

Concernant les premiers pas en algorithmique et programmation informatique, ils me sont « tombés dessus » en 2013 à travers l'échange dont une réplique est reportée ci-dessus.

Ce premier échange concerne la découverte d'un concours en ligne de programmation, le concours Algoréa, par un enfant de 12 ans qui programme un peu avec son cousin. Sa mère, en l'occurrence moi-même, ne s'intéresse pas plus que ça au domaine. J'ai bien une formation initiale en mathématiques, avec une maîtrise composée pour moitié d'UE d'informatique, mais cela fait 20 ans que je n'ai pas écrit une ligne de code. J'exerce alors comme professeur des écoles en REP+ depuis 2004.

Il s'est alors produit un effet de sidération, je suis restée médusée par l'ascension, qu'on peut qualifier de fulgurante pendant l'année 2013, d'Arthur dans le domaine de l'algorithmique, à tel point que cela m'a amenée à réorienter en profondeur mon activité. Imaginez alors une professeure des écoles en maternelle qui prétend écrire une thèse en didactique de l'informatique. Peu y ont cru au départ et cette aventure a été semée d'un certain nombre d'embûches. Mais j'ai maintenu le cap contre vents et marées, ou plutôt contre stéréotypes et rigidité institutionnelle.

À partir de 2013, nous assistons à deux trajectoires parallèles, qui je pense se nourrissent l'une l'autre. D'un côté un jeune garçon qui se hisse avec frénésie sur les épaules des géants de l'algorithmique, d'un autre sa mère qui fait de son mieux pour l'accompagner, et s'imprègne du domaine en se questionnant et en contribuant sur les premiers pas.

Concernant le second échange presque 10 ans plus tard, il s'agit du message envoyé depuis Dhaka au Bengladesh par Arthur pour m'annoncer le résultat de son équipe à l'ICPC World Finals. Je suis alors dans une phase de rédaction de cette thèse. Dès le lendemain, ce résultat m'inspire cet avant-propos.

Car cette thèse s'inscrit sur une trajectoire qui a été chamboulée, positivement s'entend, par les concours de programmation informatique. Dix ans au cours desquels j'ai découvert le concours Algoréa d'abord en tant que parent, et pris rapidement conscience de la très grande chance qu'avait eue mon garçon de croiser l'association France-ioi. Dix ans au cours desquels j'ai ensuite progressivement participé aux activités de l'association, notamment en organisant des tests d'usage en amont des concours Castor et Algoréa dans des classes d'école élémentaire, puis en contribuant à leur conception auprès de Mathias Hiron, président de l'association, et de son équipe. Dix ans au cours desquels j'ai renoncé à la stabilité professionnelle pour me frayer un chemin hors des sentiers battus vers la didactique de l'informatique. Et déjà 6 ans que j'ai rejoint Yann Secq et Yvan Peter pour l'action de médiation informatique Chticode, qui marque le début de nos expérimentations communes, et la mise sur orbite pour cette thèse.

Cette thèse vient en prolongement de mon mémoire de master en Ingénierie, Médiation, e-Éducation, master suivi totalement à distance entre 2016 et 2020. Elle constitue aussi en quelque sorte une synthèse de mon parcours professionnel antérieur.

Même après plus de 20 ans, les UE d'informatique de ma maîtrise de mathématiques m'ont apporté des bases conceptuelles solides sur lesquelles j'ai pu m'appuyer pour actualiser mes connaissances. Les 5 années en tant d'éducateur sportif professionnel, associées aux 40 années de pratique compétitive et de coaching en tennis de table, m'ont apporté une connaissance fine des mécanismes

de compétition, de ses apports et de ses écueils, et surtout une capacité d'observation et de prise de recul nécessaire en tant que chercheur. Mes 13 années de professorat des écoles m'ont apporté des compétences pédagogiques et un ancrage dans le milieu scolaire. La réflexion sur le protocole expérimental a d'ailleurs émergé au cours des nombreuses observations d'élèves lors des tests d'usage du concours Algoréa, mis en place dans le but de vérifier le bon calibrage des problèmes. J'avais déjà l'intuition à ce moment qu'en instrumentant ces observations, il serait possible d'étudier certains aspects de l'initiation à la programmation informatique. C'est l'objet de cette thèse.

Remerciements

Le présent mémoire de thèse est le fruit de nombreuses années de travail, de persévérance et de collaboration. Au cours de ces années, j'ai bénéficié du soutien de nombreuses personnes et institutions qu'il me tient à cœur de remercier.

En premier lieu je remercie mes trois directeurs de thèse, Cédric Fluckiger, Yvan Peter et Yann Secq, pour leur disponibilité et leurs précieux conseils. Ils ont su me laisser une grande latitude dans ma recherche, tout en me guidant aux moments opportuns. Merci pour cette ambiance de travail ouverte et conviviale.

Mes remerciements vont ensuite à Sébastien George et Béatrice Drot-Delange, qui ont accepté d'être rapporteur et rapporteuse de ma thèse, et à Sandra Nogry, Mariam Haspekian, et Clarisse Dhaenens les autres membres du jury. Sandra Nogry était aussi membre de mon comité de suivi de thèse avec Thibaut Carron. Un grand merci à tous les deux de m'avoir écouté présenter mon travail et pour les conseils donnés à cette occasion. Je voudrais également remercier Janine Rogalski qui a accepté d'échanger avec moi à propos de mes recherches, et d'être invitée à mon jury, son statut administratif ne lui permettant plus d'en faire partie formellement. Merci à George-Louis Baron de nous avoir mises en contact.

Faire une thèse, c'est faire partie d'une équipe, au sein d'un laboratoire. J'ai eu la chance de faire partie de deux équipes. J'exprime ma gratitude à toute l'équipe NOCE du laboratoire CRIStAL pour les projets menés ensemble et tous les précieux moments d'échanges informels. Merci aussi à Maxime Bouton et Gabriel Loiseau, stagiaires de master 2 datascience que j'ai eu le plaisir d'encadrer, pour la collaboration fructueuse. Côté sciences de l'éducation, c'est toute l'équipe Theodile-CIREL que je remercie de m'avoir accueillie. Les séminaires du vendredi, de part leurs discussions stimulantes, ont contribué à élargir ma réflexion en didactique. Un grand merci aussi à tous les doctorants de l'équipe pour l'entraide, notamment lors des séminaires doctorants. Merci à Isabelle, Alain, Kellan, Christèle, Saskia d'avoir assumé l'organisation de ces séminaires. Je suis aussi très reconnaissante envers Valérie Lantoine et l'équipe du secrétariat pédagogique, pour leur altruisme et leur promptitude à me faciliter les démarches de la vie universitaire.

Parmi les doctorants, je voudrais remercier spécialement Isabelle Vandevelde

et Matthieu Branthôme. Nous avons tous trois été dirigés au même moment par Cédric Fluckiger, sur des sujets connexes, et j'ai énormément apprécié la bonne collaboration et les échanges fructueux entre nous.

Cette thèse n'aurait pas vu le jour sans une étroite collaboration avec l'association France-ioi. Je suis très reconnaissante envers Mathias Hiron, son président, et toute son équipe pour avoir mis à ma disposition les ressources qui m'ont permis de construire le protocole expérimental de cette thèse.

Je tiens à exprimer ma gratitude aux organismes financiers et aux partenaires institutionnels, qui ont participé au financement de cette recherche doctorale. Merci aux équipes du projet Interreg France-Wallonie-Vlaanderen Teach Transition et du projet ANR IE-CARE pour leur soutien et pour les nombreuses rencontres qui ont nourri ma réflexion.

Je remercie aussi toutes les personnes qui m'ont mise dans de bonnes conditions pour aborder cette thèse : les enseignants de l'université de Poitiers, notamment Hassina El Kechaï qui a dirigé mon mémoire de Master 2 d'ingénierie Médiation e-Éducation, et la communauté du DIU Apprendre par le jeu, en particulier Julian Alvarez.

Faire une thèse en didactique implique de mon point de vue d'être connecté au terrain et de travailler étroitement avec les praticiens. Merci à Thi-Lan Luu et à l'équipe de la Maison pour la Science de m'avoir intégrée dans l'équipe de formateurs. Chaque session de formation fut une occasion de relier réflexion théorique et retours de terrain. Merci aux inspecteurs, aux Enseignants Référents aux Usages du Numérique (ERUN) de l'Éducation nationale, et spécialement aux enseignants qui m'ont accueillie dans leur classe de m'avoir fait confiance. Et un merci tout particulier aux 23 jeunes pour leur participation assidue aux sessions Algoréa à distance. Ces rendez-vous, plusieurs fois par an, ont été des moments de riches échanges, qui, pour certains, ont perduré au-delà de cette thèse.

Pendant cette aventure de plusieurs années, j'ai bénéficié du soutien de tous mes proches, et je leur en suis profondément reconnaissante. Un grand merci à mes parents qui, par leur rigueur et leur exigence, m'ont donné le sens de l'effort et le goût d'apprendre. Pour mon père, parti en avril 2023 et qui ne verra pas l'aboutissement de ce travail, je resterai l'« éternelle étudiante », comme il aimait à m'appeler. Merci à mes enfants, Camille, Sébastien et Arthur et à mon filleul, Gaëtan, qui m'ont accompagnée dans cette aventure. Les repas à la maison ont souvent ressemblé à des discussions de laboratoire, et j'ai vraiment de la chance de pouvoir m'appuyer sur une aussi grande expertise familiale en informatique et édition numérique. Et un grand merci tout particulier à Vincent, avec qui je partage ma vie depuis plus de 30 ans, qui m'a encouragée à m'engager dans cette aventure, a su s'adapter à mes contraintes de thésarde, et me soutient quotidiennement tout en restant discret.

À toutes et tous, je vous adresse mes plus sincères remerciements. Votre présence à mes côtés et vos encouragements ont été déterminants dans l'aboutissement de ce mémoire de thèse.

Sommaire

Introduction

L' objectif de cette thèse est d'étudier les premières confrontations d'élèves de 7 à 15 ans avec les notions algorithmiques de base dans un environnement de programmation par blocs. Il s'agit d'identifier les paliers de difficultés, les erreurs récurrentes et leur conceptualisation sous-jacente lors des premiers apprentissages en algorithmique, notamment lors de l'introduction de la boucle.

Contexte de la réintroduction de l'informatique dans les curricula scolaires de l'école obligatoire

En 2006, Wing publie un article intitulé « *Computational thinking* » dans lequel elle appelle à l'enseignement de la pensée informatique pour tous et assez tôt dans la scolarité (Wing, 2006). Wing considère la pensée informatique comme une compétence de base au même titre que la lecture, l'écriture et le calcul. L'article a déclenché un vif intérêt au sein des communautés éducatives et de recherche de nombreux pays, qui ont pris conscience de la nécessité de faire évoluer leur curricula scolaires pour répondre aux défis de la place grandissante du numérique dans la société. Ainsi, en 2020, de nombreux pays d'Europe disposent d'enseignements d'informatique dans leurs curricula scolaires (Commission européenne, 2022).

Il nous semble nécessaire de préciser dès maintenant à quoi réfère le mot *informatique* dans ce document. Le terme *informatique* est introduit par Dreyfus en 1962 et repris par l'Académie des sciences en 1966 pour désigner la « science du traitement rationnel, notamment à l'aide de machines automatiques, de l'information considérée comme le support des connaissances et des communications dans les domaines technique, économique et social ». Selon le philosophe Mirabail, repris par Fluckiger, l'informatique revêt une triple nature. Elle peut être considérée sous l'angle de la science, de la technologie et des usages (Fluckiger, 2019). Dans le rapport de l'Académie des sciences de 2013 (Berry et al., 2013), le mot *informatique* désigne spécifiquement la science et la technique du traitement de l'information, ce qui comprend seulement les deux premières composantes mentionnées par Fluckiger. Les usages sont plutôt associés à l'ad-

jectif *numérique*, qui selon ce même rapport, peut être accolé à toute activité basée sur la numérisation et le traitement de l'information. Dans la suite de ce document, nous adoptons cette distinction, *informatique* pour les composantes scientifique et technologique, *numérique* pour la composante des usages, communément désignés par l'expression *usages du numérique*. La présente recherche porte principalement sur la composante scientifique de l'informatique.

Pour la France, le rapport de l'Académie des sciences déjà cité et intitulé « L'enseignement de l'informatique en France. Il est urgent de ne plus attendre » rend compte de la prise de conscience du décalage entre l'explosion des usages du numérique et la quasi-absence d'enseignement de science informatique (Berry et al., 2013). Ce rapport déclenche la décision d'un retour conséquent des dimensions scientifique et technologique de l'informatique comme objet d'enseignement dans les curricula scolaires, après plusieurs décennies d'un enseignement centré sur les usages (Baron et al., 2015). Ce retour est effectif dans l'ensemble du cursus primaire et secondaire depuis 2016, sans que l'informatique ne soit instituée comme matière scolaire autonome au niveau de l'école obligatoire.

De premiers bilans révèlent les difficultés conséquentes que pose cet enseignement. D'après une première étude menée par Roche et al. à la fin de l'année scolaire 2017, si les professeurs des écoles sont plutôt convaincus de l'utilité de l'enseignement de la programmation informatique, ils font part de difficultés à s'approprier cet enseignement et à le mettre en œuvre (Roche et al., 2018). Ces résultats sont confirmés quatre ans plus tard dans le rapport publié par l'Inspection générale de l'éducation, du sport et de la recherche (IGÉSR) (Gaubert-Macon et al., 2022). Celui-ci pointe une pratique de l'informatique insuffisante pour les élèves de cycle 3 et 4 et un manque de formation des enseignants aux concepts de l'informatique. Le rapport mentionne notamment un temps de pratique insuffisant pour les élèves, le manque de formalisation des problèmes en amont de la programmation et le manque de consolidation des apprentissages. Il souligne la nécessité d'engager de plus amples recherches didactiques dans le domaine.

Appui sur des expérimentations en amont

Pour sa part, l'Université de Lille s'intéresse depuis 2014 aux premiers apprentissages en algorithmique et programmation. Nous avons notamment mené plusieurs études exploratoires sur l'initiation à la notion de boucle en programmation par blocs. Une première expérimentation menée en 2018 auprès de 447 élèves de 8-10 ans a permis de conclure que cette notion de boucle est accessible à des élèves de 8-10 ans et repose sur leur capacité d'identification et de synthèse de régularités (Léonard et al., 2019 ; Peter et al., 2020). Cette expérimentation

nous a amené à l'ébauche du concept de motif. Les expérimentations suivantes ont été conduites auprès d'élèves plus jeunes, de 6-7 ans puis de 5-6 ans, en mobilisant un scénario ludopédagogique articulant activités débranchées et programmation sur tablette (Léonard et al., 2020 ; Léonard et al., 2021). Pour cette tranche d'âge, nous avons montré que l'initiation à la notion de boucle est accessible à travers des tâches de reproduction de frises colorées. En revanche, dans un contexte de déplacement de robot virtuel en orientation relative, la gestion du repérage dans l'espace (gauche/droite) associé au fait de ne pas pouvoir visualiser le motif constituent des obstacles majeurs à l'identification de motifs redondants.

Pour toutes ces études de cas, nous avons utilisé chticode.algorea.org, une plateforme mise à disposition par l'association France-ioi. Lors de ces expérimentations, nous avons constaté une difficulté récurrente lors du passage d'une à plusieurs instructions dans le corps de la boucle. Identifier les instructions à répéter, que nous appelons motif, semble une opération difficile pour les élèves. Nous constatons cette difficulté à travers l'analyse des traces d'interaction des élèves avec l'environnement de programmation, mais une limite de ces premières expérimentations concerne l'absence de cadre d'analyse dans lequel interpréter nos résultats.

Ébauche de la problématique

La présente recherche doctorale s'inscrit dans la continuité de ces études exploratoires relatives à l'initiation à la pensée informatique. L'objectif est d'appréhender plus finement et d'élargir les premiers résultats obtenus, tant du point de vue théorique que du point de vue empirique.

Notre premier axe de travail concerne le concept de motif que nous avons utilisé sans creuser ni sa définition, ni sa portée théorique. Qu'est-ce qu'un motif dans le contexte de l'initiation à la programmation ? Dans quelles structures algorithmiques ce concept est-il impliqué ?

Il nous semble d'autre part absolument nécessaire de nous doter d'un cadre d'analyse à même de nous permettre d'interpréter nos résultats en nous situant dans une approche didactique, tout en prenant en compte la dimension de l'usage de l'environnement de programmation. Pour mener cette approche didactique, nous avons choisi de nous inscrire dans le cadre de la théorie des champs conceptuels de Vergnaud (1991). Ce cadre d'analyse, qui étudie la conceptualisation dans le cas d'activités cognitives complexes dont la programmation fait partie, nous procure des outils efficaces pour l'interprétation de nos résultats expérimentaux. Doté de ce cadre d'analyse, notre objectif est d'appréhender l'importance de l'identification de motif lors de l'initiation à la programmation

dans un environnement de programmation par blocs, en particulier lors de l'apprentissage de la boucle.

Choix méthodologiques

Les expérimentations dans les années 1980 qui ont supporté l'élaboration de la théorie des champs conceptuels ont été menées à l'échelle de classes. Elles consistaient en des analyses qualitatives à partir d'entretiens et de productions d'élèves. La méthode utilisée pour obtenir les résultats de nos premières expérimentations est différente. Ils ont surtout été établis à partir des traces d'interaction avec un environnement de programmation. Cette différence constatée nous semble offrir des perspectives intéressantes. Nous proposons de mobiliser le cadre d'analyse de la théorie des champs conceptuels de Vergnaud pour interpréter des traces d'interaction avec cet environnement de programmation. À notre connaissance, ce cadre théorique associé à l'analyse de ce type de données n'a pas encore été exploré pour la construction d'un champ conceptuel relatif à la programmation par blocs.

C'est pourquoi nous avons fait le choix de cibler nos investigations sur la programmation d'un robot virtuel sur une grille dans un environnement de programmation par blocs, tâche pour laquelle nous sommes en mesure de collecter des traces d'interaction avec une très fine granularité. Pour l'environnement de programmation Algoréa, déjà utilisé pour les expérimentations exploratoires, nous disposons de premiers outils de traitement et de visualisation de traces construits lors de notre travail de master, que nous étoffons dans le cadre de cette thèse. De plus, afin de disposer de données à très large échelle, nous adossons notre protocole expérimental au concours de programmation en ligne Algoréa organisé par l'association France-ioi, dont la participation avoisine les 250 000 élèves du CM1 à la terminale ces dernières années.

Ainsi circonscrite, notre recherche porte principalement sur la composante scientifique de l'informatique, sur ses bases conceptuelles. Cependant, nous nous trouvons dans le cas d'une dialectique objet/outil au sens de Douady (1986). En effet, bien que ce soient les notions algorithmiques qui constituent l'objet de l'apprentissage lors de l'activité de programmation, l'informatique a aussi un statut d'outil. Les élèves utilisent un ordinateur ou une tablette pour programmer, ce qui introduit de fait une dimension technologique même si l'objet de notre recherche ne porte pas sur la compréhension du fonctionnement de ce matériel. D'autre part, ils utilisent un environnement de programmation qui est un logiciel dont l'objet est d'apprendre à programmer, ce qui introduit l'usage d'un outil informatisé, dont la prise en main est à questionner. C'est pourquoi nous complétons notre approche didactique par une approche instrumentale au

sens de Rabardel (1995), afin d'étudier la prise en main de l'environnement de programmation. Cette double approche, didactique et instrumentale, situe notre recherche à l'intersection du champ de la didactique de l'informatique et de celui des Environnements Informatiques pour l'Apprentissage Humain (EIAH).

Enjeux de la recherche

Les enjeux des recherches en didactique de l'informatique sont importants du point de vue de la compréhension des premiers apprentissages en algorithmique et de leur mise à disposition pour la formation des praticiens. Des projets tant nationaux qu'européens sont menés pour supporter ce besoin en formation et en recherche. S'inscrivant dans ce contexte, notre recherche a été financée successivement dans le cadre de deux projets. Le premier, de 2020 à 2022, est le projet Interreg France-Wallonie-Vlaanderen « *Digital Transition for Teaching* » (*Teach Transition*) dont l'objet a été de mettre en place un dispositif cadre et une formation aux compétences numériques à destination des enseignants et formateurs. La didactique de l'informatique est l'un des axes de ce projet, qui s'est nourri du présent travail de recherche. Le second est le projet ANR IE-CARE qui est un projet de recherche pluridisciplinaire visant à étudier les modalités et les ressources pour l'apprentissage et l'enseignement de l'informatique à l'école obligatoire. Nous avons rejoint ce projet en janvier 2023. Au-delà de l'implication dans ces deux projets, les résultats de cette recherche visent plus largement à être mis à disposition des enseignants en charge de l'initiation à l'informatique, afin de leur fournir des repères conceptuels sur les procédures expertes et les difficultés récurrentes de leurs élèves.

Plan du mémoire de thèse

Notre mémoire de thèse est structuré en quatre parties. Une première partie, qui comprend trois chapitres, explore le cadre dans lequel s'inscrit notre recherche. Le chapitre 1 vise à la positionner dans le contexte général de l'informatique scolaire : documents prescriptifs, pratiques dans les classes, approches de la pensée informatique. Dans le chapitre 2, nous resserrons nos analyses sur la programmation par blocs, puis encore plus précisément sur les tâches fermées de déplacement d'un robot virtuel sur une grille, évaluées automatiquement. Le chapitre 3 est consacré au concept de motif, central dans notre recherche. Nous construisons une définition d'un motif dans le champ de l'initiation à l'informatique. Puis nous investiguons la place de ce concept dans les travaux sur la pensée informatique, dans les travaux en didactique des mathématiques, ainsi que dans les documents émanant de l'institution scolaire en France.

Dans une deuxième partie composée également de trois chapitres, nous présentons les cadres d'analyse que nous convoquons et construisons notre problématique. Le chapitre 4 est consacré à la théorie des champs conceptuels de Vergnaud (1991). Nous y abordons les concepts centraux de conceptualisation-en-acte, classe de situation, schème et champ conceptuel. Nous positionnons ensuite notre recherche par rapport aux travaux dans le champ de l'initiation à l'informatique qui mobilisent ce cadre d'analyse. Dans le chapitre 5, nous poursuivons en présentant les aspects de l'approche instrumentale de Rabardel (1995) sur lesquels nous nous appuyons, notamment les trois pôles d'une situation instrumentée, la distinction entre schème d'usage et schème d'action instrumentée, les genèses instrumentales. Avec l'ensemble des éléments contextuels et des outils d'analyse convoqués, nous affinons alors notre problématique dans le chapitre 6.

Nous détaillons la méthodologie et le cadre expérimental de l'étude dans une troisième partie, à nouveau divisée en trois chapitres. Dans le chapitre 7, nous analysons l'environnement de programmation Algoréa ainsi que le concours du même nom, qui constituent le support de notre dispositif expérimental. Nous décrivons ensuite notre démarche de collecte de données dans le chapitre 8. Cette démarche comporte plusieurs dimensions, qui structurent notre protocole expérimental. Dans le chapitre 9, nous expliquons comment nous avons articulé des méthodes quantitatives et des méthodes qualitatives de traitement et d'analyse de données, en nous référant au cadre d'analyse de Vergnaud.

La quatrième partie de ce document, qui comprend quatre chapitres, rapporte les résultats de notre recherche. Dans le chapitre 10, nous réalisons une étude instrumentale de l'environnement de programmation Algoréa considéré comme un EIAH. Dans le chapitre 11, nous présentons nos résultats théoriques à propos du concept de motif ainsi que les résultats en termes de classes de situations relatives à ce concept de motif. Ces résultats sont obtenus par une analyse quantitative de données issues du concours Algoréa à l'échelle nationale. Le chapitre 12 apporte des précisions sur les premiers apprentissages de la boucle. Nous y détaillons le passage de la séquence à la boucle, le passage d'une à plusieurs instructions dans le corps de la boucle, en nous appuyant sur les procédures expertes et les principales erreurs. Notre analyse se poursuit dans le chapitre 13 par une étude de l'identification et du dénombrement de motifs lors de la programmation d'une boucle. Nous complétons ce chapitre par une ouverture vers d'autres notions algorithmiques pour lesquelles l'identification de motifs est impliquée, ainsi que par une analyse de l'effet de certaines fonctionnalités de l'EIAH sur l'activité de résolution de problème.

Un dernier chapitre synthétise nos résultats et ouvre les perspectives de poursuite de cette recherche.

Première partie

Éléments de contexte et d'état de l'art

Le lecteur trouvera dans cette première partie des éléments de contexte et d'état de l'art à propos de l'apprentissage de la programmation informatique par des élèves de 7 à 15 ans. Notre état de l'art vise à dresser un état des connaissances empiriques de l'apprentissage de la programmation par blocs, et plus précisément de l'identification de motif et de l'apprentissage de la boucle. Nous adoptons une organisation qui inclut dans chaque chapitre, en plus de sections strictement consacrées à l'état de l'art, des discussions et analyses épistémologiques au service de nos axes de recherche. Cette partie comporte trois chapitres thématiques. Le premier vise à circonscrire le périmètre de notre recherche. À partir d'une première section très large, qui aborde successivement les acquis de la psychologie de la programmation, les concepts intégrateurs de l'informatique et la pensée informatique, nous resserrons ensuite successivement notre propos sur l'informatique à l'école obligatoire en France, les notions de base en algorithmique, et enfin l'apprentissage de la boucle bornée. Le deuxième chapitre fait le point sur les environnements de programmation par blocs utilisés pour l'initiation de jeunes élèves à l'algorithmique. Le troisième répertorie ce que l'on sait du concept de motif dans le champ de l'initiation à l'informatique et aux mathématiques.

Chapitre 1

Contexte général

Sommaire du présent chapitre

Une approche didactique est caractérisée par la focalisation sur les contenus et sur leurs relations à l'enseignement et aux apprentissages (Reuter et al., 2013). L'objet de ce premier chapitre est de délimiter notre recherche et de la positionner dans le contexte général de la didactique de l'informatique pour le niveau de l'école obligatoire. Nous posons le cadre épistémologique de la recherche : quels types de savoirs sont en jeu lors de l'initiation à la programmation informatique ?

Nous caractérisons l'approche didactique que nous adoptons en référence à quelques éléments historiques et à quelques approches répertoriées. Nous convoquons quelques repères dont le but est d'ancrer notre démarche, sans viser ni une exhaustivité, ni un niveau de détails très fin pour ce chapitre d'amorce.

1.1 La recherche en didactique de l'informatique

Historiquement, la recherche en didactique suit la présence de la science informatique dans les curricula scolaires (Baron & Bruillard, 2001, 2011 ; Baron & Drot-Delange, 2016b). Elle est très active pendant les années 1980 pour se tarir pendant les décennies suivantes, avant un renouveau à partir des années 2010.

1.1.1 Les acquis de la psychologie de la programmation des années 1980

Au cours de la décennie 1980-1990, de manière concomitante à l'option informatique des lycées, les recherches se sont concentrées sur les aspects cognitifs lors de l'activité de programmation. Rogalski présente un bref résumé de cet ensemble de travaux regroupés sous le terme de *psychologie de la programmation* dans le chapitre de revue « Psychologie de la programmation, didactique de l'informatique, déja une histoire... » (Rogalski, 2015).

Trente ans plus tard, en 2017, Lagrange et Rogalski dressent un bilan des recherches en didactique de l'informatique au moment où l'informatique est réintroduite dans l'ensemble des curricula scolaires (Lagrange & Rogalski, 2017). Leurs analyses se situent au niveau du lycée, pour la programmation avec des langages textuels. Ils mentionnent que les questions de recherche autour de l'apprentissage des notions de base en algorithmique ont surtout été investies par des chercheurs en psychologie cognitive. Les auteurs s'attachent à montrer que les questions déjà identifiées dans les années 1980 restent d'actualité. Aussi bien les questions abordées lors de cette période que les cadres théoriques mobilisés restent pertinents. Les technologies ont évolué et les paradigmes de programmation se sont enrichis, mais un noyau conceptuel persiste, pour lequel les analyses didactiques menées restent valides. Les difficultés conceptuelles

lors de l'initiation à la programmation sont relativement stables quels que soient le langage et le dispositif informatique utilisés. Les auteurs invitent à mobiliser les acquis de la psychologie de la programmation dans le contexte actuel, et à s'appuyer sur ces acquis pour explorer de nouvelles problématiques. Une de ces nouvelles problématiques concerne la programmation par blocs, pour lequel le travail d'analyse didactique reste à approfondir. Lagrange et Rogalski invitent à mettre en regard les analyses adossées aux expérimentations en langage LOGO mises en place dans les années 1980 et se situant dans une approche constructionniste portée par Papert (1980) et la programmation en Scratch dans le contexte actuel (Lagrange & Rogalski, 2017).

Le présent travail de recherche s'inscrit dans l'exploration des problématiques relatives à la programmation par blocs. Dans cette optique, les acquis de la psychologie de la programmation constituent des points de repère pour une mise en perspective.

1.1.2 Les concepts intégrateurs de la science informatique

En vue de son enseignement, chaque discipline scolaire se construit autour d'une *matrice disciplinaire* qui articule des objets au sens matériel du terme, des tâches, un petit nombre de concepts intégrateurs qui constituent les clés de lecture de la discipline, et des connaissances procédurales (Develay, 1993). Considérer l'informatique en tant que discipline scolaire implique donc entre autres de rechercher quels sont les concepts intégrateurs de l'informatique en tant que discipline scientifique. L'informatique étant encore une science *jeune*, une des difficultés est de caractériser un ensemble de concepts stables temporellement. Une autre difficulté est que les concepts retenus doivent être indépendants de la technologie et des usages, les deux autres facettes de l'informatique, qui, elles, évoluent rapidement.

Dans le cas de l'informatique, un obstacle à sa construction en tant que discipline scolaire en France est qu'elle ne constitue pas une discipline autonome à l'école primaire et au collège. Des fragments de discipline informatique sont inclus dans le cursus général (école élémentaire) ou dans d'autres matières (collège). Nous y reviendrons dans la section 1.2. En revanche, à partir des années 1980, l'informatique a été de manière discontinue et est actuellement une discipline autonome au lycée. Ainsi, c'est en considérant presque exclusivement le cursus de lycée que des éléments de matrice disciplinaire ont été dégagés lors d'études épistémologiques de la discipline (Dowek, 2011). En particulier, des éléments épistémologiques relatifs à la science informatique sont directement à la base de la structuration du programme de la spécialité de terminale ISN ouverte en 2012 et de l'écriture d'un manuel pour l'enseignement de cette matière scolaire (Archambault et al., 2012).

Les quatre concepts retenus par Dowek en 2011 qui, considérés ensemble, spécifient la science informatique sont les concepts d'algorithme, de machine, de langage et d'information (Dowek, 2011). Ces quatre concepts suffisent pour spécifier l'informatique en tant que champ scientifique et lui garantir une singularité et une stabilité temporelle. Nous nous appuyons sur ces quatre concepts, que nous retrouvons à quelques nuances près dans les publications d'autres auteurs, à des périodes différentes.

Rogalski et Vergnaud, dont les travaux sont bien antérieurs à ceux de Dowek, parle d'« interactions qui existent entre la représentation des données, le traitement logique, le langage, les techniques. » (Rogalski & Vergnaud, 1987). Le traitement logique est à rapprocher du concept d'algorithme chez Dowek, la représentation des données de celui d'information, et les techniques de celui de machine. Pour Berry, les concepts de langage, d'algorithme, de machine, et de données sont les *piliers de la science informatique*. Berry y adjoint le concept d'interface, que nous aborderons dans le chapitre suivant (Berry, 2017). Quant à Delmas-Rigoutsos, il propose de considérer *les machines et leurs réseaux*, *les algorithmes et les programmes*, *les données et leur codage* (Delmas-Rigoutsos, 2018).

Deux des quatre concepts identifiés, information et machine, sont directement issus de la définition de l'informatique par l'Académie française en 1966 : « science du traitement rationnel, notamment à l'aide de machines automatiques, de l'information considérée comme le support des connaissances et des communications dans les domaines technique, économique et social ». La mobilisation des deux autres concepts est une nécessité pour le traitement de l'information par des machines automatiques. Nous retrouvons cette structuration résumée dans la caractérisation de l'informatique par Baron : « si ce qui définit la spécificité de l'informatique c'est fondamentalement la question du traitement de l'information, indépendamment de son sens, par des machines automatiques, ce traitement s'effectue toujours par un programme, écrit dans un langage de programmation. » (Baron, 2017). Les paragraphes suivants visent à préciser la portée de ces quatre concepts.

Information et données

Le concept d'information fait l'objet de nombreux débats et nous observons des nuances de sens suivant les périodes et les disciplines où il est convoqué (Leleu-Merviel & Useille, 2008). Dans la définition de l'informatique, l'information est le « support formel des connaissances » (Arsac, 1987), qui prend une *forme* lui permettant d'être traitée par une machine de manière automatique (Baron & Bruillard, 2001). D'une part, le passage de la connaissance, du ressort du cerveau humain, à l'information introduit un *certain appauvrissement* (Arsac,

1987) par le nécessaire choix de ce qui est fixé sur un support, que ce support soit matériel ou formel. D'autre part, la mise en forme de l'information consiste à l'exprimer de manière symbolique, sous la forme de nombres, autrement dit à la *numériser*. Cette numérisation consiste aussi à considérer un support formel vs matériel, et conduit à rendre l'information indépendante de son support matériel. Autrement dit, avec l'informatique, le lien entre l'information et son support est brisé (Berry, 2014). Berry parle de *dématérialisation* de l'information (Berry, 2017).

Une information comprend une donnée brute, qui est une valeur, et une sémantique, qui dépend de son contexte d'interprétation, qui lui est du ressort de l'humain (Berry, 2017, p. 71). Une *donnée* est donc une information représentée dans un codage spécifique, ici symbolique, et dont on ne considère plus l'aspect sémantique. Même s'il est possible d'envisager des données autres que des données exprimées de manière symbolique (Dowek, 2011), l'emploi du terme *données* sous-entendra *données exprimées de manière symbolique* dans la suite de notre propos. « Les données sont l'objet de travail de l'informatique. Ce sont des quantités brutes numérisées ou des groupements structurés d'autres données. » (Berry, 2017, p. 67). Une caractéristique des données est que ce sont des valeurs discrètes, même si la fréquence d'échantillonnage est potentiellement très élevée.

Les données sont traitées automatiquement par la machine indépendamment de l'information qu'elles portent. Pour Berry, le passage entre information et données est *délicat* (Berry, 2017). L'information sous-entend le raisonnement humain, qui dans un sens construit une signification, de la connaissance à partir de données brutes, et dans l'autre sens exprime une connaissance de manière symbolique.

Machine

Une machine est « un outil, c'est à dire un système matériel, qui obéit donc aux lois de la physique. » (Dowek, 2011), « l'objet matériel capable de faire les calculs nécessaires pour transformer les programmes en actions. » (Berry, 2017).

L'ordinateur occupe une place particulière parmi les machines. C'est une *machine universelle* dans le sens où « ses mêmes composants physiques peuvent être utilisés identiquement pour toutes les tâches liées à l'information et au calcul » (Berry, 2017, p. 44). Une spécificité de l'informatique tient à l'utilisation de cette machine universelle, l'ordinateur, qui exécute des tâches non définies au moment de sa conception (Baron & Bruillard, 2001).

Comme système matériel, le concept de machine concerne deux des facettes de l'informatique, celles de science et de technologie. Dans le contexte scolaire, l'architecture matérielle d'un ordinateur est à placer du côté de la technologie.

Algorithme et programme

Un ordinateur est en mesure d'« exécuter n'importe quel algorithme opérant sur des données symboliques » (Dowek, 2011). Cela dit, le concept d'algorithme est bien antérieur à l'invention de l'ordinateur. Le terme *algorithme* vient du nom du mathématicien persan du IX^{ème} siècle, Al-Khwârizmî, auteur d'un ouvrage en arabe dont la traduction en français est « Algoritmie des nombres indiens ».

Nous relevons plusieurs définitions du concept d'algorithme, de la plus simple et concrète à la plus précise et abstraite.

- « Recette qui permet de résoudre un certain problème de manière systématique » (Dowek, 2011). Dowek rapproche un algorithme d'une recette de cuisine, en tant que procédé de fabrication.
- « Processus de calcul automatisable, c'est à dire exécutable de façon systématique par des machines ne comportant aucune intelligence, quel que soit le sens de ce mot » (Berry, 2017, p. 74), « mécanisme conceptuel de calcul systématique » (Berry, 2017, p. 21), « procédé de calcul effectif » (Berry, 2017, p.50).
- « Procédure formalisée et systématique résolvant un problème posé dans un cadre formel » (Delmas-Rigoutsos, 2018). Delmas-Rigoutsos adjoint une définition simplifiée qui viserait des élèves à partir du cycle 2 : « procédure systématique, abstraite et finalisée » (Delmas-Rigoutsos, 2018)
- « Procédure systématique permettant de résoudre une classe de problèmes. À partir d'une entrée (représentant une instance du problème), un algorithme suit un ensemble déterminé de règles et, en un nombre fini d'étapes, produit une sortie (représentant une réponse à l'instance donnée). » (Modeste, 2012)

Pour Berry, d'une part les algorithmes forment « le coeur et la vraie spécificité de l'informatique. » (Berry, 2017, p. 74), et d'autre part ce sont des « objets de nature mathématique » (Berry, 2017, p. 81). Un algorithme relève du *traitement logique* (Rogalski & Vergnaud, 1987). Il est indépendamment du langage, oral ou écrit, utilisé pour le formuler.

Lorsqu'il est exprimé dans un langage informatique en vue de son traitement par une machine, l'algorithme devient un programme. « Les algorithmes sont des procédures de calcul abstraites pour transformer des informations ; les programmes en sont leurs incarnations concrètes » (Knuth & Cégielski, 2011, p. 317). Le passage de l'algorithme au programme consiste en une « "traduction" de l'algorithme dans un langage acceptable par le système informatique sur lequel on travaillera. » (Rogalski, 1987)

Un algorithme permet ainsi de concevoir un traitement automatique des données, c'est-à-dire un traitement qui ne requiert pas de raisonnement humain lors de son exécution.

Langage

Dans le paragraphe précédent, nous avons déjà mentionné que le passage d'un algorithme à un programme implique nécessairement l'utilisation d'un langage. En effet, pour que le traitement de l'information puisse être effectué de manière automatique par un ordinateur, « il faut rédiger la description de la méthode de traitement (algorithme) dans un langage interprétable par l'ordinateur » (Arsac, 1987).

Dès 1970, les travaux issus du colloque de Sèvres, mentionnés par Arsac , pointent l'importance du langage dans ce qui constitue l'informatique : « L'informatique est avant tout un langage, un système de signes qui permet de communiquer au même titre que d'autres langages tels que les mathématiques ou les langues. » (Arsac, 1987)

Pour qu'un algorithme puisse être exécuté par une machine, il est nécessaire de l'exprimer dans un langage simple, précis, non ambigu. Selon Berry, « Un langage [informatique] se définit par trois caractéristiques : ses principes, sa syntaxe (textuelle ou graphique), qui décrit les programmes légaux du point de vue de leur seule écriture, et sa sémantique, c'est à dire le sens donné à ses phrases. » (Berry, 2017, p. 84)

Lagrange et Rogalski expliquent comment opèrent de tels langages : « Les langages informatiques se définissent comme des langages formels utilisés pour l'exploitation d'un système. Nous désignerons par *dispositif*, tout système commandé par un tel langage. Un texte syntaxiquement correct écrit avec le langage peut être interprété comme organisant un *traitement*, c'est-à-dire une suite de modifications apportées à l'état du dispositif. » (Lagrange & Rogalski, 2017)

Parmi les langages utilisés en informatique, on distingue les langages de haut niveau et le langage machine, le seul exécutable par un processeur d'ordinateur. On passe de l'un à l'autre par des processus de traduction automatique (Baron & Bruillard, 2001). Les langages de programmation de haut niveau sont d'une part lisibles par les humains et d'autre part traduits en langage machine.

Contextualisation à notre recherche

L'automatisation du traitement de l'information est l'aspect central de la définition de l'informatique. Dans le contexte de l'initiation, nous parlerons de passage du *faire* au *faire faire*. Nous retenons ces concepts piliers comme clé de structuration de nos investigations sur les premiers apprentissages de la programmation.

Bien que les besoins de structuration du propos nous aient amené à considérer les concepts intégrateurs de l'informatique séparément, un point important est leur présence conjointe dans la science informatique, ce qui a des implications sur la façon d'aborder l'enseignement de ce domaine (Dowek, 2011).

D'une part, la conception d'un algorithme peut être partiellement contraint par le langage de programmation attaché à la machine qui va l'exécuter. Cela empêche dans certains cas de séparer complètement les activités de conception d'algorithme et d'écriture de programme (Arsac, 1987). Nous questionnerons cet aspect dans le cas de la programmation par blocs.

D'autre part, « l'homme rédige un programme en fonction de la signification des instructions (sémantique du langage). La machine l'interprète à partir de sa seule forme (syntaxe). Il faut donc garantir que ce qui est écrit a bien la signification que l'on avait en tête » (Arsac, 1987), d'où la nécessité d'une bonne représentation du dispositif informatique qui est parfois problématique chez les débutants (Rogalski, 1988b). En conséquence, de manière complémentaire à l'analyse de l'apprentissage des notions algorithmiques de base, nous porterons une attention à la prise en main du dispositif informatique utilisé pour notre recherche.

1.1.3 La pensée informatique

Nous avons situé nos études de cas dans le champ de l'initiation à la pensée informatique (Léonard et al., 2019 ; Peter et al., 2019).

Pour Pellet, la pensée informatique est un concept relatif à l'enseignement de l'informatique, dans son aspect initiation et dans son articulation avec les autres disciplines, plus qu'un concept relatif à l'informatique elle-même (Drot-Delange et al., 2019, point de vue de Pellet). En conséquence, nous le considérons comme un concept entrant dans le champ de notre approche didactique. Et nous passons succinctement en revue la manière dont les communautés de recherche discutent de ce concept qui n'est pas encore l'objet d'un consensus (Drot-Delange et al., 2019).

Plusieurs approches de ce que recouvre la pensée informatique

Papert est le premier à utiliser l'expression *pensée informatique* dans son ouvrage *Mindstorms* en 1980, pour désigner la compétence que les enfants acquièrent lorsqu'ils pratiquent la programmation (Papert, 1980). La présence d'une forme de pensée singulière directement liée à l'informatique est aussi discutée en France dès cette période. Cependant environ quatre décennies ont passé et un consensus tarde à se dégager à propos de ce que recouvre la *pensée informatique* (Drot-Delange et al., 2019). La définition de la notion et sa caractérisation font encore l'objet de débats (Grover & Pea, 2018). Plusieurs courants existent, à travers lesquels les questions de spécificité et de caractérisation d'une forme de pensée liée à l'informatique sont posées.

Une première définition, plutôt issue des travaux de recherche française en didactique, s'appuie sur les concepts intégrateurs de la science informatique introduits précédemment. Pour Baron et al., « décrire un objet de manière algorithmique, en utilisant un langage formel à cet effet, gérer des flux d'information et utiliser des machines caractérisent la pensée informatique » (Baron et al., 2015). Berry évoque la construction d'un nouveau *schéma mental* associé à l'informatique et à une *pensée algorithmique* à l'inverse d'une *pensée physique* (Berry, 2017).

Baron et al. distinguent la pensée informatique du terme anglo-saxon *computational thinking* qu'ils traduisent par *pensée computationnelle* et résument en « la capacité à résoudre des problèmes et concevoir des systèmes informatiques » (Baron et al., 2015). Nous distinguons cependant deux courants principaux dans les travaux anglophones. Un premier courant porté actuellement par Denning (2017) et un second apparu à la suite de l'article de Wing en 2006.

En 2006, la parution de l'article de Wing marque un tournant dans l'approche de la pensée informatique. Si la pensée informatique réfère toujours à la pensée de l'informaticien, « *thinking like a computer scientist* » (Wing, 2006), l'accent est mis sur les capacités cognitives convoquées lors de la manipulation des concepts fondamentaux de l'informatique. La pensée informatique comprend alors la résolution de problèmes, la conception de systèmes et la compréhension du comportement humain, en s'appuyant sur les concepts fondamentaux de l'informatique (Wing, 2006).

Les paragraphes suivants fournissent quelques précisions sur ces deux principaux courants relatifs à la pensée computationnelle.

La pensée computationnelle dans le sens restreint de la pensée algorithmique

Denning, dans le sillage des pionniers des années 1960, considère la pensée computationnelle dans un sens que nous qualifions de *restreint*. Dans cette approche, la pensée computationnelle est limitée au domaine de l'informatique. Elle comprend essentiellement la pensée algorithmique, qui consiste à concevoir une série d'instructions afin de solutionner un problème en utilisant une machine. Dans cette approche, la programmation est indissociable de la notion de pensée computationnelle (Denning, 2017). Denning préconise la définition de pensée computationnelle élaborée par Aho : « *Abstractions called computational models are at the heart of computation and computational thinking.[...] computational thinking is the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms.* » [1] (Aho, 2012)

Dans cette approche, un algorithme est une série d'étapes qui contrôle un *modèle computationnel* sans nécessiter de jugement humain (Denning, 2017). Un modèle computationnel est une simulation d'un système complexe en utilisant une approche algorithmique ou automatique, sans que le système soit considéré dans son aspect matériel contrairement à l'approche de Dowek (2011). Pour Denning, c'est une erreur de dissocier la conception de l'algorithme du modèle d'exécution sous-jacent. Si un algorithme comprend bien pour lui une succession d'étapes, il insiste sur le fait qu'il est destiné à contrôler un modèle computationnel, ce qui rend le concept de machine indissociable de celui d'algorithme, sans pour autant que la machine ne soit considérée sous son aspect technologique.

Comme pour les didacticiens français, cette approche est centrée sur le traitement de l'information, mais avec une mise en avant plus prononcée de l'aspect calculatoire (*computation*). Son focus sur la pensée algorithmique est aligné avec notre travail sur le processus d'acquisition des premières notions d'algorithmique. En revanche, l'aspect calculatoire n'est pas central dans notre travail, car comme nous le verrons plus loin, une part de l'aspect calculatoire est pris en charge par le système et caché à l'utilisateur en programmation par blocs.

1. traduction en français : « Les abstractions appelées modèles computationnels sont au cœur du calcul et de la pensée computationnelle. [...] la pensée computationnelle consiste en des processus de pensée impliqués dans la formulation de problèmes afin que leurs solutions puissent être représentées sous forme d'étapes de calcul et d'algorithmes. »

La pensée computationnelle dans un sens élargi à la suite de l'article de Wing

Une autre approche de la pensée computationnelle, que nous qualifions de *large* s'est développée à la suite de la parution de l'article « *Computional thinking* » de Wing en 2006. Selon cette approche, la pensée computationnelle n'est pas réservée aux informaticiens, mais constitue une compétence fondamentale pour tout citoyen au même titre que lire, écrire et compter. Elle introduit une approche de la pensée informatique dans un sens élargi qui comprend la résolution de problèmes, la conception de systèmes et la compréhension du comportement humain, en s'appuyant sur les concepts fondamentaux de l'informatique (Wing, 2006).

Wing précise en 2008 une définition de la pensée computationnelle : « *the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form which can be effectively carried out by an information-processing agent.* » [2] (Wing, 2008). Dans la définition de Wing, le traitement de l'information peut être réalisé indifféremment par une machine ou par un agent humain.

Depuis 2006, une abondante littérature a été publiée à propos de cette approche de la pensée computationnelle, dont nous ne cherchons pas l'exhaustivité dans ce travail. Nous retenons que deux types de définitions sont présentes dans la littérature, définitions conceptuelles ou englobantes et définitions en extension qui tentent de délimiter un périmètre de ce que recouvrent les processus cognitifs du traitement systématique de l'information.

Les définitions conceptuelles ont pour point commun de référer à l'approche des informaticiens pour résoudre des problèmes complexes (Gouws et al., 2013b ; Grover & Pea, 2018), à stipuler que cette approche peut être mobilisée pour résoudre des problèmes complexes hors du champ de l'informatique (Furber, 2012 ; Grover & Pea, 2013).

Selon ces définitions conceptuelles, la pensée computationnelle comprend un ensemble d'habiletés identifiées par rapport au champ de l'informatique et qui peuvent être réinvesties dans d'autres domaines. Identifier ces habiletés et en déduire une définition en extension de la pensée computationnelle est l'objet de débats et de travaux de recherche. Palts et Pedaste ont établi une catégorisation des auteurs en fonction des habiletés considérées comme relevant de la pensée informatique. En prenant l'article de Wing comme point de départ, les auteurs ont identifié six clusters, et les dépendances entre les travaux de différents auteurs (Palts & Pedaste, 2020).

2. traduction en français : « le processus de pensée impliqué dans la formulation des problèmes et de leurs solutions afin que les solutions soient représentées sous une forme qui peut être efficacement exécutée par un agent de traitement de l'information. »

L'un des clusters émane de la définition de Selby et Woollard qui décomposent la pensée informatique en cinq composantes principales (Selby & Woollard, 2013). Cette catégorisation est reprise et détaillée par Csizmadia et al. dans un guide à destination des enseignants des niveaux primaire et secondaire anglais (Csizmadia et al., 2015). Selon ces auteurs, la pensée informatique est un ensemble d'habiletés impliquées dans la résolution de problèmes et la compréhension de phénomènes et de systèmes :

- La *pensée algorithmique*, qui consiste à raisonner en termes d'étapes et de règles afin de construire une solution
- Le processus de *décomposition*, qui consiste à séparer un problème en plusieurs parties qui sont considérées séparément
- Le processus de *généralisation*, qui est associé avec l'identification et l'utilisation de motifs, de similarités et de connexions. Le processus de généralisation permet d'adapter une solution construite pour un problème à d'autres problèmes qui présentent des similarités.
- Le processus d'*abstraction*, qui consiste à rendre un problème ou un phénomène plus compréhensible en repérant les éléments essentiels, en masquant les détails accessoires et en choisissant une représentation adaptée.
- Le processus d'*évaluation*, qui consiste à s'assurer qu'une solution trouvée convient, en termes de validité mais aussi en termes de performances, d'utilisabilité et d'acceptabilité.

Parmi ces composantes, deux sont plus spécifiquement adressées par notre travail de recherche : la pensée algorithmique et la généralisation, dont relève la reconnaissance de motifs. Nous y reviendrons.

En complément, nous retenons l'apport de *The Computer Science Teachers Association* (CSTA) et *The international Society for Technology in Education* (ISTE), qui abordent la pensée computationnelle non plus en termes d'un faisceau d'habiletés mais en termes d'étapes dans le processus cognitif. Ils découpent celui-ci en six étapes (ISTE & CSTA, 2011) :

- Formuler un problème d'une manière que cela permette d'utiliser un ordinateur ou d'autres outils pour aider à le résoudre
- Analyser les données et les organiser logiquement
- Représenter les données par des abstractions telles que des modèles ou des simulations
- Identifier, analyser et implémenter des solutions possibles avec l'objectif de trouver la combinaison la plus efficace en termes d'étapes et de mobilisation de ressources.

- Généraliser et transférer ce procédé de résolution de problème à une plus large variété de problèmes

Controverses et points de vigilance

L'approche *large* de la pensée computationnelle est décrite comme vague et confuse par Denning. Il soulève trois problèmes : l'imprécision de la définition, la difficulté de l'évaluer et l'assertion qu'elle bénéficie à tous. Il regrette aussi l'absence de mention à la notion de modèle computationnel. Il met en garde sur le fait que le point de vue de Wing amène à mettre en retrait le dispositif informatique, la machine qui prend en charge le processus d'exécution (Denning, 2017).

L'un des points de vigilance concerne la difficulté d'évaluer cet ensemble d'habiletés que constitue la pensée computationnelle. En 2015, une étude de González fait état de failles dans la manière de mesurer et d'évaluer la pensée computationnelle (González, 2015). Depuis, plusieurs auteurs ont proposé un cadre opérationnel qui vise à identifier et évaluer des éléments de pensée computationnelle dont voici quelques exemples : évaluation à partir de projets réalisés avec le logiciel Scratch (Moreno-León et al., 2015), évaluation des notions algorithmiques dans le contexte de la programmation par blocs sous forme de QCM (Grover, 2020 ; Zapata-Cáceres et al., 2020). Cependant un consensus n'est pas établi, comme le montre une récente revue de littérature (Poulakis & Politis, 2021).

La place de la programmation dans l'apprentissage de la pensée informatique

Nous constatons aussi une divergence de point de vue quant à la place de la programmation dans l'apprentissage de la pensée informatique, selon l'approche adoptée.

Pellet mentionne que les deux approches anglophones de la pensée informatique ont des perspectives inversées dans la relation de causalité quant à la place de la programmation (Drot-Delange et al., 2019) : dans l'approche restreinte, concevoir des algorithmes et programmer engendrent la pensée informatique, alors que dans l'approche large, les habiletés transversales identifiées comme faisant partie de la pensée informatique aident à la résolution de problème, la programmation étant un problème parmi d'autres.

Dans l'approche restreinte portée par Denning, la pensée informatique ne s'envisage pas sans programmation, car la pratique de la programmation est l'activité qui permet de construire la pensée informatique vue comme le processus cognitif impliqué lors de la formulation de problèmes, de telle manière que

leurs solutions puissent être représentées comme étapes de calcul et algorithmes (Aho, 2012).

En revanche, lorsque l'on considère la pensée informatique dans un sens élargi, alors ce sont les habiletés relevant de la pensée informatique qui rendent l'activité de programmation possible. Selon Wing, la programmation permet d'implémenter la pensée informatique mais celle-ci ne s'y réduit pas (Wing, 2006).

Entre ces deux positions, nous trouvons un continuum concernant l'importance de l'activité de programmation dans l'apprentissage de la pensée informatique. Zhang et Nouri passent en revue 55 études empiriques concernant des élèves de 15 ans et moins, afin de déterminer les compétences relatives à la pensée informatiques qui peuvent être acquises à travers la programmation en Scratch (Zhang & Nouri, 2019). À cette fin, les auteurs utilisent le référentiel défini par Brennan et Resnick (2012), dans lequel les compétences sont divisées en trois sous-ensembles : concepts, pratiques et visées/perspectives. Pour Grover et Pea, la pensée informatique ne se résume pas à la programmation mais la programmation est une activité clé (Grover & Pea, 2013). Pour Fagerlund et al., la programmation est essentielle, bien que non exclusive, car elle permet d'aborder toutes les habiletés cognitives de la pensée informatique dans son sens large (Fagerlund et al., 2021). De ces travaux, nous retenons ce rôle privilégié mais non exclusif de la programmation.

1.1.4 Synthèse sur la didactique de l'informatique

Ce bref aperçu du champ de la didactique de l'informatique montre le large périmètre des concepts convoqués et analysés dans les publications scientifiques.

Cependant, le récent rapport de l'IGÉSR, qui date de 2022, fait état de difficultés didactiques qui sont encore peu étudiées du point de vue de la recherche. Le besoin d'approche didactique pour analyser et accompagner la réintroduction de l'informatique dans les curricula scolaires est explicitement exprimé dans cet état des lieux de la pratique de l'informatique aux cycles 3 et 4 : « Il existe des ressources d'accompagnement au niveau national et académique conséquentes, mais peu abordent des préoccupations didactiques (qui ne sont par ailleurs pas explicites dans les programmes). Cependant on constate un manque de formation des enseignants aux concepts de l'informatique, notamment à la démarche algorithmique » (Gaubert-Macon et al., 2022).

Notre recherche vise à contribuer à répondre à ce besoin d'études didactiques. Les éléments de didactique de l'informatique introduits précédemment constituent les fondements sur lesquels nous bâtissons notre étude sur la confrontation avec les premières notions d'algorithmique en programmation par blocs. La programmation s'inscrit dans le champ des activités relevant de la pensée

informatique, quelle que soit l'approche adoptée pour ce concept. Elle en est une activité clé. Parmi les composantes de la pensée informatique, nous nous intéressons particulièrement à la pensée algorithmique et à la reconnaissance de motif. Nous nous appuyons aussi sur les concepts piliers de la science informatique et sur les acquis de la psychologie de la programmation, que nous visons de contextualiser à la programmation par blocs. Nous compléterons ce cadre théorique disciplinaire par des cadres d'analyse plus généraux dans la partie II, afin de nous outiller conceptuellement pour analyser la construction des premières notions d'algorithmique et la prise en main d'un environnement de programmation par blocs.

1.2 L'informatique à l'école obligatoire en France

Nous avons délimité le périmètre de notre recherche à l'initiation à la programmation informatique vers des élèves de 7 à 15 ans, qui sont donc scolarisés à l'école élémentaire ou au collège. En conséquence, nous faisons le point sur la présence de l'informatique au niveau de l'école obligatoire en France suite à sa réintroduction en 2016. Nous nous appuyons sur les textes des programmes scolaires, sur le rapport de l'IGÉSR (Gaubert-Macon et al., 2022) ainsi que sur quelques études rapportant des expérimentations menées dans des classes.

1.2.1 Les contenus de programmation informatique dans les programmes scolaires à partir de 2016

À l'école élémentaire et au collège, l'informatique ne constitue pas une discipline scolaire, c'est à dire « une construction sociale organisant un ensemble de contenus, de dispositifs, de pratiques, d'outils...articulés à des finalités éducatives, en vue de leur enseignement et de leur apprentissage » (Reuter et al., 2013). Ce choix d'organisation curriculaire implique que ce qui relève de l'informatique est disséminé au sein des matières constituées. Cela nous amène à considérer une granularité plus fine pour l'analyse des documents prescriptifs, en nous appuyant sur le concept didactique de contenu. Un contenu « renvoie à des choses aussi diverses que les savoirs, les savoir-faire ou les compétences qui sont les objets d'enseignement et/ou d'apprentissages les plus immédiatement identifiables dans un système didactique, mais aussi des valeurs, des pratiques des « rapports à », voire des comportements ou des attitudes » (Reuter et al., 2013). Pour Fluckiger, il est nécessaire de questionner à quoi réfèrent les contenus identifiés (Fluckiger, 2019). Pour ce travail de recherche, nous nous situons dans une approche qui prend pour référence la discipline scientifique informatique

adossée à une section universitaire autonome depuis les années 1970 [3]. Nous relevons les contenus dans les documents prescriptifs en conséquence, en nous limitant aux aspects algorithmique et programmation qui sont principalement l'objet de notre étude.

Dans le *Socle commun de connaissances, de compétences et de culture* [4], l'informatique apparaît dans le domaine 1, intitulé « Langages pour penser et communiquer », avec l'item « Comprendre, s'exprimer en utilisant les langages informatiques ». Plus précisément, « l'élève sait que des langages informatiques sont utilisés pour programmer des outils numériques et réaliser des traitements automatiques de données. Il connaît les principes de base de l'algorithmique et de la conception des programmes informatiques. Il les met en œuvre pour créer des applications simples. »

Dans les programmes de cycle 2 et 3 en vigueur depuis la rentrée 2020 [5], la notion de programme est explicitement convoquée dans la partie « Espace et géométrie » du domaine mathématiques. Dans la section « (Se) repérer et (se) déplacer en utilisant des repères et des représentations » figure l'item « Programmer les déplacements d'un robot ou ceux d'un personnage sur un écran en utilisant un logiciel de programmation ». Aucune précision n'est apportée sur les notions algorithmiques à aborder. La programmation est systématiquement associée à la structuration de l'espace, à travers la programmation de déplacements ou de construction de figures.

Au niveau du cycle 4, les contenus d'informatique sont présents au sein de deux matières : mathématiques et technologie [6]. Ils constituent une section dédiée dans l'un et l'autre programmes. Dans le programme de mathématiques, le thème E est intitulé « Algorithmique et programmation ». Les connaissances visées sont les notions d'algorithme et de programme, de variable informatique, de séquence d'instructions, de boucles, d'instructions conditionnelles, de déclenchement d'une action par un évènement. La compétence associée est « Écrire, mettre au point (tester, corriger) et exécuter un programme en réponse à un problème donné ». La notion de fonction n'apparaît pas dans la liste des connaissances attendues. Néanmoins, elle est mentionnée dans l'introduction : « En créant un programme, ils [les élèves] développent des méthodes de programmation, revisitent les notions de variables et de fonctions sous une forme différente, et s'entraînent au raisonnement ». Le Bulletin officiel n°22 du 29 mai 2019 établit une progression selon trois niveaux dans l'apprentissage de ces notions.

3. 1972 : sous-section « informatique » au comité consultatif des universités ; 1976 : section « informatique, automatique, analyse des systèmes, traitement du signal » au Comité national du CNRS

4. Décret n°2015-372 du 31 mars 2015

5. BOEN n° 31 du 30 juillet 2020

6. BOEN n° 31 du 30 juillet 2020

La notion de boucle, qui nous intéresse plus particulièrement, apparaît dès le niveau 1. Sa maîtrise est visée en fin de 5ème, à travers la programmation d'un déplacement ou d'une construction géométrique. Depuis 2017, un exercice sur papier du Brevet National des Collèges dans la discipline mathématiques est systématiquement basé sur des blocs en langage Scratch. Dans le programme de technologie, la section dédiée à l'informatique s'intitule « Informatique et programmation ». L'attendu de fin de cycle pour la programmation est « Écrire, mettre au point et exécuter un programme ». Parmi les compétences visées, nous relevons, sous le titre « Pratiquer des langages », « Appliquer les principes élémentaires de l'algorithmique et du codage à la résolution d'un problème simple ». Nous retrouvons les mêmes notions algorithmiques que dans le programme de mathématiques, avec en plus des notions en lien étroit avec la robotique (capteur, actionneur), les réseaux et systèmes (protocole, routage, système embarqué). Un nouveau programme de technologie est paru en février 2024[7]. Pour l'algorithmique et la proprammation, la compétence de fin de cycle 4 devient « concevoir, écrire, tester, mettre au point un programme ». Par rapport au programme précédent, de plus amples précisions sur le contenu sont données. Nous relevons aussi une mention à la pensée informatique et aux concepts piliers de la science informatique.

Depuis 2019, l'évaluation des compétences de programmation en fin de collège est également incluse dans la certification plus large des compétences numériques par PIX, sur la base du Cadre de Référence des Compétences Numériques (CRCN)[8].

Selon le rapport de Gaubert-Macon et al. (2022), la répartition entre les matières mathématiques et technologie au collège génère des progressions pédagogiques en silo, des approches dont l'articulation n'est pas concertée. L'épreuve du brevet ne rend pas compte de l'aspect pratique de l'informatique, dans son rapport à la machine, ce qui n'incite pas les enseignants à la mettre en œuvre. L'évaluation porte plus sur l'interprétation d'un programme donné, que sur l'écriture, la mise au point et l'exécution d'un programme telles que visé par le texte du programme.

1.2.2 Les pratiques de classe répertoriées

Une étude internationale de Ching et al. datant de 2018 répertorie les technologies disponibles pour l'enseignement de la pensée informatique. Les auteurs ont identifié les robots pédagogiques, les kits robotiques, les jeux de plateau, les applications en réalité augmentée, les applications ou sites Web pour la

7. BOEN n°9 du 29 février 2024
8. Décret n° 2019-919 du 30 août 2019

programmation (par blocs ou textuelle), les outils de développement orientés animation ou jeu. Ces différentes catégories se distinguent selon qu'elles utilisent une manipulation physique ou une interaction sur l'écran et selon que la rétroaction est concrète (par exemple déplacement d'un robot tangible) ou abstraite (c'est-à-dire un retour sur l'écran). Les résultats de cette étude montrent que les concepts abordés avec ces technologies vont des séquences et répétitions à des concepts plus avancés. Elles peuvent impliquer des activités créatives ou de résolution de problèmes (Ching et al., 2018).

Pour la situation en France, nous nous appuyons à nouveau sur le rapport de l'IGÉSR de Gaubert-Macon et al. (2022). Par rapport à l'étude de Ching et al., nous élargissons à l'ensemble des ressources disponibles et pas seulement à celles relevant de la technologie, avec une attention particulière aux ressources en lien avec l'algorithmique et la programmation. Nous complétons en citant quelques études qui s'appuient sur des expérimentations menées dans les classes.

Les pratiques répertoriées dans le rapport de Gaubert-Macon et al. (2022) sont l'utilisation de cartes programmables, de logiciels de programmation par blocs, de logigrammes en technologie, la robotique pédagogique, l'informatique débranchée, la participation à des concours. Dans la suite, nous apportons quelques précisions à propos de certaines d'entre elles. Nous privilégions celles qui ont un lien avec le déplacement de robots sur une grille, qui est à la base de nos expérimentations. Nous relevons notamment des activités qui mobilisent d'autres modalités de programmation de robot. Pour les environnements de programmation par blocs, qui sont au cœur de ce travail, le chapitre 2 leur est consacré.

Une approche répertoriée pour aborder l'informatique dans l'esprit des programmes scolaires, en mettant l'accent sur le repérage et les déplacements dans l'espace est la robotique pédagogique. Des robots tangibles sont commandés soit directement par pression sur des touches, soit via une interface de programmation tangible, soit en utilisant un langage de programmation par blocs, principalement à partir d'une tablette. Parmi les robots tangibles couramment utilisés, le robot *Blue Bot* et ses équivalents est destiné aux plus jeunes, à partir de 4 ans environ, et jusque 7-8 ans. Avec des élèves plus âgés, ce sont des robots tels que *Thymio*, *Ozobot* ou *mBot* qui sont utilisés. À la différence des premiers, ces robots sont pourvus de capteurs. Ils peuvent être programmés à partir d'interface de programmation par blocs. Plusieurs travaux de recherche ont étudié l'utilisation de ces robots en classe, dont ceux de Spach (2017) et Bellegarde et al. (2019). En effet, un intérêt repéré dès la fin des années 1960 par Papert (1981) et son robot tortue est, comme l'expliquent Baron et Bruillard, que ces robots permettent une « transition entre un *faire faire*, à la fois abstrait et difficile surtout pour des enfants, et un *faire*, puisque l'utilisateur (le programmeur)

peut se mettre à la place du robot et exécuter lui-même ce qu'il prévoit de lui demander. Le programme est alors une traduction des actions que lui-même (ou la tortue) effectue » (Baron & Bruillard, 2001).

Une autre approche, appelée *informatique débranchée*, consiste à aborder l'informatique sans mobiliser de matériel comportant de l'électronique (Bell et al., 2002). Le focus est alors mis sur l'aspect conceptuel de l'informatique. Dans les activités proposées, c'est souvent un humain qui joue le rôle de la machine. Cette approche, encore émergente en France mais qui tend à se propager, est portée par quelques personnes très actives dans le champ de la médiation scientifique, comme Marie Duflot-Kremer [9]. Une des activités débranchées fréquemment mise en place est le « jeu du robot », lors de laquelle le rôle du robot qui exécute les instructions données est joué par un humain (Romero, Duflot-Kremer et al., 2018).

Parmi les pratiques répertoriées dans le rapport de Gaubert-Macon et al. (2022) figurent aussi les concours d'informatique et de robotique en contexte scolaire. Le rapport mentionne un effet mobilisateur pour les élèves et les enseignants, en soulignant toutefois un manque d'évaluation sur les apports de ces concours du point de vue des connaissances et capacités.

Parmi ces concours, deux sont organisés annuellement par l'association France-ioi et ont une très large audience. Le premier est le concours Castor qui vise une initiation à l'informatique et aux sciences du numérique. Près de 700 000 élèves du CM1 à la terminale ont participé en 2022. Le second est le concours Algoréa. Il est centré sur l'apprentissage de l'algorithmique et de la programmation, et comprend plusieurs niveaux de difficulté et plusieurs phases qui se succèdent chaque année entre janvier et juillet. Plus de 250 000 élèves ont participé au premier tour de l'édition 2023. Nous détaillerons le fonctionnement de ce concours dans le chapitre 7, car il est le support du travail expérimental de cette recherche.

Cependant la mise en oeuvre de ces différentes pratiques se heurtent à de nombreuses difficultés. Le rapport de Gaubert-Macon et al. (2022) liste notamment l'absence de quotité horaire dédiée qui entraîne un temps de pratique réduit pour les élèves, la difficulté d'accès au matériel nécessaire et la compatibilité de ce matériel avec les activités visées, le peu d'appétence voire l'appréhension des enseignants face à ces activités (surtout pour le premier degré), une formation des enseignants centrée sur la prise en main d'outils plutôt que sur une approche didactique des concepts et méthodes en jeu.

9. https://pixees.fr/dans-la-famille-activites-debranchees-je-demande-les-tutos-videos-de-marie-duflot/

1.2.3 Synthèse

La revue des programmes scolaires ainsi que des pratiques mises en œuvre dans les classes confirment que la tâche de programmation de déplacements d'un robot virtuel sur une grille que nous avons choisi d'investiguer est représentative des tâches indiquées en contexte scolaire. Aucun logiciel particulier n'est mentionné dans les programmes, mais les faits que seuls des exemples avec le logiciel Scratch soient donnés dans les documents d'accompagnement, et que l'exercice du brevet utilise systématiquement des blocs Scratch orientent très fortement vers l'utilisation de ce logiciel.

Nous notons de plus que jusqu'en 2024, aucune mention à la pensée informatique n'est faite dans les documents prescriptifs, ce que déplore le rapport de Gaubert-Macon et al. (2022).

1.3 Les notions de base en algorithmique

1.3.1 La pensée algorithmique

Comme nous l'avons mentionné précédemment, la pensée algorithmique est une composante de la pensée informatique. Elle se confond même avec celle-ci dans l'approche restreinte de Denning (2017).

Nous retrouvons le concept de pensée algorithmique à la fois dans le champ des mathématiques et dans celui de l'informatique (Modeste, 2012). Nous nous questionnons sur sa spécificité dans le champ de l'informatique.

D'après Knuth (1985), repris par Modeste (2012), l'algorithmique est l'étude de ce qui est automatisable. Ce qui caractérise la pensée algorithmique en informatique est la recherche de l'effectivité et de l'efficacité à travers l'étude de la complexité. Les aspects en sont la mise en œuvre d'algorithmes notamment via leur implémentation dans un langage de programmation et leur exécution sur une machine, la description et la preuve de ces algorithmes.

La pensée algorithmique questionne l'efficacité des algorithmes conçus, d'une part en nombre d'opérations à travers l'étude de la complexité en temps, et d'autre part en nombre d'éléments d'information à stocker à travers l'étude de la complexité en mémoire.

Nous retenons que la pensée algorithmique dans le champ de l'informatique comporte un aspect expérimental à travers la possibilité d'exécuter le programme sur une machine. La pensée algorithmique est liée au *faire faire*, elle cherche à produire en le rendant systématique et à optimiser un processus de traitement. Cette spécificité de posture cognitive est souligné par Hoc (1979), repris par Rogalski (2015). Dans le cadre de notre recherche, c'est exclusivement l'aspect

du *faire faire* qui nous intéresse, et nous n'abordons pas l'étude de la complexité étant donnés la simplicité des algorithmes impliqués et le jeune âge du public de l'étude.

1.3.2 Les notions algorithmiques de base en programmation impérative

De nombreux langages de programmation existent, regroupés en familles, qui définissent des paradigmes de programmation, parmi lesquels la programmation impérative (que l'on nomme aussi séquentielle ou procédurale), la programmation fonctionnelle, la programmation évènementielle, la programmation orientée objet, la programmation parallèle.

Pour l'initiation, le paradigme de programmation impérative est largement mobilisé (Lagrange & Rogalski, 2017). La quasi-totalité des études de la décennie 1980-1990 ont été menées dans ce paradigme (Rogalski, 2015). Comme c'est celui dans lequel se situe notre étude, nous nous concentrons sur celui-ci.

« Dans un langage impératif, le traitement s'exprime comme une succession d'affectations de variables qui modifient l'état du dispositif » (Lagrange & Rogalski, 2017). Le concept de variable est donc central dans ce paradigme de programmation. Par défaut, les instructions qui commandent le traitement sont organisées sous forme séquentielle. Les structures de contrôle (boucles et structures conditionnelles) permettent d'agir sur la séquentialité du traitement.

Quelles sont les notions algorithmiques répertoriées comme de base pour l'initiation dans le paradigme de la programmation impérative ?

Nous répertorions ces concepts fondamentaux à partir des référentiels disponibles pour le domaine : le Cadre de Référence des Compétences Numériques (CRCN) [10] qui est une contextualisation pour le système éducatif français du DigCompEdu [11] européen, le référentiel PIAF (développer la Pensée Informatique et Algorithmique dans l'enseignement Fondamental) (Busana et al., 2019), le référentiel de types de tâches de Jolivet et al. (2023).

Un consensus entre ces différents référentiels porte sur les notions d'instruction, de séquence d'instructions, de boucles, d'instructions conditionnelles, de variable, et de décomposition d'un programme en sous-programmes. Ces notions sont les éléments de base des langages impératifs, et ont déjà été identifiées comme stables dans le temps par Rogalski et Vergnaud (1987). Nous caractérisons brièvement les notions qui ne l'ont pas été précédemment.

Les boucles permettent de faire exécuter plusieurs fois le même traitement

10. Décret n° 2019-919 du 30 août 2019
11. Digital Competence Framework for Educators publié par la Commission européenne en 2017

à une machine. Une boucle est caractérisée par le traitement à faire répéter et par une condition d'arrêt. Deux types de boucles existent suivant l'expression de cette condition d'arrêt. Si le nombre d'itérations est connu à l'avance, la condition d'arrêt peut s'exprimer sous la forme d'un compteur. Ce type de boucle est appelé boucle à compteur ou boucle bornée. En revanche, si le nombre d'itérations est inconnu avant de commencer le traitement, la condition d'arrêt s'exprime sous la forme d'une expression booléenne qui est évaluée lors de chaque itération. Dans ce dernier cas, les boucles sont appelées boucles conditionnelles ou boucles non bornées, et se subdivisent en deux (boucles *tant que* et boucles *jusqu'à*).

L'instruction conditionnelle, appelée parfois alternative, permet de différencier un traitement suivant qu'une condition est vraie ou fausse. Une instruction conditionnelle comprend deux parties : une condition et le traitement à effectuer en fonction de la valeur, vraie ou fausse, de la condition. L'instruction conditionnelle peut être complète (si.. alors.. sinon) ou incomplète (la branche sinon, vide, est absente).

Une variable au sens informatique du terme est une adresse en mémoire, à laquelle on a donné un nom, et qui permet de stocker une valeur, une donnée. Cette valeur évolue généralement au fil de l'exécution du programme, grâce à l'instruction d'affectation.

Enfin, la notion de fonction au sens informatique du terme permet de décomposer un programme en sous-programmes. La modularité des programmes ainsi obtenue concourt à leur lisibilité et à la réutilisation de certaines parties au sein du même programme ou éventuellement d'autres programmes.

Parmi ces notions, nous nous concentrons sur les premiers apprentissages de la boucle bornée.

1.4 Focus sur l'apprentissage de la boucle

Dans les référentiels mentionnés précédemment, nous trouvons peu d'éléments spécifiques sur la notion de boucle. Dans le document d'accompagnement du CRCN, la boucle n'est pas explicitement mentionnée, contrairement aux autres notions de base. Elle est implicite dans la formulation « Modifier un algorithme simple en faisant évoluer ses éléments de programmation ». Nous supposons que la boucle est un des éléments de programmation possibles. Dans le référentiel PIAF, la notion de boucle est répertoriée comme notion de base pour la compétence 3, intitulée « Contrôler une séquence d'actions ». Les deux types de boucle sont distingués, la boucle bornée en « 3.1 Répéter une séquence d'actions un nombre donné de fois » et la boucle conditionnelle en « 3.2 Répéter une séquence d'actions jusqu'à ce qu'un objectif soit atteint ».

Nous recherchons des éléments plus précis dans les travaux de recherche

du courant de la psychologie de la programmation des années 1980, dans les travaux francophones récents, ainsi que dans quelques travaux de l'abondante littérature anglophone des années 2000 relatives à la programmation par blocs.

1.4.1 La boucle dans les travaux du courant de la psychologie de la programmation des années 1980

La plupart des travaux de cette période concernent la programmation au lycée, avec des langages textuels de type impératif. Quelques travaux sont relatifs aux expérimentations en langage LOGO à l'école élémentaire et au collège. Nous extrayons de ces travaux quelques points qui concernent la notion de boucle.

Au niveau du lycée, Samurçay et Rouchier mènent une expérimentation qui les amènent à collecter, puis à analyser des modèles spontanés de la structuration d'un programme à l'aide d'une boucle (Samurçay & Rouchier, 1985). Rogalski et Samurçay décomposent la construction d'une boucle en tâches cognitives plus élémentaires :

> - la planification répétitive du traitement ; c'est à dire l'identification, l'élaboration et l'expression de l'invariant dans les actions à répéter (l'invariant de boucle) ;
> - l'identification du statut fonctionnel du contrôle d'arrêt : Sur quelle variable porte-t-il ? Pour quelle valeur ? A quel moment intervient-il ? En relation avec quel aspect du problème à résoudre ?
> - la détermination de l'état initial dans lequel se trouvent ou doivent se trouver les variables qui sont transformées dans l'invariant de boucle.
>
> (Rogalski & Samurçay, 1986)

Les auteurs établissent un ordre dans lequel sont réalisées ces tâches cognitives :

> L'ordre dans lequel les opérations apparaissent dans la planification d'une boucle est le suivant : la description de l'action, l'indication de répétition et le contrôle ; le compteur lorsqu'il est explicité, est placé après la description de l'action. (Rogalski & Samurçay, 1986)

Au niveau de l'école élémentaire et du collège, Rouchier étudie la programmation d'une boucle dans le cadre d'expérimentations avec le langage LOGO introduit par Papert. Il constate que les enfants de CM1-CM2 éprouvent des difficultés à appréhender la structure de répétition, notamment à cause d'une

différence d'acception du terme *répéter* entre le langage courant et le langage de programmation. Dans le langage courant, la première exécution n'est pas comptée comme itération, alors qu'elle l'est dans le langage de programmation (Rouchier, 1990).

Dans les langages textuels utilisés par Rogalski et Samurçay (Pascal, LSE, Basic), la programmation d'une boucle implique explicitement le concept de variable, alors que ce concept reste implicite dans la structure REPETE(*nombreFois*) du langage LOGO. Dans leur conclusion, les auteurs invitent à confirmer les résultats obtenus en investiguant d'autres classes d'âge, langages de programmation et systèmes informatiques (Rogalski & Samurçay, 1986). Pour notre part, nous examinerons dans quelle mesure ces résultats restent valides dans le cas des langages de programmation par blocs, manipulés par un public plus jeune. Les résultats mentionnés précédemment constitueront des points de repère auxquels nous confronterons les résultats de nos propres expérimentations.

1.4.2 La boucle dans des travaux francophones récents

La boucle est l'une notions sur lesquelles portent les types de tâches identifiés par Jolivet et al. (2023). Les auteurs fournissent un diagramme (repris en figure 1.1), qui montre comment se décompose la tâche de programmation d'une boucle, en distinguant boucle bornée et non bornée. Il est à noter que cette décomposition a été établie en prenant le langage textuel Python comme référence. Au cours de notre recherche, nous questionnerons ce qui reste valide dans le cas de la programmation par blocs.

Declercq et Tort ont réalisé une analyse a priori de l'activité d'élèves de fin d'école élémentaire confrontés à différentes situations de programmation d'une boucle avec un langage par blocs : programmation du déplacement d'un personnage sur une grille, répétition d'un motif élémentaire pour dessiner une frise, défi du concours Castor centré sur l'identification d'un motif à répéter. L'analyse permet d'identifier quatre comportements type : utilisateur de télécommande (manipulation directe), programmeur pas à pas, programmeur de télécommande (en utilisant la programmation événementielle dans le logiciel Scratch), programmeur qui est le comportement visé (Declercq & Tort, 2018). À l'issue de l'analyse, l'environnement de programmation Pixel'Art est introduit. Par la suite, Declercq et Zeyringer étudient l'activité d'élèves de 9-11 ans dans cet environnement, lors de la programmation de séquences et de boucles. Pour ce faire, ils enregistrent les traces horodatées d'interaction avec le système et construisent des indicateurs afin de caractériser l'activité de l'élève par rapport à une tâche donnée (Declercq & Zeyringer, 2018). Nous retenons ces éléments méthodologiques, qui sont similaires à ceux déjà mobilisés pour nos études exploratoires en amont de la présente recherche.

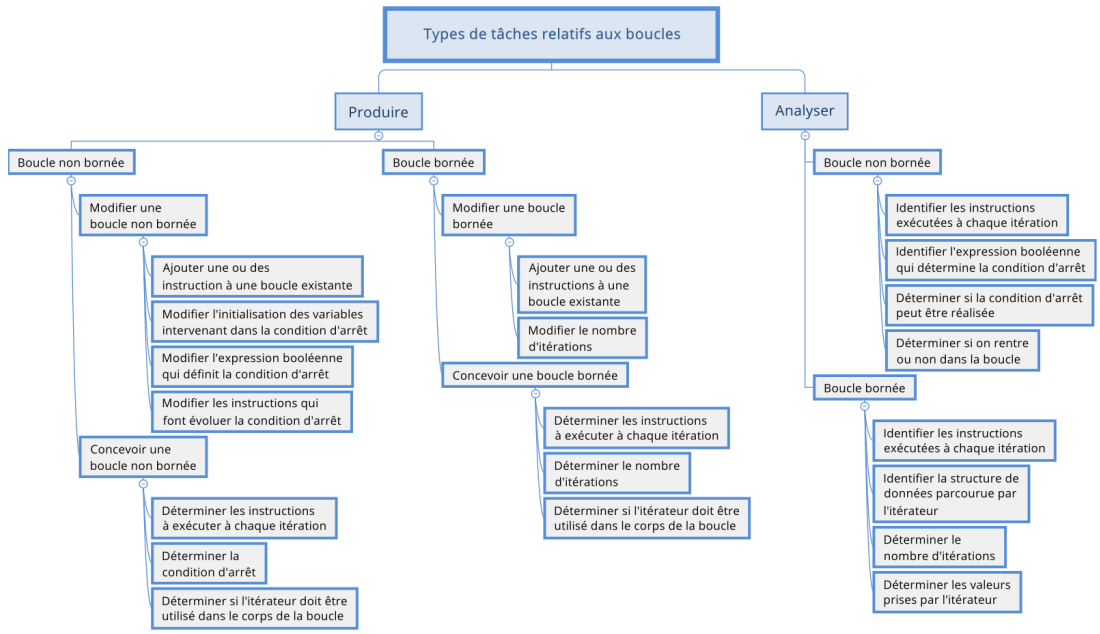


FIGURE 1.1 – Types de tâches relatives aux boucles, diagramme issu de l'article de JOLIVET et al. (2023)

1.4.3 La boucle dans les travaux anglo-saxons relatifs à la programmation par blocs à partir des années 2000

À partir des années 2000, les travaux anglophones sur l'apprentissage de la boucle lors de l'initiation à la programmation sont beaucoup plus nombreux que les travaux francophones. Nous limitons nos investigations aux travaux sur la programmation par blocs, qui concernent des élèves de 15 ans et moins (K-9 dans le système scolaire anglo-saxon). En 2019, ZHANG et NOURI relèvent, en se référant au cadre de BRENNAN et RESNICK (2012), 28 études qui traitent au moins en partie de l'apprentissage de la boucle en Scratch (ZHANG & NOURI, 2019). Nous retenons ceux qui sont le plus proches de notre recherche, que nous complétons par des travaux trouvés par ailleurs, notamment des publications plus récentes.

Une partie de ces travaux vise à fixer un cadre pour l'apprentissage des notions de base en algorithmique et programmation, dont la boucle. BRENNAN et RESNICK fixent un cadre pour l'apprentissage de la pensée informatique à travers l'activité de programmation (BRENNAN & RESNICK, 2012). La boucle fait partie des sept concepts visés. Un exemple est donné en langage Scratch, avec l'objectif d'exprimer une séquence d'instructions plus succinctement, mais sans plus

de détails. Rich et al. définissent des trajectoires d'apprentissage en se basant sur une étude de la littérature pour les élèves de 13 ans et moins (Rich et al., 2017). Une trajectoire d'apprentissage est constituée d'un ensemble d'objectifs d'apprentissage relatifs à un concept. Ces objectifs sont placés sur un chemin d'apprentissage qui est divisé en trois niveaux de difficultés. L'une des trajectoires d'apprentissage concerne le concept de répétition, dont nous proposons une version traduite en français sur la figure 1.2. Les objectifs d'apprentissage sur fond gris sont prévus pour être visés en utilisant des activités débranchées alors que ceux sur fond blanc nécessitent un environnement de programmation. Les flèches noires indiquent qu'un objectif est un pré-requis pour celui qui est pointé, alors que les flèches grises indiquent seulement une relation de facilitation.

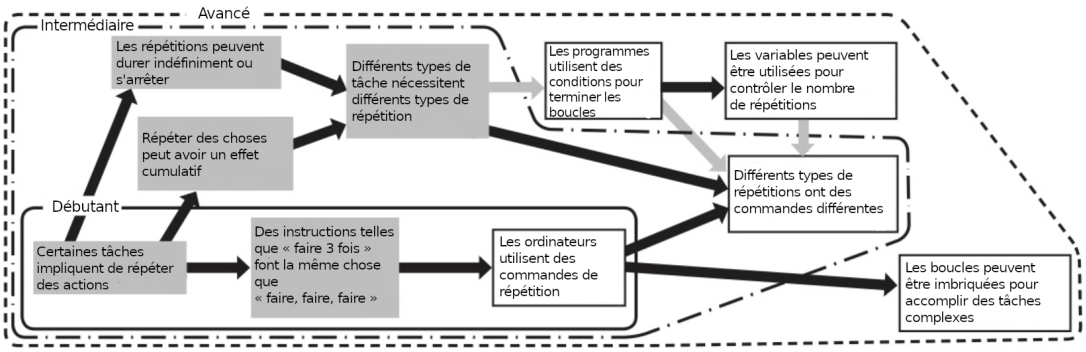


Figure 1.2 – Trajectoire d'apprentissage pour le concept de répétition, issue de l'article de Rich et al. (2017) (version traduite)

Plusieurs auteurs rapportent que la boucle est une notion particulièrement difficile à appréhender. Pour Fraser, les blocs de boucle figurent parmi les plus difficiles à utiliser pour les programmeurs novices (Fraser, 2015). Grover et Basu ont conçu des exercices de programmation par blocs pour examiner les idées fausses de collégiens. Ils ont constaté que même après avoir suivi un cours d'introduction à la programmation, les collégiens ne comprennent pas bien les concepts de base de la programmation, notamment les boucles et les variables (Grover & Basu, 2017).

L'étude des difficultés et idées fausses (*misconceptions* en anglais) fait l'objet de plusieurs travaux. Suite à la conception et à la mise en œuvre d'un test d'évaluation sous forme de QCM, Grover et Basu relèvent des difficultés à propos du fonctionnement de la boucle. Par exemple, lorsqu'il y a plusieurs instructions dans une boucle, au lieu d'exécuter la séquence d'instructions dans la boucle puis de répéter la séquence entière, certains élèves répètent chaque instruction séparément avant de répéter la ou les instructions suivantes. Ils notent aussi un palier de difficulté lorsqu'une variable est impliquée dans

le corps de la boucle (Grover & Basu, 2017). Vaníček et al. ont mené une expérimentation auprès d'élèves âgés de 12 à 13 ans sans expérience préalable en programmation, afin d'examiner comment ils apprennent à utiliser des boucles bornées. En analysant les programmes conçus par les élèves, ils identifient quatre idées fausses relatives aux premières confrontations avec la notion de boucle : ne pas insérer de blocs dans le corps de la boucle, y insérer systématiquement un seul bloc, laisser le nombre de répétitions à la valeur par défaut (Vaníček et al., 2023). La quatrième idée fausse rejoint celle déjà relevée par Grover et Basu, c'est-à-dire concevoir une séquence de boucles au lieu d'une boucle avec plusieurs instructions. Liu et al. ont concentré leurs investigations sur la phase de débogage des programmes par des élèves de 11 à 13 ans. Ils montrent que les programmes comprenant une boucle figurent parmi ceux qui sont difficiles à déboguer, en particulier lorsqu'une variable doit être incrémentée dans le corps de la boucle (Liu et al., 2017).

1.5 Synthèse : positionnement de notre travail de recherche

Au terme de ce chapitre, nous en synthétisons les principaux éléments qui permettent de positionner notre recherche dans le champ de la didactique de l'informatique, en considérant les niveaux scolaires de l'école élémentaire et du collège.

Bien qu'il ne fasse toujours pas l'objet d'un consensus (Drot-Delange et al., 2019) et qu'il tarde à s'imposer dans l'institution scolaire française (Gaubert-Macon et al., 2022), le concept de pensée informatique est l'objet de milliers de publications scientifiques, notamment anglophones, et constitue actuellement le cadre conceptuel de l'initiation à l'informatique au niveau international.

Notre travail s'inscrit dans l'approche de Dowek et Berry qui associent la pensée informatique au traitement automatique de l'information et donc aux concepts de la science informatique : information, langage, algorithme et machine. Ces quatre concepts guideront notre travail théorique. Néanmoins, les approches anglo-saxonnes nous apportent un éclairage complémentaire, sans qu'il nous soit nécessaire de trancher sur l'adoption de l'un des deux courants. D'une part, notre travail est centré sur la pensée algorithmique, composante de la pensée informatique commune aux différentes approches bien que ne revêtant pas la même importance. D'autre part, l'activité de programmation qui constitue le support de notre étude, est considérée comme une activité privilégiée pour l'initiation à la pensée informatique, bien que n'étant pas la seule possible dans l'approche large de ce concept. De plus, contrairement au concept de

pensée informatique lui-même, l'activité de programmation est mentionnée explicitement dans les programmes scolaires français.

Dans les textes des programmes des cycles 3 et 4, la tâche de conception d'un programme en utilisant un logiciel de programmation par blocs apparaît comme centrale, la programmation de déplacements d'un robot virtuel sur une grille en étant une déclinaison possible. En conséquence, cibler ce type de tâche pour nos investigations sur les premières confrontations avec un environnement de programmation par blocs nous semble judicieux. En effet, nous sommes en mesure de nous appuyer sur le concours Algoréa, dont tous les exercices consistent à concevoir un programme pour faire remplir une mission à un robot virtuel. Ce concours est déployé à large échelle, il est cité dans le rapport de l'IGÉSR comme dispositif apprécié (Gaubert-Macon et al., 2022), et l'auteur de cette recherche contribue à l'élaboration des problèmes. Parmi ceux-ci, nous privilégions ceux pour lesquels la notion de boucle bornée est impliquée. La confrontation avec cette notion, que nous avons commencé à explorer lors des expérimentations en amont de cette recherche, est décrite dans la littérature comme difficile à appréhender et l'objet d'idées fausses. Nous visons d'investiguer et de caractériser ces difficultés en nous appuyant sur le concept de motif que nous explorons dans le chapitre 3.

Nous nous situons dans le prolongement des travaux sur la psychologie de la programmation développés dans les années 1980, en ciblant l'aspect conceptuel lors de l'introduction des premières notions en algorithmique. Ces travaux ont été menés dans un contexte de programmation textuelle. Une mise en perspective avec le contexte actuel a été réalisée pour le niveau du lycée par Lagrange et Rogalski en 2017. Les auteurs ont conclu que, malgré l'évolution des langages et de la technologie, les résultats établis dans les années 1980 restent valides dans le contexte actuel et constituent des acquis sur lesquels s'appuyer pour les recherches à venir en didactique de l'informatique. À notre connaissance, il n'y a pas encore eu de mise en perspective pour les niveaux de l'école élémentaire et du collège. Nous visons une telle démarche dans un contexte de programmation par blocs.

À cette fin, nous investiguons les caractéristiques de la programmation par blocs dans le chapitre 2 et nous nous plaçons dans le cadre d'analyse du courant constructiviste, notamment de la théorie des champs conceptuels (Vergnaud, 1991), qui a été largement mobilisé dans les travaux relevant de la psychologie de la programmation (chapitre 4).

Chapitre 2

La programmation par blocs

Sommaire du présent chapitre

Dans le présent chapitre, nous précisons la forme que prend l'activité de programmation dans le contexte de l'école obligatoire en France, dont les élèves constituent le public cible de cette étude. Les programmes scolaires préconisent des environnements de programmation par blocs, tels que l'environnement Scratch (Figure 2.1).

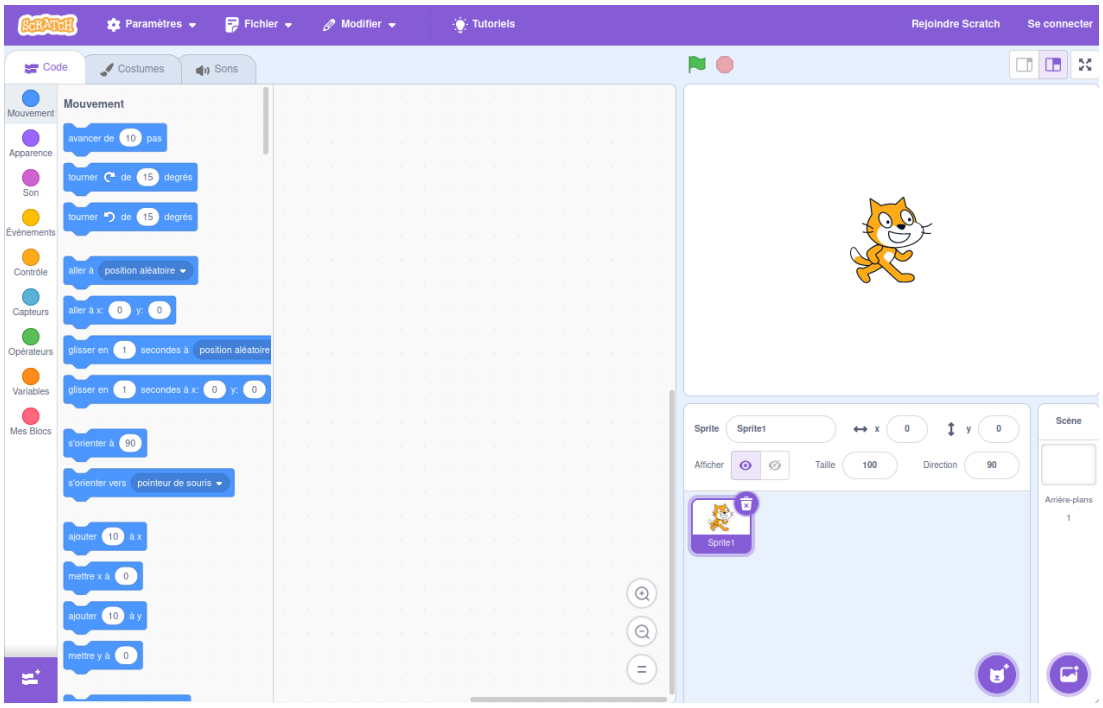


FIGURE 2.1 – Exemple d'environnement de programmation par blocs : le logiciel Scratch

Après avoir identifié les avantages des langages de programmation par blocs et décrit les caractéristiques générales de ces environnements, nous détaillons quelques exemples largement utilisés, puis nous nous attardons sur la modalité des puzzles implémentés dans de tels environnements, que nous avons mobilisée dans nos études de cas. Nous expliquons pourquoi nous reprenons cette modalité pour la présente recherche.

2.1 Caractéristiques et intérêt des langages de programmation par blocs

2.1.1 Les avantages des langages de programmation par blocs pour des publics novices

Dans cette section, nous nous demandons en quoi les langages de programmation par blocs sont bien adaptés à l'introduction de la programmation auprès d'un public novice et en particulier auprès de notre public cible d'élèves de 7 à 15 ans.

Programmer consiste à utiliser un système symbolique pour planifier une suite d'opérations à faire exécuter par une machine. Classiquement, un programme prend une forme textuelle.

Depuis une dizaine d'années, une autre modalité est de plus en plus utilisée lors de l'initiation à la programmation. Cette modalité est connue sous le nom de programmation visuelle ou programmation par blocs. Nous utilisons plutôt l'expression *programmation par blocs* dans ce document.

Le principe de ce type de langage est de déplacer des blocs par glisser et déposer, et de les accrocher les uns aux autres comme des pièces de puzzle pour concevoir un programme (Weintrop, 2019). Ces blocs constituent les éléments de base, ou primitives, du langage.

Plusieurs études montrent que les langages par blocs favorisent l'apprentissage de la programmation (Bau et al., 2017 ; Hu et al., 2021 ; Weintrop & Wilensky, 2018), voire le rend accessible dès le plus jeune âge (Bers, 2019). Nous reprenons les principaux arguments avancés par les auteurs, en les illustrant.

La programmation par blocs constitue un palier dans le passage du langage naturel au formalisme des langages textuels de programmation. Les erreurs de syntaxe sont courantes chez les débutants qui abordent la programmation avec un langage textuel. Par exemple, le symbole « : » et l'indentation (décalage vers la droite) des instructions imbriquées dans les structures de contrôle sont les premiers éléments de syntaxe rencontrés en langage Python. Ils sont abordés de manière concomitante aux notions algorithmiques et constituent des obstacles à leur apprentissage. En programmation par blocs, contrairement aux langages textuels, le formalisme du langage n'est pas géré par l'utilisateur. La syntaxe du langage est prise en charge par la forme des blocs. Les erreurs de syntaxe sont ainsi rendues impossibles car seuls des blocs adéquats peuvent être accrochés ensemble, le système empêchant l'utilisateur d'accrocher des blocs qui ne sont pas compatibles. De plus, la structure des programmes est contrainte par les emplacements où des blocs peuvent être accrochés entre eux.

Les langages textuels ont des mots clés qui constituent un lexique qu'il est

nécessaire de retenir pour pouvoir programmer. Ce n'est pas le cas pour la programmation par blocs dans la mesure où l'ensemble des blocs est visible sur l'interface. L'utilisateur n'a pas à se rappeler l'existence et le label d'un bloc qu'il veut utiliser, il lui suffit de le reconnaître parmi les blocs disponibles. Bau et al. mettent en regard la disponibilité des blocs via une palette sur l'interface et la fonctionnalité d'autocomplétion des langages textuels. Ils concluent que même si la fonctionnalité d'autocomplétion aide à la mobilisation des mots clés du langage, la mémorisation est quand même en partie requise et le mode de mise à disposition (alphabétique vs thématique pour la palette de blocs) est moins structurante au niveau conceptuel. La disponibilité des éléments de langage sur une palette sans mémorisation en amont constitue donc un deuxième aspect facilitant pour un public novice (Bau et al., 2017).

La programmation par blocs apporte une vue de la structure du programme qui peut s'apparenter à des *chunks*, c'est à dire des unités d'information qui se distinguent parce qu'elles ont du sens (G. A. Miller, 1956). Bau et al. mettent en regard la notation dense en symboles conventionnels mais dont le sens échappe à l'utilisateur de la programmation textuelle avec les unités de sens que constituent les blocs. Cette manière de découper le programme en unités de sens plus large que sont les blocs par rapport à des symboles textuels allège la charge cognitive lors de l'édition du programme (la capacité moyenne de rétention étant de sept *chunks* en mémoire de travail) et facilite ainsi la compréhension de la structure du programme. (Bau et al., 2017).

De plus, les labels des blocs sont plus proches du langage naturel que le code textuel équivalent car ils contiennent si nécessaire des mots de liaison explicatifs là où le code textuel contient des symboles. Des mots tels que *for* qui ne font pas sens pour l'utilisateur sont remplacés par des mots, tels que *repeat* qui décrivent explicitement le traitement. Des éléments de sémantique sont en outre portés par la couleur (attachée à une catégorie d'instructions) et la forme des blocs (par exemple type des variables).

En conséquence, les langages de programmation par blocs permettent de se focaliser sur l'aspect algorithmique de la programmation en déchargeant l'utilisateur de la gestion de la syntaxe, et en le guidant fortement pour la gestion de la structure du programme. Cette impossibilité d'erreurs syntaxiques facilite aussi l'analyse de l'activité des utilisateurs pour le chercheur, qui peut se concentrer sur l'apprentissage de notions algorithmiques sans être pollué par le bruit engendré par les erreurs syntaxiques comme c'est le cas pour les langages textuels. En plus du fait que les langages par blocs soient préconisés par l'institution scolaire pour le public cible de notre recherche, ils sont aussi bien adaptés aux analyses que nous souhaitons mener sur les notions algorithmiques de base.

2.1.2 Caractéristiques générales des environnements de programmation par blocs

Au-delà des caractéristiques des langages par blocs présentées dans la section précédente, les environnements dans lesquels ce type de langage est implémenté comprennent des fonctionnalités qui facilitent l'initiation à la programmation.

Les langages par blocs sont implémentés au sein d'environnements de programmation qui comportent une fenêtre unique. Leur installation est simple, souvent avec un accès directement en ligne dans un navigateur (Bau et al., 2017).

Au sein des environnements de programmation par blocs, nous assistons à un compromis entre une *manipulation directe* et la planification qu'impose l'exécution différée du programme. En effet, l'utilisateur a possibilité de manipuler les blocs dans la zone d'édition, de procéder à des essais en isolant des morceaux du programme principal, en n'exécutant qu'une partie du programme édité, voire en testant séparément l'exécution de certains blocs. Chaque bloc du langage est associé à un programme en arrière-plan, non visible par l'utilisateur. Une partie de la complexité lui est ainsi masquée. Bau et al. parlent de manipulation d'abstractions de haut niveau (*high-level abstractions*).

Dans les environnements de programmation par blocs, l'exécution des programmes est rendue visible (*visible state*) (Bau et al., 2017). Parmi les fonctionnalités qui se rapportent à cet aspect, nous trouvons, le mode d'exécution *pas à pas*, qui montre la correspondance entre le bloc dans le programme et l'action exécutée. Une fonctionnalité plus avancée, non encore largement répandue, consiste à permettre à l'utilisateur de faire des retours en arrière dans la chronologie de l'exécution du programme. Une autre consiste en l'affichage à l'écran du contenu des variables utilisées dans le programme. Ces fonctionnalités permettent à l'utilisateur de suivre l'exécution de son programme, et facilitent de ce fait l'identification des erreurs et le débogage.

Néanmoins, quelques points de vigilance concernant l'utilisation des environnements de programmation par blocs sont répertoriés. Les emboîtements de blocs prennent plus de place sur l'interface que le code textuel équivalent. Lorsque la zone d'édition est saturée et que seule une partie du code est visible, il peut devenir difficile de se repérer dans le programme, afin de trouver un endroit précis, par exemple lors du débogage (Bau et al., 2017). Ranger certains blocs dans la même catégorie peut induire des confusions (par exemple structure conditionnelle et boucle conditionnelle). Il peut s'avérer difficile de déplacer des blocs déjà accrochés à d'autres. Appréhender que plusieurs blocs peuvent être placés à l'intérieur des blocs en forme de 'C' n'est pas forcément intuitif pour les nouveaux utilisateurs. De plus, les blocs de pivotement, largement présents dans ces environnements, sont l'objet de nombreuses confusions dans le sens de rotation (Fraser, 2015).

Par ailleurs, Fraser incite à questionner la durée durant laquelle la programmation par blocs est proposée aux élèves. En l'absence d'une stratégie de transition vers la programmation textuelle, il met en garde contre un phénomène de dépendance (« *gateway drug that gets students addicted* ») (Fraser, 2015).

Nous retenons les caractéristiques des environnements de programmation par blocs, leurs avantages et leurs points de vigilance. Nous les prenons en compte dans l'organisation de notre protocole expérimental et dans la structuration de nos analyses.

2.2 Quelques exemples d'environnements de programmation par blocs

Plusieurs travaux répertorient et comparent un certain nombre d'environnements de programmation par blocs (João et al., 2019, p10), (McGill & Decker, 2020), (Begosso et al., 2020), (Fagerlund et al., 2021, p.42), (AbdulSamad & Romli, 2022, p.630). Nous en relevons quelques-uns, par rapport auxquels nous positionnons l'environnement choisi pour notre recherche.

2.2.1 Le logiciel Scratch

L'environnement de programmation par blocs le plus utilisé de par le monde est Scratch (Fagerlund et al., 2021 ; Zhang & Nouri, 2019). Le logiciel Scratch a été développé par le MIT media lab (Resnick et al., 2009). Il est disponible gratuitement en ligne dans plus de 70 langues ou peut être installé localement sur une machine. Il est destiné aux enfants à partir de huit ans en contexte scolaire et extra-scolaire, mais aussi aux adultes novices en informatique. Il permet de programmer des animations, des jeux et de les partager avec la communauté en ligne.

Scratch est un logiciel très complet, qui permet d'aborder la programmation sans être confronté dans un premier temps à des difficultés d'ordre technique (installation, enregistrement de fichiers dans une arborescence,..). Il est possible de mener un projet entier sans sortir de l'environnement Scratch.

Dans l'environnement de programmation Scratch, l'utilisateur conçoit des morceaux de programmes, appelés scripts, en accrochant des blocs les uns aux autres comme expliqué dans la section 2.1.1 au début de ce chapitre.

Les blocs sont rangés dans des catégories, et tous les blocs d'une même catégorie sont de la même couleur, qui lui est réservée. La version par défaut, sans extension, comporte huit catégories : mouvement, apparence, son, événements, contrôle, capteurs, opérateurs, variables et « mes blocs ». Des extensions existent,

qui permettent d'ajouter des fonctionnalités au logiciel.

Une fonctionnalité de partage permet à l'utilisateur de stocker son projet en ligne et de le rendre accessible aux autres utilisateurs. La communauté est très étendue, plusieurs millions de projets sont partagés et accessibles à tous.

Nous relevons cependant quelques points de vigilance tant au niveau informatique qu'aux niveaux pédagogique et didactique.

L'environnement de programmation Scratch comporte de très nombreux blocs même dans sa version par défaut (plus de 130 blocs). Un utilisateur novice peut se sentir noyé dans cet environnement très riche, avoir des difficultés à repérer les blocs les plus faciles à prendre en main et finalement errer sans véritable finalité. En effet, il n'est pas possible de restreindre l'accès à un sous-ensemble de blocs pour faciliter la prise en main de l'environnement. Cette difficulté est relevée par Romero, Lille et al. en contexte scolaire (Romero, Lille et al., 2018). Elle peut être partiellement contournée en chargeant un projet avec une sélection de blocs déjà sortis du menu et placés dans l'éditeur, ce que les auteurs cités précédemment ont fait. Ces difficultés de prise en main sont identifiées aussi pour les enseignants (Haspekian & Gélis, 2021). Les auteurs notent une maîtrise insuffisante pour leur permettre de mobiliser cet environnement efficacement dans leur enseignement.

Une fois un petit nombre de blocs pris en main, les élèves ne cherchent pas à enrichir le sous-ensemble de blocs qu'ils maîtrisent. Une analyse de 250 000 projets Scratch réalisée en 2016 par Aivaloglou et Hermans montre que la plupart des projets Scratch sont de taille réduite, que certains concepts comme les procédures sont très peu utilisés, que les programmes sont révélateurs de mauvaises pratiques comme de trop gros scripts, du code « mort », du code redondant ou des messages non appariés (Aivaloglou & Hermans, 2016).

Les quelques blocs qui permettent de manipuler les notions de base de l'algorithmique et qui correspondent aux instructions que l'on retrouve dans la plupart des langages de programmation actuels (Python, C/C++, Java..) sont noyés dans l'ensemble des blocs. Ils sont difficilement distingués de blocs spécifiques au logiciel Scratch.

2.2.2 Quelques alternatives au logiciel Scratch

Nous mentionnons quelques alternatives au logiciel Scratch, en précisant leurs différences principales avec celui-ci.

Blockly

Blockly est un langage de programmation par blocs similaire à Scratch du point de vue de l'utilisateur. Il est issu d'un projet open source de Google. La

principale différence avec Scratch est que l'éditeur Blockly est conçu comme une interface de programmation générique pouvant être intégrée au sein d'autres applications, alors que Scratch constitue un logiciel en lui-même. Le langage Blockly est notamment utilisé pour le parcours blockly.games et par la plateforme code.org, plateforme gérée par une organisation à but non lucratif qui vise la mise à portée de la programmation le plus largement possible, et en particulier à en faciliter l'accès pour les femmes et les groupes sous-représentés (Kalelioğlu, 2015).

Snap!

Snap! est un autre logiciel de programmation par blocs similaire à l'environnement Scratch. Inspiré de la première version du logiciel Scratch, Snap! est développé par l'Université de Californie à Berkeley. Snap! est plus orienté vers l'aspect apprentissage de la programmation informatique que Scratch. En l'occurrence, Snap! comporte plusieurs instructions et structures de données largement utilisées en programmation informatique et manquantes dans le logiciel Scratch. Entre autres, l'utilisateur dispose avec Snap! de la boucle *for* pour laquelle la variable de boucle est explicite, de la boucle *tant que*, de fonctions avec valeur de retour (seules les procédures sont disponibles dans Scratch), de tableaux multi-dimensionnels (seules les listes sont disponibles dans Scratch). Avec Snap!, il est possible de créer des variables locales, dont la portée est le script, et des variables globales (Harvey et al., 2013).

Scratch Jr

Le logiciel ScratchJr est une adaptation de Scratch pour un public plus jeune, à partir de 5 ans. Comme Scratch, il a été développé par le MIT Media Lab. ScratchJr est disponible essentiellement sur tablette. Hormis un plus petit nombre de blocs mis à disposition, une différence notable est l'orientation du programme, qui est horizontale au lieu de l'orientation verticale classique. Du point de vue de la recherche, l'initiation à la programmation avec le logiciel ScratchJr est étudié notamment par Bers (Bers, 2018, 2019; Bers & Resnick, 2015).

2.2.3 L'environnement de programmation par blocs choisi pour notre recherche

Étant donné que l'âge de notre public cible s'étend jusque 15 ans, nous éliminons ScratchJr. Comme nous souhaitons contrôler le menu des blocs que

nous mettons à disposition, nous nous tournons vers un éditeur de type Blockly. Néanmoins, nous souhaitons garder l'apparence des blocs Scratch, pour rester le plus proche possible des exemples donnés dans les documents d'accompagnement des programmes scolaires. L'environnement de programmation Algoréa offre la possibilité d'utiliser des blocs Scratch, mis à disposition au sein d'un menu restreint contrôlé par le concepteur en fonction de la tâche à réaliser. Nous détaillons les fonctionnalités de l'environnement de programmation Algoréa, que nous avons choisi, dans le chapitre 7.

2.3 Les environnements de programmation par blocs considérés comme des EIAH

Un Environnement Informatique pour l'Apprentissage Humain (EIAH) au sens de Tchounikine (2009) est un logiciel spécifiquement conçu dans le but d'amener un apprenant à développer une activité favorable à l'atteinte d'objectifs d'apprentissage. Dans notre cas, le logiciel est un environnement de programmation par blocs, et les objectifs d'apprentissage concernent les notions de base en algorithmique et programmation.

2.3.1 Le concept de micromonde

Rieber (1996), cité par Pelánek et Effenberger (2020, p.4) définit un micromonde comme une version réduite, mais complète, d'un domaine d'intérêt. Un micromonde réfère à un environnement contrôlé et simplifié qui est régi par des règles spécifiques, et avec lequel l'utilisateur peut interagir.

Dans le domaine de l'initiation à l'informatique, le premier micromonde, qui consiste à contrôler une tortue virtuelle via le langage LOGO pour dessiner des figures géométriques, a été créé par Papert (1980). Papert se situe dans une perspective constructionniste de l'apprentissage. Il considère que les micromondes sont des outils qui permettent aux enfants, par l'exploration et la manipulation, de structurer leurs idées. Pour Baron et Bruillard, le succès du micromonde *turtle* et du langage LOGO associé provient de la capacité à piloter un robot pédagogique, ce qui permet « une transition entre un *faire faire*, à la fois abstrait et difficile surtout pour des enfants, et le *faire*, puisque l'utilisateur (le programmeur) peut se mettre à la place de la tortue et exécuter lui-même ce qu'il prévoit de lui demander. Le programme est une traduction des actions que lui-même (ou la tortue effectue). » (Baron & Bruillard, 2001).

En 2012, Papadopoulos et Tegos ont passé en revue plusieurs micromondes utilisés pour l'initiation à la programmation (Papadopoulos & Tegos, 2012).

Hormis le logiciel Scratch, considéré comme un micromonde, les plus répandus dans le champ de l'initiation à la programmation informatique sont donc ce micromonde *turtle* issu des travaux de Papert dans les années 1980 et le micromonde *Karel the robot*. Le micromonde *turtle*, en langage textuel LOGO à l'origine, est par la suite décliné dans plusieurs langages de programmation dont des langages par blocs. De son côté, le micromonde *Karel the robot*, à l'origine popularisé par les travaux de Pattis (1981), cité par Pelánek et Effenberger (2020), a été repris dans de nombreux micromondes dont le point commun est la commande d'un robot virtuel sur une grille en deux dimensions. Notre recherche est ancrée dans ce dernier type de micromonde.

2.3.2 Deux approches pour l'initiation à la programmation

Deux principales approches sont répertoriées pour initier à la programmation, notamment dans des environnements de programmation par blocs : une approche orientée algorithmique et une approche orientée créativité (Tchounikine, 2017).

Dans l'approche orientée algorithmique, une tâche est assignée à l'utilisateur, avec l'objectif de faire entrer celui-ci dans un processus de résolution de problème. Chaque exercice est conçu avec pour objectif la mobilisation d'un concept précis. Les problèmes sont organisés selon une progression dans les notions à aborder. Seul un sous-ensemble de blocs est mis à disposition de l'utilisateur pour résoudre le problème. C'est l'approche adoptée par exemple par la plateforme code.org.

La deuxième approche est l'approche dite créative. Dans cette approche issue du constructionnisme (Papert, 1980), un environnement de programmation complet est mis à disposition de l'apprenant sans qu'aucune tâche ne lui soit prescrite. La démarche CAL (*Coding as Another Language*) proposée par M. Bers relève de cette approche (Bers, 2019).

Évaluation des programmes conçus

Une différence majeure entre les deux approches se situe au niveau de l'évaluation des programmes conçus : par l'utilisateur ou par le système.

Dans l'approche créative, l'évaluation des programmes par l'utilisateur lui-même est privilégiée. Celui-ci évalue si le programme qu'il a conçu fait ce qu'il attend. Dans le cas contraire, deux comportements peuvent être adoptés : corriger le programme afin qu'il fasse ce qui est visé, ou s'adapter à ce que le programme fait en l'état. L'évaluation est laissée à la charge de l'utilisateur par exemple au sein des logiciels Scratch et Snap !.

Dans un contexte scolaire, cette évaluation des programmes peut se retrouver à la charge de l'enseignant. Elle implique d'une part que celui-ci se soit doté d'une bonne maîtrise des notions en jeu et de bonnes pratiques en terme de programmation, ce qui est loin d'être le cas général actuellement. D'autre part, le nombre de projets peut rapidement rendre l'évaluation chronophage. Dans ce contexte, un outil d'évaluation externe, tel Dr Scratch, peut constituer une aide pour évaluer les projets des élèves (Moreno-León et al., 2015).

Dans le cas d'une tâche prescrite à réaliser dans une approche algorithmique, les programmes exécutés sont évalués par le système, à travers une fonction d'évaluation implémentée par les concepteurs, fonction qui détermine si le programme résout la tâche demandée. C'est le cas pour les exercices de la plateforme code.org, et aussi pour ceux de l'environnement Algoréa que nous utilisons.

Éléments de comparaison entre les deux approches

Alors que l'approche algorithmique cible précisément des concepts de programmation, l'approche créative est envisagée en lien avec les apprentissages dans les autres disciplines (Bers, 2019). Plusieurs reproches sont formulés à l'encontre de l'approche algorithmique, parmi lesquels le fait qu'elle ignore la programmation comme moyen d'expression et de communication. Un autre point de vigilance concerne la motivation à long terme (Resnick & Siegel, 2015).

La mise à disposition de l'ensemble des blocs du langage dans l'approche créative donne à l'utilisateur l'opportunité d'explorer la richesse du langage, mais en pratique, les utilisateurs restent dans leur zone de confort, et n'utilisent que des blocs déjà connus, certaines catégories de blocs restant peu utilisées (Aivaloglou & Hermans, 2016). En revanche, l'approche algorithmique contraint la découverte de nouveaux blocs, car il est indispensable de les utiliser pour réussir à résoudre le problème posé. Bien que partisane de la seconde approche, Bers (2019, p. 504) et Resnick et Siegel (2015, pp1-3) reconnaissent que l'approche algorithmique peut aider à apprendre les concepts de base de la programmation. Cependant, s'ils admettent une dimension résolution de problème à l'approche créative, ils l'envisagent au service de l'expression personnelle.

Ainsi, les deux approches diffèrent sur qui décide de la tâche que doit exécuter la machine, et qui évalue l'exécution du programme conçu : le concepteur de l'application par l'intermédiaire de l'environnement de programmation, ou l'utilisateur lui-même. Dans la pratique en contexte scolaire, une troisième modalité est parfois constatée : l'enseignant prescrit ce que doit exécuter la machine, même si l'environnement utilisé est conçu pour une approche créative.

Dans notre travail, nous nous focalisons sur l'apprentissage des concepts

de base en algorithmique. Dans ce cadre, nous retenons l'approche orientée algorithmique, qui nous paraît la mieux adaptée pour contrôler les variables du protocole expérimental, et pour recueillir des traces d'activité précises. Nous sommes conscient qu'une telle approche aura intérêt à être articulée avec une approche créative, qui ne rentre pas dans le cadre de notre étude.

2.4 Focus sur le cas des puzzles de programmation

L'approche algorithmique retenue nous amène à nous focaliser sur la caractérisation des tâches impliquées. Nous adoptons le terme anglophone de *puzzle* pour les désigner.

Pelánek et Effenberger définissent un puzzle comme un problème structuré et attrayant dont les règles et les critères de solution sont clairement explicités (Pelánek & Effenberger, 2020). Il s'agit de problèmes fermés, où la tâche à réaliser est prescrite et pour lesquels une solution proposée peut être évaluée de manière exacte, comme valide ou non. Dans le cas des puzzles de programmation, la solution est évaluée par l'environnement de programmation, par opposition aux problèmes ouverts dans Scratch, pour lesquels les programmes sont exécutés mais non évalués par le système.

Vahldick et al. ont réalisé une revue de littérature pour les puzzles dans le domaine de la programmation (Vahldick et al., 2014). Plus récemment, le cas des puzzles de programmation au sein de micromondes de programmation par blocs a été analysé par Pelánek et Effenberger (Pelánek & Effenberger, 2020). Ces auteurs jugent les puzzles pertinents pour l'initiation à la programmation car ils permettent de :

- pratiquer la pensée informatique à travers la résolution de problème
- introduire les concepts de base de la programmation. Ils citent la séquence d'instructions, la répétition, la structure conditionnelle et les variables.
- motiver les élèves à approfondir leur apprentissage de la programmation

Cependant, selon Linehan et al., les puzzles doivent respecter les principes suivants pour être pertinents (Linehan et al., 2014) :

- Les principales compétences visées dans chaque puzzle sont introduites séparément.
- Chaque compétence est introduite par une énigme simple qui ne nécessite que des performances de base dans cette compétence.
- L'utilisateur est en mesure de pratiquer et d'intégrer cette compétence en s'appuyant sur les compétences acquises précédemment.

- Les énigmes gagnent en complexité jusqu'à ce que la prochaine nouvelle compétence soit introduite.

Afin de générer des puzzles qui respectent ces principes, Pelánek et Effenberger signalent des paramètres qui sont à la disposition des concepteurs :

- Aspect des blocs, avec un label visuel ou textuel, paramétrés ou non
- Choix des commandes mises à disposition
- Manière de mettre à disposition ces commandes, via un menu ou non
- Fixation de limites
- Regroupement des puzzles

Les exercices donnés au concours Algoréa auquel est adossée notre recherche sont des puzzles de programmation. L'exploration de la littérature nous apportent des repères sur lesquels nous appuyer pour les analyser.

2.5 Synthèse

La programmation par blocs est largement reconnue comme facilitant l'introduction de la programmation auprès d'un public novice. Elle est préconisée dans le contexte scolaire français pour les élèves jusque 15 ans.

La programmation par blocs présente quelques spécificités qui orientent nos choix et nos investigations à propos des premières confrontations avec les notions de base en algorithmique. Du point de vue des concepts intégrateurs de la science informatique discutés dans le chapitre 1, le formalisme du langage est porté par le système, rendant impossible les erreurs de syntaxe, de sorte qu'en programmation par blocs, les concepts d'algorithme et de programme n'ont pas besoin d'être absolument distingués lors de l'analyse. Le concept de machine est pour sa part rendu visible dans une fenêtre d'exécution.

Parmi les approches existantes pour l'initiation à la programmation par blocs, nous optons pour une approche algorithmique, à travers la confrontation avec des puzzles de programmation dans un micromonde dont l'objet est de commander un robot virtuel sur une grille en deux dimensions. Nous retenons les paramètres identifiés par Linehan et al. (2014) et Pelánek et Effenberger (2020) à propos des puzzles de programmation. Nous analysons les puzzles du concours de programmation Algoréa utilisés pour cette recherche en nous appuyant sur ceux-ci.

Les caractéristiques des environnements de programmation par blocs relevées dans ce chapitre, et celles plus spécifiques des puzzles de programmation dans un micromonde, sont contextualisées à l'environnement Algoréa dans le

chapitre 7. Afin d'en approfondir l'analyse, nous avons besoin de nous doter d'un cadre théorique. C'est pourquoi nous introduisons l'approche instrumentale de Rabardel (1995) dans le chapitre 5, approche que nous appliquons à notre contexte dans le chapitre 10.

À la fois en raison des caractéristiques des environnements de programmation par blocs et en raison de la nature de la tâche à réaliser, l'aspect visuel apparaît comme occupant une place essentielle : grille portant les données à traiter, structure du programme constituée de l'agencement d'entités visuelles que sont les blocs, retour visuel de l'exécution. Nous retenons cette place importante des éléments visuels pour la structuration de nos investigations.

Notre recherche sur les notions de base en algorithmique, et notamment la notion de boucle, étant articulée autour du concept de motif, nous questionnons la forme que prend ce concept lors de la confrontation avec un puzzle de programmation en langage Scratch. À cette fin, nous effectuons des investigations dans la littérature dans le chapitre 3, de manière large, ce qui nous donne les points d'appui pour un travail théorique sur le concept de motif contextualisé à la programmation par blocs, que nous présentons dans le chapitre 11.

Chapitre 3

Le concept de motif

Sommaire du présent chapitre

Dans ce chapitre, nous approfondissons l'étude du concept de motif autour duquel s'articule notre recherche. Nous commençons par explorer les différentes acceptions du terme *motif* dans le but d'en construire une définition pour le champ de la didactique de l'informatique. Nous passons ensuite en revue la littérature anglophone relative à la pensée informatique afin de relever et d'analyser les processus cognitifs et les tâches impliquant des motifs. Doté de ces repères, nous revenons aux documents émanant de l'institution scolaire française afin de discuter la place du concept de motif dans ce contexte.

3.1 Définition du concept de motif

3.1.1 Sens courant du mot *motif* en français et en anglais pour les disciplines artistiques

En français, le mot *motif* est un terme polysémique employé dans de nombreux contextes. Il fait partie du langage courant, et a probablement été déjà rencontré sous l'une de ses acceptions par un public non expert en informatique.

L'un des sens du mot *motif* est issu des disciplines artistiques. Un motif est un « dessin, ornement, le plus souvent *répété*, sur un support quelconque » [1], un « *sujet* qui domine une oeuvre d'art, un ouvrage » [2], un « *sujet* ornemental ou figuratif *formant en lui-même un tout* » [2]. On retrouve cette acception du mot motif de manière très répandue : en architecture, sculpture, orfèvrerie, tissage, tricot, marqueterie…

En musique, un motif est un « *petit élément caractéristique* d'une composition musicale, qui en assure l'unité. » [1] une « phrase mélodique possédant un sens expressif *propre*, qui parvient à prendre un certain relief dans une œuvre musicale (ou une partie de celle-ci) et qui en assure l'unité » [2]. En musique, on distingue trois sortes de motifs. Un motif harmonique est « un enchaînement d'accords *repris* au cours d'un morceau » [2]. Un motif mélodique est un « enchaînement de sons *revenant* plus ou moins librement » [2]. Un motif rythmique est un « ensemble de valeurs rythmiques pouvant *se répéter* indépendamment du contexte mélodique et harmonique » [2].

Dans la langue anglaise, le terme *motif* existe également. Il est employé dans les disciplines artistiques et désigne « an idea that is *used many times* in a piece of writing or music » [3], « a design which is used as a decoration or *as part of an artistic pattern* » [4].

1. Dictionnaire Larousse en ligne
2. Centre National de Ressources Textuelles et Lexicales (CNRTL), créé en 2003 par le CNRS
3. Cambridge dictionnary
4. Collins Dictionary

Dans ces définitions, deux aspects reviennent à plusieurs reprises : le fait que le motif peut être considéré comme un tout au sein d'un ensemble plus grand, et le fait qu'il est présent plusieurs fois au sein de cet ensemble. Nous ne trouvons pas de définition du terme *motif* contextualisée au champ de l'informatique en français, ce que nous cherchons à construire.

3.1.2 Mot anglais *pattern* dans le champ de l'informatique

En informatique, le terme français *motif* est associé au mot anglais *pattern*. D'une part, il est présent dans des travaux menés autour des *design patterns* dans le champ du génie logiciel (Gamma et al., 1995). D'autre part, *pattern recognition* désigne une habileté identifiée comme faisant partie de la pensée informatique (1.1.3).

Nous cherchons à définir le terme anglais *pattern* dans ce contexte de la pensée informatique. « *The concept of patterns* » est mentionné comme un précurseur de la modularité par Bers (2019) mais il n'est pas explicitement défini. Comme nous ne trouvons pas de travaux dans le champ de *computational science education* qui discutent de ce que recouvre le terme *pattern*, nous élargissons la recherche à la didactique des mathématiques.

3.1.3 Mot anglais *pattern* en didactique des mathématiques

Nous trouvons le terme pattern dans des travaux de didactique des mathématiques en langue anglaise. Pour Collins et Laski, le terme *pattern* désigne une séquence avec une régularité réplicable, qui peut varier selon une ou plusieurs dimensions (Collins & Laski, 2015). Nous notons une différence de sens entre les termes *motif* et *pattern* : le terme *pattern* désigne l'ensemble de la séquence d'éléments, alors que dans les définitions de la section précédente, le terme *motif* désigne l'unité de répétition. Ce terme *motif* n'est pas utilisé dans les travaux anglophones de didactique des mathématiques que nous avons consultés.

Liljedahl distingue deux types de *pattern* : les *repeating patterns* et les *number patterns* (Figure 3.1). L'expression *repeating pattern* désigne un ensemble d'éléments au sein duquel une structure cyclique générée par la répétition d'une unité est discernable (Liljedahl, 2004). Le caractère discernable comprend d'éventuelles variations prédictibles. Cette définition est reprise par d'autres auteurs (Papic, 2007 ; Warren & Cooper, 2006 ; Warren et al., 2012). Pour désigner l'unité de répétition, les auteurs emploient les expressions « *repeating unit* » (Collins & Laski, 2015 ; Warren et al., 2012), « *unit of repeat* » (Liljedahl, 2004 ; Papic, 2007 ; Warren et al., 2012), « *pattern unit* » (Papic, 2007), « *repeating element* » (Warren & Cooper, 2006), « *repeating part* » (Warren & Cooper, 2006 ; Warren et al., 2012).

L'autre type de *pattern*, « *number pattern* » (Liljedahl, 2004), ou « *growing pattern* » lorsqu'il prend une forme visuelle (Papic, 2007), n'est plus caractérisé par une unité de répétition mais par une règle de répétition, qui peut être une constante, fonction de la position dans la séquence ou d'un ou plusieurs éléments précédents. Ce type de *pattern* se rapproche de la relation de récurrence sans que celle-ci soit citée explicitement par les auteurs.

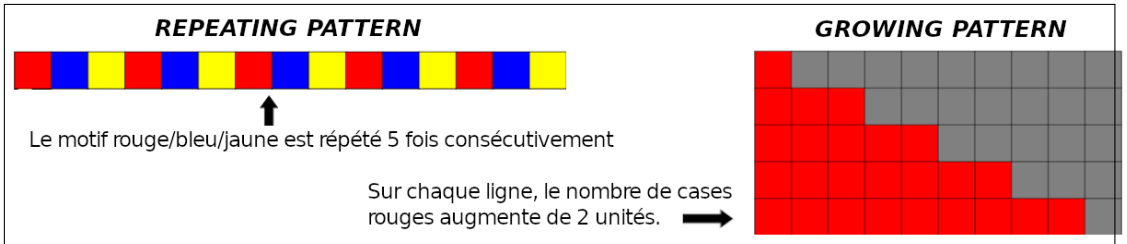


Figure 3.1 – Catégorisation de motifs en didactique des mathématiques

Des travaux ont exploré comment ce concept de *pattern* pouvait supporter l'introduction de structures mathématiques telles que la structure multiplicative (Warren & Cooper, 2007), les fonctions au sens mathématique du terme (Warren & Cooper, 2006) et l'algèbre (Warren, 2005). De manière plus générale, des études montrent l'importance des compétences en lien avec les *patterns* dans les premières années de scolarité, qui seraient corrélées avec la réussite ultérieure en mathématiques et surtout en algèbre (Lee et al., 2011 ; Rittle-Johnson et al., 2019). Burgoyne et al. ont passé en revue une trentaine d'études relatives à la compréhension des *patterns*. Ils concluent que les compétences sur les *pattern* seraient en lien avec celles en mathématiques et en lecture, mais que les résultats des études manquent malheureusement souvent de robustesse (Burgoyne et al., 2017).

3.1.4 Notre définition du concept de motif

En nous fondant sur le sens du terme *motif* dans les disciplines artistiques en français et en anglais et sur le sens du terme *pattern* dans les travaux anglophones de didactique des mathématiques, nous définissons un motif dans notre contexte comme une **entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles**.

Dans notre définition, le terme *motif* réfère à une unité, alors que pour les auteurs anglophones mentionnés dans la section précédente, le terme *pattern* fait référence à l'ensemble de la séquence. Cette nuance implique d'utiliser l'expression *séquence de motifs* pour traduire précisément le mot anglais *pattern* qui n'a pas d'équivalent en français. Par ailleurs, notre définition est plus large

que celle des auteurs mentionnés précédemment. En effet, nous n'imposons pas la structure cyclique. Des éléments peuvent tout à fait être intercalés entre deux motifs. Lorsqu'une structure cyclique est présente, nous parlerons de *motifs consécutifs* (Figure 3.2).

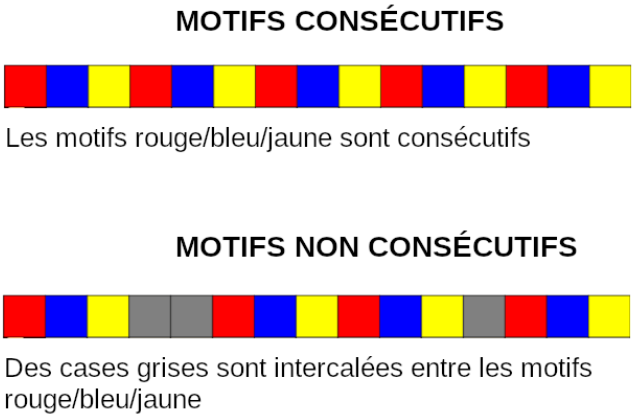


FIGURE 3.2 – Motifs consécutifs et non consécutifs

3.2 État de l'art sur l'identification de motif comme habileté relevant de la pensée informatique

Dans le but de questionner la pertinence d'associer concept de motif et initiation à l'algorithmique et à la programmation informatique, nous investiguons l'identification de motifs comme composante de la pensée informatique. En l'absence de travaux francophones qui abordent précisément cette question, nous passons en revue des travaux anglophones comportant le mot clé *computational thinking* dans le champ *computer science education*.

3.2.1 *Pattern recognition* : une expression souvent employée mais peu documentée

Comme mentionné précédemment, *pattern recognition* est une habileté répertoriée dans les définitions en extension de la pensée informatique. Dans une revue de littérature sur la pensée informatique datant de 2016, KALELIOGLU et al. relèvent *pattern recognition* comme une expression souvent mentionnée dans la définition de la pensée informatique (9%), juste derrière *abstraction*

(20%), *algorithmic thinking* (14%) et *problem solving* (13%) (Kalelioglu et al., 2016).

Cette habileté est cependant peu documentée dans la littérature. Toujours en 2016, Dasgupta et Purzer ont réalisé une revue systématique de la littérature à partir des expressions clés *pattern recognition* et *pattern generalization* associées au mot clé *education* afin d'étudier l'identification de motifs en tant qu'habileté faisant partie de la pensée informatique (Dasgupta & Purzer, 2016). Ils n'ont trouvé que deux articles investiguant ce qu'est l'identification de motifs. Aucun des deux ne relève du champ *computational thinking* : l'un est centré sur l'apprentissage des mathématiques (Tanimoto, 1998), l'autre est situé dans le contexte des études médicales (Kellman & Garrigan, 2009).

Six années plus tard, en 2022, une nouvelle revue de littérature est réalisée par Da Silva Junior et al. Les auteurs mentionnent que *pattern recogition* est rarement le thème central de l'article. Ils retiennent cinq articles en plus de la revue de littérature citée précédemment (Da Silva Junior et al., 2022) : (Abdullah et al., 2019 ; Barrón-Estrada et al., 2021 ; J. Miller, 2019 ; Polledo et al., 2021 ; Saxena et al., 2020)

Dans la classification des habilités associées à la pensée informatique de Hsu et al., *Pattern recognition* est le titre d'une catégorie (Hsu et al., 2018). Mais parmi les 19 catégories répertoriées, *pattern recognition* est la seule pour laquelle aucune ressource n'est associée.

Dans le but de collecter d'éventuels travaux plus récents, nous menons notre propre recherche bibliographique avec les expressions clés « *looking for patterns* », « *identifying patterns* », « *finding patterns* », « *pattern recognition* » et « *pattern identification* » chacun tour à tour associé à l'expression *computational thinking*.

Sur Google Scholar, nous obtenons 2560 résultats pour « *pattern recognition* », 89 pour « *pattern identification* », 6 pour « *identifying patterns* », 4 pour « *finding patterns* » et 3 pour « *looking for patterns* ». En langue anglaise, nous utiliserons donc l'expression *pattern recognition* qui est l'expression de loin la plus largement employée.

Nous consultons les résultats renvoyés par le moteur de recherche (seulement les premières pages pour *pattern recognition*) et comme Dasgupta et Purzer (2016), nous nous rendons compte que peu de ces résultats sont pertinents au regard de nos axes de travail. Nous complétons cependant avec des articles qui n'apparaissent pas dans les résultats renvoyés mais que nous avions déjà collectés au cours de recherches bibliographiques antérieures. Au total, nous travaillons avec 18 articles.

3.2.2 *Pattern recognition* : définitions

Que recouvre l'expression *pattern recognition* ? Comment est-elle définie dans les articles retenus ?

D'une part, *pattern recognition* réfère à un processus cognitif qui consiste à combiner des informations perçues dans l'environnement avec d'autres préalablement stockées en mémoire à long terme. Plus précisément, les informations prélevées dans l'environnement sont stockées en mémoire à court terme, et des contenus stockés en mémoire à long terme qui permettent de traiter ces informations de manière similaire sont mobilisés (Barrón-Estrada et al., 2021). Cette définition, qui s'appuie sur la citation de travaux en sciences cognitives, est très générale. Contextualisé au champ de la pensée informatique, *pattern recognition* consiste à examiner quelles parties d'un problème sont similaires à quelque chose que l'on a déjà résolu ou programmé (Grover & Pea, 2018).

Csizmadia et al. donnent l'exemple d'un élève qui doit programmer plusieurs dessins, dont un carré et un triangle, dans un environnement inspiré de la tortue LOGO. Après avoir programmé ces deux premières figures, on lui demande de programmer le dessin d'un octogone. Il peut reconnaître la même structure que celle de ses programmes pour le carré et le triangle, et définir une solution générale pour tous les polygones réguliers, en prenant comme paramètre le nombre de côtés et en mobilisant la relation entre nombre de côtés et angle entre deux côtés consécutifs (Csizmadia et al., 2015).

Pour cet aspect qui concerne plutôt la généralisation de processus, nous traduisons l'expression anglaise *pattern recognition* par *reconnaissance de motif*, le mot ***re**connaissance* indiquant que le motif a préalablement été rencontré, est déjà connu et stocké en mémoire à long terme.

D'autre part, l'expression *pattern recognition* est associée à l'analyse des données ; « *Observing patterns, trends, and regularities in data* » (Hsu et al., 2018 ; Saxena et al., 2020), « *recognizing patterns in data* » (Grover & Pea, 2018), « *identify patterns/rules underlying the data/information structure* » (Shute et al., 2017), « *gathering, representing and analysing data* » (Kalelioglu et al., 2016). Pour ce second aspect, nous traduisons l'expression anglaise *pattern recognition* par *identification de motif*, le mot *identification* indiquant une analyse de la structure des données, la prise de conscience de la présence de régularité, c'est à dire de motifs au sens où nous l'avons défini précédemment. Il s'agit d'un processus d'élaboration mentale au cours duquel le motif est constitué comme unité, qu'il fasse l'objet d'une première construction ou d'un rappel par analogie d'un motif similaire déjà stocké en mémoire à long terme.

Cependant, les deux aspects sont liés dans la mesure où identifier un motif dans les données peut conduire à prendre conscience que l'on a précédemment traité ce type de motif et amener à mobiliser et adapter un traitement déjà connu.

Cette double mobilisation du concept de motif est pointée par Csizmadia et al. : « *the process of recognising patterns both in the data being used and the processes/ strategies being used* » (Csizmadia et al., 2015), les stratégies mentionnées faisant référence aux algorithmes mobilisés pour résoudre un problème spécifique, qui peuvent être généralisés à une classe de problèmes similaires.

Pour notre recherche, nous retenons cette double mobilisation du concept de motif, à la fois pour l'analyse des données et pour la conception des algorithmes.

3.2.3 Pas de consensus sur la catégorisation

Pattern recognition est reconnu par un large panel d'auteurs comme faisant partie de la pensée informatique. La question qui vient alors est : à quelle facette de la pensée informatique associer cette habileté ? Nous ne trouvons pas de consensus sur ce point parmi les auteurs étudiés. Certains l'associent au processus de généralisation, d'autres au processus d'abstraction et d'autres encore considèrent *pattern recognition* comme une facette à part entière de la pensée informatique.

Pattern recognition associé à la facette généralisation de la pensée informatique

Lorsqu'associé à la facette généralisation de la pensée informatique, reconnaître des motifs s'entend généraliser une solution en réinvestissant une solution ou un morceau de solution précédemment trouvé. Nous sommes en présence de l'aspect *reconnaissance* de motifs. Le motif a été précédemment identifié et il est reconnu et adapté / généralisé pour un nouvel usage.

La reconnaissance de motif est explicitement associée au processus de généralisation soit à travers l'expression « *pattern recognition and generalization* » (Csizmadia et al., 2015 ; Fagerlund et al., 2021), soit à travers une catégorie appelée « *pattern generalization* » (Hsu et al., 2018).

Pour d'autres auteurs, la description donnée de la reconnaissance de motifs amène aussi un classement dans cette catégorie (Angeli et al., 2016 ; Barrón-Estrada et al., 2021 ; Dagienė et al., 2019 ; Palts & Pedaste, 2020).

Ainsi l'aspect reconnaissance de motif renvoie au processus de généralisation.

Pattern recognition associé à la facette abstraction de la pensée informatique

Parmi les six facettes de la pensée informatique identifiées par Shute et al. (*Decomposition*, *Abstraction*, *Algorithms*, *Debugging*, *Iteration* et *Generalization*),

pattern recognition est classé dans la facette abstraction de la pensée informatique et défini comme « *identify patterns/rules underlying the data/information structure* » (Shute et al., 2017).

Pour la programmation d'un personnage virtuel sur une grille, Brackmann et al. donnent un exemple qui illustre l'association du processus d'abstraction et de l'identification de motif dans les données. L'exemple est classique, le personnage virtuel doit parcourir un chemin marqué sur la grille (plusieurs côtés d'un carré de 5 cases de côté). L'élève doit identifier le motif visuel répété sur le chemin marqué, ce qui relève du processus d'identification de motif. Pour cela, il doit se focaliser sur les éléments pertinents du problème, en détournant son attention des détails inutiles (comme la couleur du chemin ou l'aspect des personnages), ce qui relève du processus d'abstraction (Brackmann et al., 2017).

Pattern recognition considéré comme une facette à part entière

Lorsque *pattern recognition* est considéré comme une composante de la pensée informatique, l'aspect identification de motif dans les données et l'aspect reconnaissance de motif en vue de la généralisation de solutions sont toutes deux prises en compte.

C'est le cas dans le cadre de description des compétences liées à la pensée informatique défini par Gouws et al. à partir d'une revue de littérature (Gouws et al., 2013a). En effet, une catégorie s'intitule « *Patterns and Algorithms* ». Pour ces auteurs, le concept de motif est étroitement lié à celui d'algorithme. Ils mettent l'accent sur le lien entre motifs et pensée algorithmique, en particulier lors de l'utilisation d'itérations, de récursion et de fonctions pour concevoir des solutions simples, élégantes et adaptables. L'aspect identification de motif dans les données et l'aspect reconnaissance de motif en vue de la généralisation de solutions sont présents et intimement liés l'un à l'autre.

De la même manière, dans la catégorisation des composantes de la pensée informatique de Grover et Pea, une catégorie s'intitule « *patterns and pattern recognition* » (Grover & Pea, 2018). Sa description indique que cette composante englobe les deux aspects identification de motif dans les données et reconnaissance de motif amenant à la définition de solutions généralisables.

Par ailleurs, Grover et Pea regroupent abstraction et généralisation dans une même catégorie. Ils expliquent que cacher des détails permet de se focaliser seulement sur les entrées et sorties et que dans ce sens le processus d'abstraction amène à généraliser des solutions construites pour des instances spécifiques par la définition de paramètres (Grover & Pea, 2018).

Rich et al. regroupent aussi abstraction et généralisation dans une même catégorie intitulée « *abstraction and pattern generalization* » lorsqu'ils définissent

des catégories de concept en lien avec des objectifs et trajectoires d'apprentissage (Rich et al., 2017). Cette catégorie n'aborde que l'aspect reconnaissance de motif. Par ailleurs, dans la section sur la trajectoire d'apprentissage dédiée aux structures itératives, les auteurs mentionnent l'importance de la perception de régularités car intimement liée à l'initiation à la notion de boucle. Mais le terme *pattern* n'est pas employé à ce propos.

Dans la classification des habiletés associées à la pensée informatique de Hsu et al., nous retrouvons le terme *pattern* dans plusieurs catégories (Hsu et al., 2018). D'une part, *Pattern recognition* est le titre d'une catégorie dont la définition est « *Observing patterns, trends, and regularities in data* ». Un deuxième titre de catégorie de cette classification est « *Pattern Generalization* », ce qui distingue les deux aspects identifiés dans la section précédente, à savoir l'identification de motif et la reconnaissance de motif.

3.2.4 Synthèse

Cette revue de littérature dans les travaux anglophones autour de l'expression *pattern recognition* nous permet de confirmer que le concept de motif est largement mobilisé en lien avec la pensée informatique.

Nous avons pu distinguer plusieurs processus cognitifs en rapport avec le concept de motif que nous avons défini : identification de motif dans les données, reconnaissance et généralisation de motif préalablement rencontrés. Lorsque ces deux habiletés sont distinguées, l'identification de motif est plus souvent associée au processus d'abstraction et la reconnaissance de motif au processus de généralisation.

Nous remarquons en outre que les auteurs cités n'associent pas *pattern recognition* avec la facette décomposition de la pensée informatique. Est-ce corrélé au fait que le terme *pattern* désigne l'ensemble de la suite de motifs et ne porte pas la décomposition en unités qui se répètent ? Nous trouvons un seul auteur, Bers, qui mentionne le concept de *pattern* comme un précurseur de la modularité (Bers, 2019, p.512). Par contraste, dans notre acception du terme, le motif est issu d'une décomposition afin de mettre le focus sur la partie qui se répète.

Tous les travaux analysés concernent la pensée informatique dans son ensemble. Bien que quelques exemples de situations en rapport avec l'identification et/ou la reconnaissance de motif soient donnés, aucun de ces travaux n'est focalisé sur la mobilisation du concept de motif lors de la résolution de tâches de programmation. Notre recherche vise à approfondir cet aspect.

Dans la suite de ce document, lorsque nous n'avons pas besoin de distinguer identification et reconnaissance de motif, nous utilisons l'expression *identification de motif*, pour désigner l'ensemble des aspects attachés à l'expression anglaise *pattern recognition*. En effet, cette expression est plus en phase avec

notre définition du terme *motif* comme unité de répétition. Elle englobe alors l'aspect *reconnaissance* d'un motif précédemment rencontré.

3.3 Tâches répertoriées dans les travaux anglophones de didactique des mathématiques

Comme nous ne trouvons pas de travaux qui détaillent la mobilisation du concept de motif tel que nous l'avons défini en didactique de l'informatique, que ce soit dans des travaux francophones ou anglophones, nous reprenons les tâches mobilisant le concept de motif présentées dans les travaux de didactique des mathématiques déjà mentionnés (3.1.3).

Liljedahl répertorie différentes tâches impliquant des motifs visuels consécutifs (Liljedahl, 2004). En s'appuyant sur des expérimentations menées auprès de jeunes enfants, Warren et Cooper construisent une séquence pédagogique (Warren & Cooper, 2006) puis établissent une progression dans la difficulté de ces tâches (Warren & Miller, 2010 ; Warren et al., 2012). Nous passons en revue les tâches présentes dans cette progression.

Copy a pattern - Copier une suite de motifs

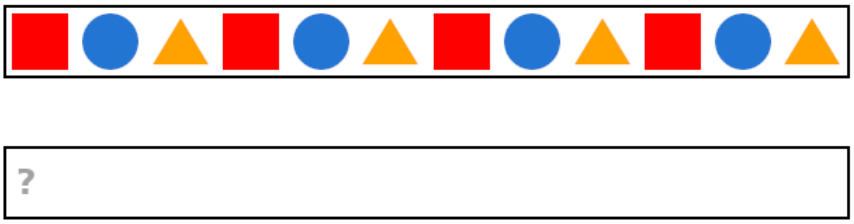


Figure 3.3 – *Copy a pattern* / Copier une suite de motifs - exemple

Copy a pattern est la situation la plus facile répertoriée dans la progression établie par Warren et Miller . Elle consiste à reproduire la suite de motifs à l'identique (Figure 3.3). Cette situation est accessible aux enfants dès l'âge de trois ans lorsque le modèle est visible (Warren & Miller, 2010). Il est à noter que la tâche peut être menée à bien en mobilisant seulement la correspondance terme à terme, qui consiste à traiter les éléments un par un indépendamment les uns des autres. Cette situation n'impose pas de prendre conscience de la structure répétitive de la suite (Collins & Laski, 2015). Elle peut par ailleurs être résolue sans tenir compte de l'ordre des éléments. Il s'agit donc plutôt d'une

situation d'imprégnation. Pour complexifier la tâche, le modèle peut être placé hors de la vue au moment de le reproduire (Papic, 2007).

Continue a pattern - Continuer une suite de motifs

Figure 3.4 – *Continue a pattern* / Continuer une suite de motifs - exemple

Dans cette situation, une suite de motifs est donnée et il est demandé de continuer la suite en ajoutant les motifs suivants (Figure 3.4). Cette extension de la suite peut s'envisager dans plusieurs directions. Pour une séquence linéaire, l'extension est demandée vers la droite et/ou vers la gauche (Warren & Cooper, 2006 ; Warren & Miller, 2010).

Complete a pattern - Retrouver les éléments manquants dans une suite de motifs

Figure 3.5 – *Complete a pattern* / Retrouver les éléments manquants d'une suite de motifs - exemple

Cette situation est composée d'une suite de motifs dont quelques éléments ont été enlevés. La tâche consiste à compléter la suite de motifs en retrouvant les éléments manquants (Figure 3.5). (Papic, 2007 ; Warren & Miller, 2010).

Transfer a pattern - Transférer une suite de motifs d'une représentation vers une autre

Transfer a pattern correspond à la situation où une suite de motifs est donnée dans une certaine représentation. Il est alors demandé de reproduire le motif en utilisant un autre matériel (Figure 3.6), qui n'est pas similaire visuellement – coccinelles et phares vs formes (Warren & Miller, 2010), voire dans une autre modalité – formes vs mouvements corporels (Warren & Cooper, 2006).

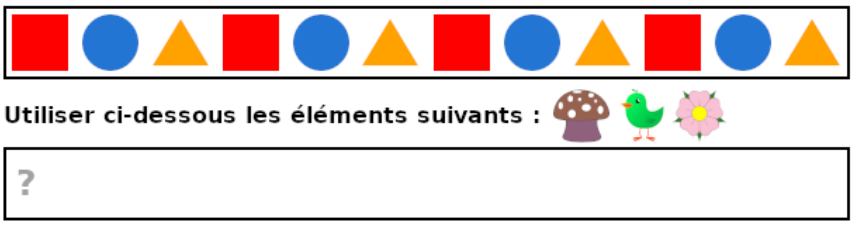


FIGURE 3.6 – *Transfer a pattern* / Transférer une suite de motifs d'une représentation vers une autre - exemple

La stratégie de correspondance terme à terme reste possible lors d'une tâche de transfert d'une représentation vers une autre si la suite de motifs dans sa représentation initiale reste visible pendant le transfert, ce qui semble être le cas dans les travaux des auteurs cités dans la section précédente. Pourtant ces auteurs indiquent que l'objectif de cette tâche est de prendre conscience de la structure commune des deux suites de motifs, ce qui nous pose question. En effet, pour s'assurer de la prise de conscience de la structure du motif, il nous semble que la représentation initiale ne devrait pas être visible au moment de la nouvelle représentation.

Identify the repeat - Identifier un motif en l'isolant

FIGURE 3.7 – *Identify the repeat* / Identifier un motif en l'isolant - exemple

Lors de la tâche *Identify the repeat*, il est demandé d'isoler le motif qui est répété. L'activité d'identification de motif amène donc à repérer le motif en tant qu'unité (Figure 3.7). C'est la situation fondamentale qui révèle la compréhension de la structure de la suite de motifs (WARREN & MILLER, 2010). En effet, la stratégie de correspondance terme à terme, qui consiste à traiter les éléments du motif un par un sans considérer celui-ci dans sa globalité, est systématiquement mise en échec lors de cette tâche (COLLINS & LASKI, 2015).

Create a pattern - Créer une suite de motifs

La tâche *Create a pattern* est mentionnée par plusieurs auteurs comme une activité de synthèse qui induit plusieurs des tâches précédemment décrites. PAPIC

demande de créer une suite de motifs qui constitue une marelle puis dans un second temps d'identifier l'unité de répétition (Papic, 2007). Dans l'étude de Warren et Miller avec des élèves de 4 ans, tous les participants ont produit une alternance (Figure 3.8), le motif le plus élémentaire, lors de cette activité (Warren & Miller, 2010).

Figure 3.8 – Créer une suite de motifs - exemple d'alternance

Contribution au positionnement de notre recherche

Identifier un motif en l'isolant est la tâche la plus difficile de la progression établie à l'issue des expérimentations de Warren et al. (2012). Cette tâche est également pointée comme difficile par Baron et Drot-Delange (2016a). Les auteurs indiquent que l'identification de motif est accessible aux élèves à partir de 6-7 ans dans le contexte de l'étude. Une autre étude indique que moins d'un tiers des élèves de 3 à 5 ans sont capables d'isoler un motif (Collins & Laski, 2015).

Lors des expérimentations de Warren et al. (2012), seuls des motifs pour lesquels le premier et le dernier élément sont différents ont été testés. La taille du motif n'est pas non plus discutée. L'analyse reste donc de notre point de vue incomplète, les caractéristiques des motifs à identifier étant peu pris en compte dans l'évaluation de la difficulté de la situation. Nous proposons d'approfondir l'étude des caractéristiques de ces motifs visuels, avec une contextualisation à une grille (en deux dimensions) qui sert de support à une tâche de programmation d'un robot virtuel.

3.4 Concept de motif dans les documents émanant de l'institution scolaire en France

Dans notre contexte, nous avons défini un motif comme une entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles. Nous investiguons la manière dont ce concept est mobilisé dans les documents émanant de l'institution scolaire en France, alors qu'aucune mention n'est faite à la pensée informatique (Gaubert-Macon et al., 2022).

Nous cherchons d'abord des occurrences de ce concept de motif dans les programmes scolaires de 2020. Bien que notre recherche cible l'école élémentaire et le collège, il est intéressant pour ce point précis d'élargir les investigations aux programmes des cycles 1 à 4 (maternelle à la fin de collège). Nous ne trouvons pas explicitement le terme *motif* dans ces documents prescriptifs. Le concept de motif reste implicite jusqu'à la note très récente du Conseil Scientifique de l'Éducation Nationale (CSEN), publiée en juin 2023, que nous analysons dans une deuxième partie (Ciccione & Dehaene, 2023).

3.4.1 Programmes scolaires de 2020 pour l'école primaire et le collège

Nous commençons par relever, dans les programmes scolaires de 2020 pour les cycles 1 à 4, les éléments que nous analysons comme en lien avec notre définition du concept de motif.

Nous trouvons celui-ci sous deux modalités, visuelle et auditive. Ces modalités correspondent aux acceptions du terme motif présentées dans la section 3.1.1 pour les domaines artistiques. Sous forme visuelle, un motif est un ensemble d'éléments graphiques repérable comme un tout et répété sur un support (exemples sur la Figure 3.2). Sous forme auditive, c'est une suite de sons repérable comme un tout et répétée au sein d'une production sonore.

Dans les programmes scolaires, le concept de motif sous une forme visuelle est présent en maternelle à travers l'exploration des suites organisées : « Dès la petite section, les enfants sont invités à organiser des suites d'objets en fonction de critères de formes et de couleurs ; les premiers algorithmes qui leur sont proposés sont constitués d'alternances simples. Dans les années suivantes, progressivement, ils sont amenés à *reconnaître un rythme dans une suite organisée* et à continuer cette suite, à inventer des "*rythmes*" de plus en plus compliqués, à compléter des manques dans une suite organisée. ». L'attendu de fin d'école maternelle associé est « Identifier le principe d'organisation d'un algorithme et poursuivre son application ».

Ce passage du programme de cycle 1 pour le domaine « Acquérir les premiers outils mathématiques » correspond bien à la définition de motif que nous avons construite : reconnaître un *rythme* induit de prendre conscience de la présence d'une entité qui se répète, l'ensemble plus grand étant une suite organisée d'objets. L'expression « rythme dans une suite organisée » induit aussi que nous sommes dans le cas d'une séquence de motifs consécutifs.

Nous remarquons par ailleurs que l'emploi du terme *algorithme* est ambigu dans ce paragraphe : fait-il référence à la suite de motifs consécutifs ou à la succession d'étapes pour construire cette suite de motifs consécutifs ?

Dans cette modalité visuelle, le concept de motif n'est plus repérable explicitement dans le programme scolaire des cycles suivants. Est-ce un « outil mathématique », pour reprendre l'expression utilisée dans le programme du cycle 1, censé être acquis à la fin de l'école maternelle ?

Néanmoins, pour les cycles 2 à 4, nous retrouvons des aspects de notre définition du concept de motif dans la modalité visuelle dans d'autres domaines des programmes. En effet, ces programmes mentionnent à de nombreuses reprises les « régularités orthographiques » qu'il est nécessaire d'observer, d'identifier, d'interroger afin de maîtriser la langue française à l'écrit. Dans ce cas, le motif serait constitué d'une suite de lettres. Ces mêmes programmes contiennent des sections relatives aux formes géométriques qu'il s'agit de reconnaître. Dans les deux cas, nous sommes en présence d'une entité repérable, mais la notion de répétition de cette entité sur le même support est absente.

De manière explicite, le concept de motif est présent plus largement sous une forme auditive, au sein du domaine musical. Repérer un motif rythmique est une compétence visée pour les cycles 1 à 4. Par exemple on trouve pour le cycle 3 la compétence « *Repérer et nommer une organisation simple* dans un extrait musical : *répétition* d'une mélodie, d'un *motif rythmique*, d'un thème, d'une partie caractéristique, etc. ; en déduire une *forme simple (couplet/refrain, ABA par exemple)* ».

Ainsi, nous trouvons dans les programmes scolaires des cycles 1 à 4 quelques allusions au concept de motif, mais aucun en lien direct avec les éléments de programme relevant de la science informatique, pourtant présents (1.2.1). Nous sommes donc dans la situation suivante. D'une part, dans les programmes scolaires à partir du cycle 2, le concept de motif est beaucoup plus mobilisé comme unité de répétition dans sa modalité sonore que dans sa modalité visuelle. D'autre part, dans ces mêmes programmes scolaires, la programmation est presque systématiquement associée à la structuration de l'espace, sans que le concept de motif soit mentionné.

3.4.2 Note du CSEN de juin 2023

En juin 2023, le CSEN publie une note intitulée « Les motifs, source d'éveil aux mathématiques en maternelle et au primaire ». La note est rédigée par Lorenzo Ciccione, auteur d'une thèse en sciences cognitives sur « les bases cognitives et neurales de la perception et compréhension des graphiques » et Stanislas Dehaene, psychologue d'envergure internationale spécialisé en neuropsychologie, qui préside le CSEN depuis 2018 (Ciccione & Dehaene, 2023).

Bien que l'article soit focalisé sur l'apport des motifs pour l'apprentissage des mathématiques, plusieurs autres disciplines sont mentionnées (musique, graphisme, lecture). Pour sa part, l'informatique n'est mentionnée que comme

bénéficiant bien plus tard de ces compétences sur les motifs : « Nous ne proposons les motifs que comme une introduction concrète, intuitive, attrayante et précoce aux procédés d'abstraction qui seront, bien plus tard, aux fondements de concepts bien plus élaborés, tels que ceux de bijection, de groupe, de symétrie, de programme informatique, etc. »[p.9].

Dans cette note, le terme *motif* est une traduction littérale du mot anglais *pattern*, et désigne une « disposition ordonnée d'éléments qui se répètent selon une certaine règle »[p.3]. les motifs sont « des séquences dans lesquelles nous reconnaissons un certain ordre et une certaine prévisibilité, à tel point que nous pourrions les prolonger »[p.2].

Les auteurs insistent à plusieurs reprises sur la présence d'une règle, abstraite, qui régit le motif. Cette règle est définie comme la « formule abstraite qui détermine comment les éléments d'un motif sont organisés »[p.3]. La règle est une composante du motif. Un même motif peut être régi par une combinaison de plusieurs règles. Plusieurs modalités de motifs sont mentionnées, parmi lesquelles les motifs visuels et les motifs rythmiques musicaux.

Pour l'entité qui est répétée au sein du motif, les auteurs parlent d'« unité minimale »[p.3]. La définition induit que les unités minimales sont consécutives, disposées en séquence. Pourtant il est fait mention de disposition géométrique en deux dimensions, où les unités minimales ne sont pas strictement ordonnées séquentiellement. Dans un certain nombre de représentations visuelles de la note, le nombre d'unités minimales du motif n'est pas entier, ce qui selon nous marque le peu d'importance que les auteurs accordent à la tâche d'identification de cette unité minimale. Lorsque les auteurs relèvent les tâches en lien avec les motifs, l'identification explicite de l'unité minimale, pourtant répertoriée dans les travaux cités, ne figure pas parmi cette liste. Parmi la « hiérarchie d'activités », la tâche mentionnée en dernière position, donc a priori considérée comme la plus difficile, consiste à « choisir le motif qui correspond à l'exemple proposé », c'est à dire à choisir parmi les items celui qui est une transposition de cet exemple.

Alors qu'il n'est fait aucune mention à la pensée informatique, nous retrouvons nombre de ses composantes : abstraction lorsqu'il s'agit de percevoir un motif indépendamment de la spécificité de ses éléments, généralisation lorsqu'il s'agit de passer d'une modalité à une autre.

En résumé, notre définition du terme motif, qui désigne l'unité de répétition comme dans le sens en français courant, diffère de celle adoptée par les auteurs. Il est à noter que nos premières publications, antérieures à la note du CSEN, contiennent déjà cette définition (Léonard, Peter, Secq & Fluckiger, 2022 ; Léonard, Secq et al., 2022). Une conséquence est que dans notre acception, la règle de distribution des motifs au sein de l'ensemble ne fait pas partie de la définition du motif. La seule contrainte est que l'on trouve plusieurs occurrences

du motif au sein de l'ensemble considéré. Notre définition est donc plus large et peut adresser des domaines comme des agencements de lettres. Pour nous, le graphème *eau* est un motif. Il est répété dans l'ensemble des mots de la langue française, bien que sa distribution ne soit bien sûr pas régulière dans un texte.

Hormis cette différence de construction du concept, nous retrouvons les modalités de motif, et la plupart des tâches impliquant des motifs que nous avons identifiées. Dans ce travail de recherche, nous visons à montrer comment nous retrouvons ces tâches élémentaires dans le champ de l'initiation à l'algorithmique et à la programmation informatique en contexte scolaire, que cette note ne mentionne pas.

3.5 Synthèse

Dans ce chapitre, notre contribution a consisté à définir le terme *motif* dans le champ de l'initiation à l'informatique (Léonard, Secq et al., 2022).

> ***Un motif est une entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles***

Cette définition met en exergue l'unité de répétition, qui est centrale pour les tâches relatives au concept de motif. Cette acception, qui respecte le sens du mot français *motif* dans le langage courant, constitue une distinction par rapport au terme anglais *pattern*. Le fait de ne pas inclure la règle qui stipule la disposition des motifs les uns par rapport aux autres donne au concept une portée plus générale que l'acception du terme anglais *pattern*. Nous avons d'ailleurs introduit le terme *motif* comme unité de répétition dans une communication anglophone (Léonard, Peter, Secq & Fluckiger, 2022) :

> ***A motif is an entity that can be identified within a set, because it is repeated identically or with predictable variations.***

La recherche de l'expression *pattern recognition* dans les travaux traitant de la pensée informatique a amené à distinguer deux processus cognitifs à l'œuvre : identification de motif dans les données, reconnaissance et généralisation de motif précédemment rencontrés. Nous avons également montré qu'il n'y a pas de consensus sur l'identification de motif parmi les composantes de la pensée informatique, et qu'il n'existe pas, à notre connaissance, de travaux focalisés sur cet aspect.

Pour ce qui concerne les documents français émanant de l'Éducation Nationale, une note récente du CSEN (2023) établit explicitement l'importance

du concept de motif, alors que celui-ci est peu présent dans les programmes scolaires de 2020. Cependant, le champ de l'informatique, dans sa spécificité du passage du *faire* au *faire faire* n'est pas convoqué.

Or, dans le domaine de l'apprentissage des premières notions d'algorithmique, nos études de cas mentionnées dans le chapitre d'introduction ont pourtant montré l'importance de la tâche d'identification de motifs consécutifs lors de l'initiation à la notion de boucle. Lors de ces expérimentations, nous avons repéré un palier de difficulté récurrent lors du passage d'un motif de longueur un à un motif de longueur strictement supérieur à un. C'est en particulier le cas pour l' étude « MOTIF.. MOTIF.. » (Léonard et al., 2021) qui impliquent des suites de motifs visuels similaires à celles étudiées dans les travaux de didactique des mathématiques analysés.

Au vu de l'ensemble de ces éléments, nous visons d'approfondir l'étude du concept de motif lors de l'initiation à la programmation informatique. Une étude des caractéristiques des motifs à identifier plus approfondie que celle réalisée par Warren et al. nous semble notamment nécessaire. Nous nous appuyons néanmoins sur ces travaux pour contextualiser les tâches impliquant le concept de motif au champ de l'initiation à l'informatique dans le chapitre 11. Afin d'approfondir notre problématique, nous nous dotons d'un cadre théorique (partie II), puis nous concevons un dispositif expérimental (partie III) afin d'étudier l'identification de motifs lors de l'introduction de la notion de boucle bornée en programmation par blocs.

Deuxième partie

Cadres d'analyse

La deuxième partie de ce document, qui comporte à nouveau trois chapitres, regroupe les deux cadres d'analyse sur lesquels nous appuyons l'essentiel de notre propos. Le premier chapitre est consacré à la théorie des champs conceptuels (Vergnaud, 1991), le deuxième aborde l'approche instrumentale (Rabardel, 1995). Un troisième chapitre reprend l'ébauche de notre problématique de l'introduction, pour la reformuler dans les termes de ces cadres d'analyse et la décliner en questions de recherche.

Chapitre 4

La théorie des champs conceptuels

Sommaire du présent chapitre

Pour approfondir notre étude de l'apprentissage des premières notions d'algorithmique par des élèves d'âge scolaire, nous avons besoin de nous appuyer sur un cadre d'analyse relevant de la didactique. Nous souhaitons analyser les actions que les élèves réalisent lorsqu'ils résolvent des puzzles dans un environnement de programmation par blocs pour accéder au raisonnement qui est sous-jacent, quand bien même celui-ci reste implicite. La théorie des champs conceptuels de Vergnaud répond à ce besoin. En effet, cette théorie propose un cadre d'analyse pour expliquer la manière dont les sujets construisent et organisent leurs connaissances à travers des actions concrètes et des interactions avec leur environnement.

Nous présentons dans ce chapitre les principaux points de la théorie des champs conceptuels que nous mobilisons par la suite. Nous le complétons par un positionnement de notre travail par rapport aux travaux dans le champ de l'informatique mobilisant ce cadre d'analyse.

4.1 Objet d'étude

Gérard Vergnaud est l'auteur de la théorie des champs conceptuels, qu'il a élaboré dans le but premier d'analyser l'apprentissage de compétences complexes en mathématiques (Vergnaud, 1991). La théorie des champs conceptuels s'inscrit dans une approche constructiviste et cognitiviste de l'apprentissage. Vergnaud a obtenu son doctorat sous la direction de Jean Piaget à Genève. Il reprend dans ses travaux une large partie de l'approche constructiviste de Piaget, notamment le concept de schème et la théorie de l'équilibration (Piaget, 1935). En revanche, il laisse de côté tout ce qui concerne les stades piagétiens. Dans l'introduction de l'article de 1991, Vergnaud définit lui-même sa théorie comme cognitiviste. Le cognitivisme réfère au processus de traitement de l'information pour l'étude de la pensée.

La théorie des champs conceptuel prend comme unité d'analyse le couple sujet / situation. Pour Vergnaud, le mot *situation* a le sens de tâche.

La théorie des champs conceptuels vise à comprendre la conceptualisation, en particulier dans le cas des activités cognitives complexes, dont la programmation informatique fait partie (Rogalski & Vergnaud, 1987). Par conceptualisation, Vergnaud désigne « l'identification des objets du monde et de leurs propriétés et relations » (Vergnaud, 2012, p.20), que cette identification résulte des informations directement fournies par le réel, ou qu'elle résulte d'une construction. Son hypothèse est que c'est dans l'action que le sujet mobilise et développe des ressources pour faire évoluer sa conceptualisation. C'est une approche qui envisage la progressivité de la conceptualisation, qui nécessite d'être considérée sur un temps long.

4.2 Conceptualisation-en-acte

Vergnaud étudie le sujet lorsqu'il se trouve dans une situation où son but est de réaliser une tâche. Il fait l'hypothèse que toute action finalisée repose sur une *conceptualisation-en-acte*, c'est-à dire que les actions du sujet révèlent l'activité cognitive sous-jacente, qui n'est souvent pas formulée par le sujet. La conceptualisation-en-acte est intimement liée au déroulement de l'action dans les situations auxquelles le sujet est confronté. Cependant, Vergnaud montre que « la conceptualisation sous-jacente à l'action ne se suffit pas toujours à elle-même, et qu'elle est profondément transformée lorsqu'elle est explicitée, débattue, et organisée en un système cohérent de concepts, de principes et d'énoncés, c'est-à-dire lorsqu'elle prend une forme théorique. » (Vergnaud, 1996, p.275). Vergnaud distingue cette forme de connaissance construite dans l'action, qu'il nomme opératoire, d'une forme prédicative de la connaissance, c'est à dire une forme déclarative (Vergnaud, 2012). Le passage de la forme opératoire, c'est-à-dire d'une connaissance-en-acte, à la forme prédicative correspond au passage de la conceptualisation-en-acte à une conceptualisation plus conscientisée et donc énonçable. Toutefois les deux formes de conceptualisation sont intimement liées : « La conceptualisation est une condition de l'énonciation. En retour l'énonciation apporte à la conceptualisation une contribution décisive. » (Vergnaud, 2012, p.12)

Afin de préciser comment opère la conceptualisation-en-acte, Vergnaud développe deux axes. Un premier axe est porté par les situations. Le second est porté par le sujet en action, autour du concept de schème.

4.3 Classes de situation

Afin de comprendre comment opère l'activité cognitive du sujet, Vergnaud invite premièrement à analyser les situations auxquelles le sujet est confronté, et de les regrouper en classes (Vergnaud, 1991). Les classes de situation sont donc des ensembles de situations concrètes ou abstraites qui partagent des caractéristiques communes. Toutes les situations d'une même classe sont traitées sensiblement de la même manière par le sujet.

Les variables de situation sont des paramètres qui différencient des situations proches. Deux cas se présentent suivant que le changement de valeur d'une variable de situation affecte ou non la structure du traitement de la situation par le sujet. Dans le cas où le sujet traite les deux situations de façon similaire, ces deux situations appartiennent à la même classe. Dans le cas contraire, deux classes de situations sont distinguées en fonction de la valeur de la variable de situation.

En conséquence, la notion de classe de situation peut être envisagée de deux points de vue : du point de vue de l'expert, par une analyse des caractéristiques des situations, et du point de vue du sujet, en analysant la manière dont il traite les situations qui se présentent à lui.

4.4 Schèmes

Vergnaud reprend le concept piagétien de schème (Piaget, 1935). Ce concept de schème est le second axe considéré par Vergnaud pour comprendre les processus cognitifs du sujet à partir de son activité.

En effet, selon Piaget et Vergnaud, le sujet mobilise des schèmes, qui sont des unités d'action déjà prêtes, pour traiter une situation à laquelle il est confronté. Un schème n'est cependant pas une action stéréotypée, reproduite à l'identique, mais une « organisation invariante de la conduite pour une classe de situations donnée » (Vergnaud, 1991). Un schème permet d'adapter l'activité à la spécificité d'une situation.

L'apport de Vergnaud par rapport à Piaget se situe notamment dans l'élaboration d'une définition analytique du schème. Cette définition décrit les composantes d'un schème, qui comprend :

- Un but
- Des règles de conduite de l'action, qui se succèdent :
  - des règles de prise d'information
  - des règles d'action proprement dites
  - des règles de contrôle de l'action
- Des invariants opératoires de deux types, qui constituent la partie cognitive du schème
  - concept-en-acte qui est « un concept tenu pour pertinent dans l'action en situation. » (Vergnaud, 2013b)
  - théorème-en-acte qui est« une proposition tenue pour vraie dans l'action en situation. » (Vergnaud, 2013b)
- Des inférences, c'est-à-dire des raisonnements en situation qui visent à prendre en compte les spécificités d'une situation et à anticiper les résultats de l'action.

À titre d'exemple, nous renseignons de manière analytique le schème de dénombrement mentionné plusieurs fois dans les travaux de Vergnaud (1991, 2007, 2012) :

Schème de dénombrement

- **but** : « Associer un nombre à une collection discrète » (Vergnaud, 2010)
- **règles de conduite de l'action** :
  - **prise d'information** : en amont de l'action, identifier les éléments à compter, repérer quelle est l'unité
  - **action proprement dite** : pointer les unités les unes après les autres en leur associant un mot nombre
  - **contrôle** : contrôle au cours de l'action de la coordination entre le regard, le geste de pointage et l'attribution du mot nombre, recomptage éventuel à titre de vérification
- **invariants opératoires** dont certains correspondent aux principes de Gelman (Gelman & Gallistel, 1978) :
  - **concepts-en-acte** : nombre, correspondance terme à terme, cardinal
  - **théorèmes-en-acte** : « Le dernier mot prononcé est le cardinal de la collection. », « L'ordre dans lequel on considère les éléments est indifférent. »
- **inférences** : identifier le prochain élément à traiter, maintenir deux ensembles : les éléments déjà pointés et ceux qui ne le sont pas encore.

L'ensemble des schèmes dont dispose un sujet à un moment donné est structuré : « ces schèmes sont hiérarchiquement organisés, les uns étant les schèmes élémentaires destinés à être intégrés dans des schèmes de plus haut niveau, et permettant d'organiser des activités plus complexes. » (Vergnaud, 1996, p.284). Cette structure évolue avec le développement et l'exposition à de nouvelles situations, suivant les processus d'assimilation et d'accommodation (Piaget, 1935). Au fur et à mesure de l'expérience, les schèmes se complexifient : un seul schème, complexe, peut suffire à traiter une classe de situations de manière assez automatisée et immédiate.

4.5 Concept

Lorsqu'un sujet raisonne, il manipule des concepts, unités fondamentales de la pensée, construites progressivement, qui représentent des idées ou des catégories mentales. La prise en compte des classes de situations et des schèmes amène Vergnaud à définir un concept comme « un triplet de trois ensembles :

- l'ensemble des situations qui donnent du sens au concept (la référence)
- l'ensemble des invariants sur lesquels repose l'opérationnalité des schèmes (le signifié)
- l'ensemble des formes langagières et non langagières qui permettent de représenter symboliquement le concept, ses propriétés, les situations et les procédures de traitement (le signifiant) »

(Vergnaud, 1991)

4.6 Champ conceptuel

La théorie des champs conceptuels donne un éclairage sur la manière dont les concepts sont liés les uns aux autres chez les sujets et comment ces relations entre concepts influencent la compréhension, l'apprentissage et la résolution de problèmes.

Dans la suite logique de ce qui précède, Vergnaud définit un champ conceptuel à la fois comme un ensemble de situations et comme un ensemble de concepts : « Un champ conceptuel est à la fois un ensemble de situations et un ensemble de concepts. L'ensemble des situations dont la maîtrise progressive appelle une variété de concepts, de schèmes et de représentations symboliques en étroite connexion. L'ensemble des concepts qui contribuent à la maîtrise de ces situations. » (Vergnaud, 2013b, p.291)

Vergnaud invite à considérer des champs conceptuels assez larges (Vergnaud, 1991, 2013b). La mise en relation d'un schème avec la classe de situations dans laquelle il est opérant nécessite d'être exposé à des classes de situations connexes, pour pouvoir identifier les généralisations possibles du schème et les ruptures entre les classes de situations.

Par ailleurs, Rogalski et Vergnaud indiquent qu'il convient de choisir un niveau d'analyse pour chaque concept. En effet, un concept n'a pas la même signification suivant le niveau d'expertise du sujet dans le domaine. Pour Rogalski et Vergnaud, il y a « nécessité de mener conjointement une analyse épistémologique des notions en jeu et une analyse de la complexité des opérations cognitives qu'elles impliquent. Ce sont là les bases nécessaires pour l'élaboration de situations didactiques. » (Rogalski & Vergnaud, 1987, p.270).

Dans la caractérisation d'un champ conceptuel, plusieurs facteurs sont donc à prendre en considération :

- La nature des relations entre les classes de situations et entre les concepts : filiations, ruptures
- Les représentations symboliques disponibles pour représenter chaque concept
- Un niveau d'analyse pour chaque concept, en lien avec le niveau de développement des sujets considérés.

4.7 Aspect psychogénétique

Vergnaud insiste aussi sur le fait que, du point de vue du sujet, l'organisation d'un champ conceptuel est progressive, qu'elle ne peut s'envisager que sur un temps long, et qu'elle n'est éventuellement jamais achevée. L'apprentissage et le développement conceptuel sont la conséquence de la confrontation entre les connaissances préexistantes d'une personne et les nouvelles situations qu'elle rencontre. Au cours du développement, ces confrontations à des situations nouvelles conduisent à des ajustements, des clarifications et des réorganisations de la structure du champ conceptuel (Vergnaud, 1996).

4.8 Positionnement de notre travail par rapport aux travaux dans le champ de l'informatique mobilisant ce cadre théorique

Dans le champ de l'informatique, plusieurs travaux ont déjà mobilisé la théorie des champs conceptuels. Nous positionnons notre travail par rapport à ces travaux antérieurs. Deux auteurs ont cherché à définir un champ conceptuel attaché à l'initiation à l'informatique dans les années 1980 à 2000. Plus récemment, des travaux font référence à la théorie des champs conceptuels, ou à certains de ses aspects, sans que cela se superpose à notre travail.

4.8.1 Les travaux des années 1980-2000

En 1987, Rogalski et Vergnaud publient un article qui mentionne explicitement un champ conceptuel dont le périmètre est l'*alphabétisation informatique* au niveau du lycée. Ce champ conceptuel est articulé autour du concept de variable. Il inclut les structures de contrôle (Rogalski & Vergnaud, 1987). Rogalski précise aussi que « les spécificités de l'informatique doivent [...] être envisagées du

point de vue du contenu à enseigner et du point de vue du sujet qui doit acquérir des connaissances » (Rogalski, 1987), contextualisant ainsi à l'informatique les deux axes d'étude d'un champ conceptuel définis par Vergnaud.

Une décennie plus tard, Vergnaud dirige la thèse de El Rouadi intitulée *Programmation informatique et conceptualisation entre 13 et 15 ans* (El Rouadi, 1999). Dans cette thèse, un champ conceptuel de l'initiation à la programmation informatique est défini en considérant des élèves de collège débutant l'apprentissage de la programmation dans le langage Turbo Pascal. Le travail se concentre sur la caractérisation d'un schème de l'itération, ce schème adressant les concepts d'itération, de variable, de boucle et de test.

Si l'objectif de notre recherche est similaire à ceux des travaux de Rogalski et d'El Rouadi, plus de vingt ans ont passé depuis la publication de la thèse de El Rouadi, et le contexte de l'initiation à l'informatique et les possibilités de traitement et d'analyse de données ont fortement évolué, ce qui rend légitime d'interroger à nouveau le sujet. Comme El Rouadi, nous visons la définition du périmètre et la documentation d'un champ conceptuel pour l'initiation à l'algorithmique, avec un focus sur la structure itérative. Cependant, plusieurs points différencient fortement notre travail des travaux précédemment cités :

- La différence la plus notable est l'évolution dans l'environnement de programmation utilisé. Rogalski et El Rouadi considèrent des langages textuels, les langages impératifs pour Rogalski, le langage Turbo Pascal pour El Rouadi. Depuis, les langages de programmation par blocs sont apparus et ont amené un changement dans la manière d'aborder l'initiation à la programmation (chapitre 2). C'est ce dernier type d'environnement de programmation que nous avons retenu pour notre étude.
- Il en découle que la nature des problèmes est différente : problèmes numériques dans les travaux de Rogalski et El Rouadi, problèmes de déplacement d'un robot virtuel sur une grille pour notre recherche.
- La programmation par blocs permet de s'adresser à un public plus large, en particulier plus jeune. Les élèves qu'a étudié El Rouadi ont entre 13 et 15 ans, ceux visés par l'*alphabétisation informatique* de Rogalski sont des élèves de lycée. Pour notre part, nous nous intéressons à des élèves de 7 à 15 ans.
- Que ce soit Rogalski ou El Rouadi, les échantillons sur lesquels portent le travail expérimental sont de taille réduite. Avec la collecte de traces d'interaction sur une plateforme en ligne, nous sommes en mesure d'élargir considérablement la taille de l'échantillon. Cela nous ouvre des perspectives en termes de robustesse statistique.
- Depuis la publication des travaux précédents, des méthodes automatisées de traitement de données se sont répandues, des algorithmes de *machine*

learning sont devenus accessibles. Nous visons d'explorer comment ces outils peuvent supporter l'exploration et la documentation d'un champ conceptuel de l'initiation à l'algorithmique en programmation par blocs.

4.8.2 Les travaux récents, depuis 2015

Peu de travaux récents dans le champ de l'initiation à la programmation se sont inscrits dans le cadre théorique des champs conceptuels. Celui-ci a été mobilisé par Spach dans une thèse consacrée à la robotique éducative au niveau de l'école élémentaire (Spach, 2017). Le travail expérimental dans le cadre de cette thèse a mobilisé des robots de sol programmables. Les expérimentations ont été menées soit avec des cartes de programmation, soit en utilisant des fonctions pré-programmées. Bien qu'ils soient répandus, Spach n'a utilisé aucun environnement de programmation par blocs dans son travail sur la robotique pédagogique. Les analyses menées sont essentiellement qualitatives.

On trouve par ailleurs dans l'article de Denning 2017 une mention de ce qui peut référer à de la conceptualisation-en-acte. Denning cite les travaux du philosophe Michael Polanyi et discute de la différence entre « connaissances explicites » et « connaissances tacites ». Il écrit en particulier « We know more than we can say » (Denning, 2017). Denning mentionne cet auteur pour signifier que programmer relève d'après lui de connaissances tacites, ce qui est selon nous une autre manière de formuler ce qu'est la conceptualisation-en-acte.

4.9 Synthèse

Parmi les nombreux travaux de Vergnaud, nous avons sélectionné et présenté les principaux concepts sur lesquels s'appuie le présent travail de recherche. En inscrivant celui-ci dans le cadre de la théorie des champs conceptuels, étudier les premières confrontations d'élèves de 7 à 15 ans avec des puzzles d'initiation à la programmation revient à les analyser selon les deux axes préconisés par Vergnaud .

- l'axe des situations, à travers une analyse épistémologique du point de vue de l'expert
- l'axe des schèmes, centré sur l'activité du sujet

L'analyse suivant ces deux axes aboutit finalement à définir un champ conceptuel englobant les notions algorithmiques en jeu pour la programmation par blocs au niveau de l'école élémentaire et du collège. L'objectif est d'identifier un noyau conceptuel qui soit pertinent au niveau de l'école obligatoire, et constitue aussi un invariant par rapport aux niveaux conceptuels du niveau scolaire

suivant (lycée). Il sera dans ce sens nécessaire de distinguer les concepts algorithmiques qui sont des invariants conceptuels (Rogalski, 1987) et ont une portée générale par rapport à des spécificités de l'environnement et du langage choisis pour l'expérimentation.

Pour ce faire, nous nous appuyons notamment sur les travaux de Rogalski, qui a documenté un champ conceptuel de l'*alphabétisation informatique* pour la programmation avec un langage textuel au niveau du lycée dans les années 1980. Pour Rogalski, les concepts informatiques, leurs relations et leurs représentations symboliques sont constitutifs de ce champ conceptuel (Rogalski, 1988a, p. 132).

Si notre recherche présente des similarités avec ces travaux antérieurs au regard de l'objectif de l'identification d'un champ conceptuel relatif à l'initiation à la programmation informatique, l'évolution significative du contexte en termes d'environnement de programmation, de tranche d'âge ciblée et de capacités de traitement et d'analyse de données, ainsi que la place centrale accordée au concept de motif, justifient ces nouvelles investigations.

Chapitre 5

L'approche instrumentale

Sommaire du présent chapitre

Dans le contexte actuel de l'initiation à la programmation avec des langages par blocs, la prise en main de l'environnement de programmation nous semble occuper une place importante, et nous éprouvons le besoin de nous doter d'un cadre théorique spécifique pour questionner ce point. C'est pourquoi, en complément de la théorie des champs conceptuels de Vergnaud (1991), nous mobilisons l'approche instrumentale de Rabardel (1995).

Les rapports entre ces deux cadres théoriques sont notamment discutés dans les articles de Vidal-Gomel et al. (2022) et Gouédard et Bationo-Tillon (2022). Vergnaud et Rabardel partagent une perspective constructiviste et développementale du sujet, dans une filiation à Piaget, autour des concepts de schème, de situation et de classes de situation. Les apports de Vergnaud concernent les concepts d'invariants opératoires, de conceptualisation-en-acte et de champs conceptuels. De son côté, Rabardel se concentre spécifiquement sur les situations d'activité instrumentée ainsi que sur les situations d'activité collective instrumentée (Rabardel, 1995). Pour ces dernières, un aspect socio-constructiviste est présent de manière complémentaire, dans une filiation à Vygotsky. Rabardel se situe dans une approche *anthropotechnique*, c'est à dire qu'il considère les objets et systèmes comme des moyens d'action pour les sujets. Il introduit de nouveaux concepts afin d'analyser la relation entre sujets et objets, et entre action et cognition : les concepts d'artefact, d'instrument, de genèse instrumentale. Il caractérise les processus de production de connaissances dans l'utilisation d'objets.

Dans ce chapitre, nous présentons les éléments du cadre théorique de Rabardel que nous mobilisons pour analyser la prise en main de l'EIAH utilisé dans le cadre de notre recherche. Nous limitons nos investigations aux activités instrumentées, dont l'activité du sujet en train de programmer dans le contexte d'un concours en ligne fait partie.

5.1 La situation instrumentée comme unité d'analyse

Rabardel définit une situation d'activité instrumentée comme une situation dans laquelle un sujet utilise un outil (objet technique, mais aussi logiciel, outil conceptuel) pour mener à bien son activité. Pour Rabardel, les situations d'activité instrumentée constituent une classe de situations, qu'il adopte comme unité d'analyse (Rabardel, 1995, p.4). Une situation d'activité instrumentée est constituée de trois pôles : sujet, objet et instrument. Rabardel étudie les interactions entre ces trois pôles.

Le sujet est considéré comme l'acteur principal de la situation, acteur qui

utilise des objets pour atteindre ses objectifs. C'est pourquoi Rabardel étudie les objets et les systèmes du point de vue du sujet, qui constitue le premier pôle de la situation d'activité instrumentée.

Le deuxième pôle est occupé par l'objet de l'action, qui est le but que le sujet cherche à atteindre à travers son comportement. L'objet de l'action peut concerner un objet concret, mais aussi un objet abstrait à travers par exemple un problème à résoudre, un besoin à satisfaire. Dans l'approche instrumentale, la compréhension de l'objet de l'action est cruciale pour comprendre le comportement du sujet et les processus cognitifs sous-jacents.

Ces deux premiers pôles se rapprochent du couple sujet/situation qui est l'unité d'analyse du cadre théorique de Vergnaud. L'objet de l'activité au sens de Rabardel peut être mis en correspondance avec le but de la tâche dans la théorie de Vergnaud. Mobiliser conjointement la théorie des champs conceptuels et l'approche instrumentale s'envisage donc en gardant ce couple sujet/situation comme invariant tout au long de nos analyses.

Dans les situations d'activité instrumentée intervient en plus un troisième pôle, l'instrument, central et spécifique de l'approche instrumentale de Rabardel. C'est pourquoi la section suivante lui est dédiée.

5.2 L'instrument

Nous détaillons le concept d'*instrument* au sens de Rabardel, qui est une entité mixte constituée d'un artefact et de schèmes d'utilisation.

5.2.1 L'artefact

Rabardel désigne par *artefact* « une catégorie générale neutre correspondant à toute chose produite ou transformée par l'homme dans une visée finalisée » (Rabardel, 2005, p.257). Il précise que le concept d'artefact ne réfère pas seulement à des objets matériels. Il inclut aussi des objets symboliques (Rabardel, 1995, p.3). Parmi les artefacts symboliques, le cas du langage nous intéresse particulièrement.

Nous trouvons une mention de *composantes artefactuelles*, au pluriel (Rabardel, 2005, p.256). Le pluriel est important, car il signifie que dans le cas d'artefacts assez complexes, tels des ordinateurs, ils peuvent être décomposés en composantes artefactuelles distinctes.

Le concept d'activité relativement requise

Selon Rabardel, l'artefact détermine une part de l'activité du sujet, mais non la totalité. Il est porteur à la fois de contraintes et de ressources pour l'action, d'où l'introduction du concept d'*activité relativement requise*. Rabardel identifie différents types de contraintes (Rabardel, 1995, p.5) :

- Les contraintes de modalités d'existence : il s'agit de maintenir l'artefact en état de pouvoir être utilisé. Un exemple dans notre contexte est de maintenir la connexion internet.
- Les contraintes de finalisation : elles réfèrent à ce qu'il est possible de réaliser avec l'artefact, et sur quels objets. Dans notre cas, les blocs disponibles contraignent certaines structures algorithmiques.
- Les contraintes de pré-structuration de l'action : elles se rapportent aux modalités de l'action requises par la structure de l'artefact. Par exemple dans notre contexte, la manipulation directe du robot virtuel n'est pas possible, il est requis d'utiliser les blocs d'action pour commander celui-ci.

Le concept de transparence opérative

Selon ses compétences, son but, et la situation d'action où il se trouve, le sujet a des besoins différents en *information* à propos de l'artefact. Celui-ci peut fonctionner sur le mode de la *boîte noire* (invisibilité du système technique) ou de la *boîte de verre* (système technique perceptible par l'utilisateur). Le concept de *transparence opérative* désigne « les propriétés caractéristiques de l'artefact, pertinentes pour l'action de l'utilisateur, ainsi que la manière dont l'artefact les rend accessibles, compréhensibles, voire perceptibles pour l'utilisateur » (Rabardel, 1995, p.151).

Nous convoquons ces concepts afin de questionner dans quelle mesure l'activité du sujet est contrainte et soutenue par les caractéristiques de l'EIAH utilisé pour les expérimentations.

5.2.2 Les schèmes d'utilisation

Nous retrouvons le concept de schème dans l'approche instrumentale de Rabardel. Contextualisés aux situations d'action instrumentée, les schèmes sont attachés aux artefacts et appelés schèmes d'utilisation.

Rabardel distingue trois types de schèmes d'utilisation : les schèmes d'usage, les schèmes d'action instrumentée et les schèmes d'activité collective instrumentée (Rabardel, 1995, p.4). Nous détaillons les deux premiers, qui se rapportent à la classe des situations d'activité instrumentée. Dans la mesure où l'activité du

sujet lors de nos expérimentations est individuelle, nous laissons le troisième de côté.

Un *schème d'usage* répond à la question « Qu'est-ce que je peux faire avec ça ? », « Comment ça fonctionne ? », alors qu'un *schème d'action instrumentée* répond à la question « Qu'est-ce que je veux faire muni de cet artefact ? ». Lors d'un schème d'usage, l'artefact a le statut d'objet de l'activité, alors que pour un schème d'action instrumentée, il a le statut de moyen de l'action.

Les schèmes d'action instrumentée incorporent à titre de constituant les schèmes d'usage. Nous retrouvons la hiérarchie d'un système de schèmes, déjà introduite pour la théorie des champs conceptuels (4.4).

Rabardel précise que « les deux dimensions de l'instrument, artefact et schème, sont associées l'une à l'autre, mais elles sont également dans une relation d'indépendance relative » (Rabardel, 1995, p.4). C'est cette indépendance relative de l'artefact et des schèmes d'utilisation qui rend possible l'adaptation du sujet aux situations (Rabardel, 2005).

5.2.3 Les interactions entre les trois pôles

Rabardel relève plusieurs types d'interactions au sein du système sujet-objet-instrument, parmi lesquelles les interactions entre le sujet et l'instrument, celles entre l'instrument et l'objet sur lequel il permet d'agir, et celles entre le sujet et l'objet médiatisées par l'instrument.

Plus particulièrement, Rabardel mentionne les activités instrumentées dont l'artefact est un ordinateur comme un cas particulier du point de vue des interactions entre les trois pôles : « Il arrive cependant qu'elle [l'interaction artefact-objet] ne soit pas explicitée, en particulier lorsque les situations sont caractérisées par une position interne de l'objet dans l'artefact (ordinateur, processus), c'est alors sur la relation à l'objet, médiée par l'artefact que l'accent est mis, tandis que, pour des raisons évidentes l'interaction directe sujet-objet disparaît. » (Rabardel, 1995, p.62).

5.3 Les genèses instrumentales

Selon Rabardel, l'accroissement des ressources du sujet se fait à travers des *genèses instrumentales*, qui sont de l'ordre de l'interaction du sujet avec son environnement et les artefacts qui s'y trouvent. L'étude de ces genèses instrumentales constitue « une approche développementale des instruments » (Rabardel, 1995, p.4). Dans notre contexte, nous nous demandons dans quelle mesure l'interaction du sujet avec l'EIAH accroît ses ressources pour réussir les

puzzles de programmation auxquels il est confronté et pour appréhender les notions algorithmiques sous-jacentes.

5.3.1 Activité productive et activité constructive

L'activité réalisé par le sujet revêt deux natures, *productive* et *constructive*. « L'*activité productive* est orientée vers l'atteinte des buts en situation ainsi que la configuration des situations. » (Rabardel, 2005, p.254). C'est la réalisation de la tâche dans la situation présente qui est visée (à mettre en regard avec la réussite au sens de Piaget (1974b)). Quant à l'*activité constructive*, elle « est orientée vers l'accroissement, le maintien, la reconfiguration des ressources du sujet pour l'activité productive à venir ». Elle « porte notamment sur la transformation, le développement et la mise en forme de ces organisateurs de l'activité que sont les schèmes. » (Rabardel, 2005, p.257). Elle perdure au-delà de l'action. (Rabardel, 2005, p.254).

Du point de vue temporel, l'activité productive s'inscrit plutôt dans le court terme alors que l'activité constructive s'inscrit dans la temporalité du développement du sujet qui relève du moyen et long terme. En ce sens, les genèses instrumentales sont constitutives de l'activité constructive.

5.3.2 Instrumentation et instrumentalisation

Lors des genèses instrumentales, un double processus est à l'œuvre.

D'une part, le processus d'*instrumentalisation* désigne l'utilisation d'un artefact pour atteindre un objectif particulier en conformant cet artefact à l'usage en situation. Les processus de sélection, de détournements, de transformation de l'artefact relèvent du processus d'instrumentalisation, processus qui peut aller jusqu'à la création intégrale d'un artefact. Le sujet projette alors sur cet artefact des schèmes construits par ailleurs. Rabardel donne l'exemple d'une clé anglaise utilisée comme marteau (Rabardel, 2005).

D'autre part, le processus d'*instrumentation* réfère à la constitution et à l'évolution des schèmes d'utilisation en relation avec les artefacts auxquels ils sont associés. Cette évolution se fait par assimilation et accommodation (Piaget, 1935). Ce processus est dirigé vers le sujet lui-même (Rabardel, 2005), qui accroît, organise, complexifie son système de schèmes et élargit les situations pour lesquelles ces schèmes sont mobilisables.

« Les deux processus contribuent solidairement à l'émergence et l'évolution des instruments, même si, selon les situations, l'un d'eux peut être plus développé, dominant, voire seul mis en œuvre. » (Rabardel, 1995, p.112)

5.3.3 L'activité représentative

« Les processus d'instrumentation et d'instrumentalisation par lesquels se constituent les composantes schème et artefact de l'instrument, impliquent, de la part du sujet, une activité représentative dont le rôle dans la structuration, le contrôle et la régulation des actions est essentiel » (Rabardel, 1995, p.118). Dans le cas des activités instrumentées, cette activité représentative consiste à se construire des *représentations pour l'action*, c'est à dire des représentations mentales nécessaires à la bonne conduite de l'action.

Il faut distinguer « la constitution initiale de la représentation (au sens de modèle mental ayant une permanence) de son utilisation. Les deux processus ne s'excluent évidemment pas l'un l'autre, la distinction entre construction et utilisation indique simplement le pôle dominant, l'objet principal de l'activité du sujet. La construction de la représentation participe de la genèse instrumentale, tandis que son utilisation participe de la mise en œuvre de l'instrument. » (Rabardel, 1995, p.121)

Rabardel met en correspondance les représentations pour l'action avec les invariants opératoires (concepts-en-acte et théorèmes-en-acte) de la théorie de Vergnaud (Rabardel, 1995, p.88). En ce sens, les représentations pour l'action sont partie intégrante du schème. En poursuivant l'analogie, l'activité constructive, de par les ressources pour l'action à venir qu'elle produit, peut être mise en correspondance avec la conceptualisation-en-acte au sens de Vergnaud, dans le cas où les concepts-en-acte en jeu se rapportent à un instrument.

Rabardel précise que « Les représentations des artefacts constitués en instruments par le sujet, appartiennent à la catégorie des représentations pour l'action. » (Rabardel, 1995, p.119). Pour le cas particulier de la programmation, Rabardel réfère aux travaux de Rogalski à propos de la représentation du dispositif informatique (Rogalski, 1988b). Rogalski montre que la construction de la représentation du dispositif informatique ne va pas de soi, qu'elle est une source de difficulté pour des élèves de seconde initiés à la programmation textuelle. En effet, ceux-ci ont tendance à considérer que la machine va se comporter comme ils le feraient.

5.3.4 Le langage en tant qu'artefact symbolique

Un langage est un artefact symbolique, dont l'apprentissage est d'ordre instrumental (Rabardel, 1995, p.37). Il a le statut d'intermédiaire entre le sujet et l'objet (Rabardel, 1995, p.61). Ce point nous intéresse particulièrement dans la mesure où l'activité de programmation suppose l'apprentissage d'un langage spécifique, qualifié de formel.

Le sujet construit le langage comme instrument dont la finalité est de trans-

mettre des informations ou/et d'accéder au sens d'informations reçues, c'est à dire de communiquer. Bruner, cité par Rabardel, parle de « construction de la signification » (Rabardel, 1995).

La construction de la signification est de l'ordre du schème d'usage. Se forger des représentations pour l'action consiste à construire la signification de chaque élément de langage et appréhender les règles de syntaxe. L'objet est alors l'élément de langage qui est constitué en instrument, celui-ci étant incorporé à titre de constituant au schème d'action instrumentée lorsque le sujet se sert de cet élément de langage pour communiquer (comprendre ou émettre des messages, dans une approche simplifiée).

Rabardel reprend la catégorisation de Inhelder et De Caprona (1992) qui distinguent deux types de schèmes. La construction de la signification fait partie des schèmes *présentatifs*, qui visent à comprendre la réalité, par distinction avec les schèmes *procéduraux* qui sont des suites d'actions dont le but est de réussir une tâche.

5.4 Le cas des enfants, sujets en développement

L'approche instrumentale permet d'étudier l'activité instrumentée des adultes mais aussi des enfants (Bationo Tillon & Rabardel, 2015). Les auteurs abordent les questions spécifiques qui se posent lorsque le sujet est un enfant. Une caractéristique des enfants est qu'ils sont en développement et que leur manière d'aborder les situations évolue de ce fait rapidement. Les jeunes sujets sont acteurs, de part leur développement, de la transformation des situations. Les situations de rupture, par exemple l'introduction d'un nouvel artefact dans une situation, est souvent une occasion de développement pour le jeune sujet.

Pour les enfants très jeunes, le chercheur peut ne pas pouvoir recourir au langage pour recueillir le point de vue du sujet. Les auteurs invitent alors à s'appuyer sur des observations répétées.

Bationo Tillon et Rabardel se posent la question des caractéristiques des instruments qui sont de nature à favoriser le développement. Du point de vue des concepteurs, il est nécessaire d'identifier l'activité constructive qui est visée et dont l'instrument est potentiellement porteur. Ces activités constructives potentielles s'inscrivent dans différents horizons temporels qu'il est nécessaire d'appréhender. Les auteurs prennent l'exemple des programmes scolaires qui identifient des situations et des étapes avec les compétences à acquérir associées.

5.5 Synthèse contextualisée

Au sein de la classe des activités instrumentées conceptualisée par Rabardel, nous nous situons dans le cas particulier des interactions humain-ordinateur, avec une position interne de l'objet de l'activité dans l'artefact. Cette position interne est essentielle, car elle octroie une double fonction à l'artefact, qui supporte à la fois l'objet de l'activité et les moyens de celle-ci.

De plus nous nous adressons à un public de jeunes sujets, identifié comme spécifique par Bationo Tillon et Rabardel (2015), notamment parce que leur manière de traiter les situations évolue rapidement, et nécessite une adaptation des artefacts et une identification des activités constructives en rapport avec ce développement.

En conséquence, nous contextualisons l'approche instrumentale aux particularités de l'initiation à la programmation par blocs d'un public de jeunes sujets de 7 à 15 ans.

Nous distinguons d'emblée deux composantes artefactuelles dans notre contexte : une composante artefactuelle matérielle, l'ordinateur, et une composante artefactuelle symbolique, le langage de programmation. Cette pluralité de composantes artefactuelles nous incite à envisager une étude instrumentale de l'EIAH, afin d'appréhender comment ces composantes sont appréhendées et constituées en instrument par de jeunes sujets. Nous retenons la préconisation d'identifier les potentielles activités constructives en jeu dans une situation. Nous visons donc de distinguer les schèmes d'usage qui se rapportent à la prise en main de l'EIAH et des schèmes d'action instrumentée qui mettent en jeu les premiers apprentissages en algorithmique. L'activité constructive, de par les ressources pour l'action à venir qu'elle produit, nous semble pouvoir être mise en relation avec la conceptualisation-en-acte au sens de Vergnaud. Nous étudierons comment genèses instrumentales (Rabardel) et genèses conceptuelles (Vergnaud) s'articulent.

Nous prendrons en compte les trois dimensions caractéristiques de la médiation propre à l'artefact, identifiées par Payne (1991) dans le cas des interactions homme-ordinateur (Rabardel, 1995, cité p.135) :

- Le dispositif détermine les tâches que le sujet est en mesure de mener. Dans notre contexte, nous questionnerons dans quelle mesure les paramétrages de l'EIAH (blocs mis à disposition, nombre de blocs disponibles) ont un aspect contraignant sur l'activité du sujet, notamment lors de l'introduction de la notion de boucle bornée. Nous mobiliserons à cette fin le concept d'activité relativement requise.
- Le sujet n'agit pas directement sur les objets de la tâche. Les actions sont exprimées dans un langage artificiel, dans notre contexte un langage de

programmation par blocs, qui sert d'intermédiaire. Quelles sont les genèses instrumentales attachées à la construction du langage de programmation comme instrument par de jeunes sujets ?

- Les actions de l'utilisateur doivent être articulées avec le fonctionnement du système (EIAH comme logiciel, mais aussi dispositif informatique en arrière-plan), ce qui implique pour l'utilisateur de s'en construire une représentation mentale. De manière similaire à ce qu'a réalisé Rogalski pour la programmation textuelle, il sera intéressant d'identifier la construction de représentations du dispositif informatique et les difficultés associées dans le cas de la prise en main d'un environnement de programmation par blocs par de jeunes sujets.

Il nous semble aussi intéressant d'analyser l'EIAH en termes de transparence opérative. En effet, un certain nombre d'aspects du système technique est rendu invisible dans un objectif de simplification vers de jeunes sujets. Qu'est-ce qui reste perceptible du dispositif informatique ? Est-ce qu'avoir masqué une part du fonctionnement du système entraîne des conséquences sur la conceptualisation ?

Enfin, nous retenons que les observations répétées sont un moyen pertinent d'investigation de l'appréhension des situations par des sujets jeunes en plus du recueil de leurs verbalisations, dans le but de dégager des invariants dans la conduite de ces situations, c'est à dire des schèmes.

Chapitre 6

Approfondissement de la problématique

Sommaire du présent chapitre

L'objectif général de ce travail est d'étudier la conceptualisation des notions de base en algorithmique par les élèves de 7 à 15 ans, à travers notamment l'analyse de traces d'interaction avec un environnement de programmation par blocs lors de la résolution de puzzles.

Dans les premiers chapitres, nous avons précisé le contexte de cette étude en resserrant progressivement nos investigations respectivement sur la pensée informatique, la programmation par blocs, la résolution de puzzles de programmation et enfin le concept de motif, qui, à la suite de nos expérimentations préparatoires, est l'objet principal de notre travail de recherche. Nous avons exprimé une première version de notre problématique à la fin de l'introduction de ce document :

Dans quelle mesure l'identification de motifs est-elle un élément crucial lors de l'initiation à la programmation dans un environnement de programmation par blocs, en particulier lors de l'apprentissage de la boucle ?

Dans le présent chapitre, nous précisons et approfondissons la problématique de cette étude en la reformulant dans les termes des cadres d'analyse que nous avons introduits dans les chapitres précédents : théorie des champs conceptuels de Vergnaud (1991) et approche instrumentale de Rabardel (1995). D'une part, nous contextualisons à notre étude les concepts de ces cadres d'analyse sur lesquels nous nous appuyons pour interpréter les résultats de nos expérimentations. D'autre part, nous déclinons notre problématique en définissant des questions de recherche.

Une didactique de l'informatique a été élaborée dans les années 1970 et 1980, notamment en France à travers ce qu'on a appelé la *psychologie de la programmation*. Parmi les travaux de cette période, certains se sont appuyés sur la théorie des champs conceptuels dans le but d'étudier l'apprentissage des notions algorithmiques de base par un public novice. Un champ conceptuel de *l'alphabétisation informatique* a été identifié dans les années 1980 pour la programmation textuelle au niveau du lycée (Rogalski & Vergnaud, 1987).

Nous nous situons dans la continuité de ces travaux. Notre but est de mener une étude du même type pour la programmation par blocs au niveau de l'école obligatoire.

De manière constructive, nous visons à identifier puis documenter un champ conceptuel relatif aux notions de base en algorithmique en nous appuyant sur la théorie des champs conceptuels de Vergnaud. Nous nous concentrons plus particulièrement sur une des notions algorithmiques, la notion de boucle, qui est étroitement liée à notre objet de recherche principal : le concept de motif. Nous avons défini celui-ci dans le chapitre 3 comme une entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles.

6.1 Champs conceptuels

Notre approche didactique consiste à nous questionner sur ce qu'implique l'apprentissage des notions algorithmiques de base du point de vue de la conceptualisation. La théorie des champs conceptuels propose une approche théorique pour comprendre comment les individus construisent et organisent leurs connaissances dans un domaine donné. Lorsqu'il s'agit d'analyser les premiers apprentissages en algorithmique, nous pensons que cette théorie offre des outils conceptuels pertinents. C'est pourquoi nous avons fait le choix de la mobiliser pour notre travail.

- Premièrement, l'initiation à la programmation informatique implique une pluralité de concepts imbriqués et interconnectés (instruction, séquence, boucle, condition, variable, fonction). La théorie des champs conceptuels donne des clés pour analyser comment ces concepts sont structurés hiérarchiquement et repérer comment des élèves d'école élémentaire et de collège peuvent construire cette structure hiérarchique de concepts au fil de confrontations avec des puzzles dans un environnement de programmation par blocs.
- Deuxièmement, la théorie des champs conceptuels met en avant l'idée de progression des acquisitions cognitives sur un temps long, où les sujets développent la maîtrise de concepts de manière graduelle en passant par différentes étapes. Dans le contexte de l'initiation à la programmation dans le laps de temps long que constitue la période de l'école obligatoire, nous espérons nous appuyer sur cette théorie pour identifier des paliers dans le développement des compétences en programmation, de la compréhension des bases à la maîtrise de concepts plus avancés.
- Enfin, la théorie des champs conceptuels aborde également les obstacles épistémologiques, c'est-à-dire les difficultés particulières que les apprenants peuvent rencontrer lorsqu'ils tentent de maîtriser et d'intégrer certains concepts. Dans notre contexte, maîtriser un concept tel que la boucle peut s'avérer ardu pour de jeunes élèves. La théorie des champs conceptuels, notamment à travers les concepts d'invariants opératoires, apporte des outils conceptuels pour identifier et documenter ces obstacles et ainsi être en mesure de proposer des pistes pour les surmonter.

Mener ce travail pour l'ensemble des concepts algorithmiques de base serait possible, mais s'avère trop ambitieux dans le cadre d'une thèse. C'est pourquoi, dans la continuité de nos premiers travaux, nous limitons nos investigations aux concepts algorithmiques en lien avec notre objet de recherche principal, le concept de motif.

Nous visons donc à caractériser un champ conceptuel dans lequel l'un des concepts est celui de motif. Une question se pose sur le périmètre de ce champ conceptuel. D'un côté, Vergnaud recommande de définir des champs conceptuels larges, englobants, comme l'est celui attaché à l'*alphabétisation informatique* (Rogalski & Vergnaud, 1987). D'un autre côté, nous trouvons dans la littérature des champs conceptuels attachés à des concepts algorithmiques particuliers : variable (Lagrange & Rogalski, 2017), itération (El Rouadi, 1999). Ces deux approches sont complémentaires et non antagonistes si on considère le caractère imbriqué des champs conceptuels. Pour notre part, le périmètre de nos investigations est délimité par le type de langage que nous étudions, les langages par blocs, et l'âge des sujets, 7 à 15 ans. À l'intérieur de ce périmètre et selon la granularité et le focus de nos analyses par rapport au concept de motif, nous pourrons considérer un champ conceptuel centré sur la boucle, ou inclure celui-ci dans un champ conceptuel plus large des notions de base en algorithmique.

Dans la mesure où nous nous plaçons dans le cadre d'analyse de la théorie des champs conceptuels, le couple sujet/situation constitue notre unité d'analyse. Nous parlerons donc de sujets dans notre propos. Le sujet peut être un élève s'il se trouve dans le cadre scolaire, ce qui sera majoritairement le cas. Nous faisons donc abstraction dans nos analyses des éventuelles interventions de personnes tierces (enseignants, autres élèves, expérimentatrice,..) et ne prenons en compte que les traces d'interaction des sujets avec l'environnement de programmation. Cependant, pour un échantillon restreint de sujets, ces traces d'interaction sont complétées par des données qualitatives captées par enregistrement vidéo.

La théorie des champs conceptuels admet deux axes d'étude. Un premier axe est porté par les situations. Vergnaud invite à débuter par une analyse épistémologique des situations. Quelles situations nous permettent de nous focaliser sur l'identification de motif lors d'une activité de programmation ? Après avoir explicité le choix des puzzles de programmation pour notre recherche et caractérisé les situations qui requièrent l'utilisation d'une boucle, nous en mènerons une analyse épistémologique comme le préconise Vergnaud. Nous chercherons à décomposer la programmation d'une boucle en tâches plus élémentaires, afin de mieux appréhender comment est impliqué le concept de motif lors de ces situations de programmation.

Le second axe consiste à étudier la conceptualisation-en-acte des sujets, c'est à dire à inférer la conceptualisation du sujet à partir de son activité. Ce qui nous amène à nous poser la question de la manière dont se manifeste l'activité du sujet lors de la résolution de puzzles dans un environnement de programmation par blocs.

Que sommes-nous en mesure de collecter comme traces d'interaction avec l'environnement de programmation qui nous renseignent sur l'activité du sujet ?

Comment ces traces rendent-elles compte de la conceptualisation-en-acte du sujet ? Dans quelle mesure les traces collectées sont-elles suffisantes pour établir des conclusions robustes ?

Selon Vergnaud, l'activité du sujet est caractérisée par des schèmes, un schème étant une organisation invariante de la conduite. Quels sont les schèmes impliqués dans les situations de programmation définies précédemment ? Si plusieurs schèmes sont impliqués, comment caractériser le système hiérarchique de schèmes ? Parmi ces schèmes, lesquels sont en lien avec le concept de motif ? Nous chercherons à caractériser le ou les schèmes identifiés en utilisant la définition analytique de Vergnaud, en documentant notamment les règles d'action, et les invariants opératoires (concepts-en-acte et théorèmes-en-acte).

Par ailleurs, dans une perspective de généralisation de notre travail, nous discuterons à quel point ces schèmes sont spécifiques au contexte des puzzles de programmation, à l'environnement de programmation par blocs utilisé. Les schèmes identifiés sont-ils aussi mobilisés dans d'autres situations, dans d'autres domaines ? Sont-ils déjà documentés dans la littérature ? Sous quelle forme ?

6.2 Nécessité de l'approche instrumentale

Une différence majeure entre notre étude et les travaux menés dans les années 1980 se situe au niveau de l'environnement de programmation. Les travaux des années 1980 avaient pour objet l'initiation à la programmation avec des langages textuels. Notre travail concerne les environnements de programmation par blocs, qui sont apparus ultérieurement. Nous avons décrit les spécificités de ces environnements de programmation par blocs dans le chapitre 2. Nous nous demandons dans quelle mesure et de quelle manière les spécificités d'un environnement de programmation par blocs contraignent la confrontation aux concepts algorithmiques.

L'activité de programmation au sein des environnements de programmation par blocs constitue une activité instrumentée au sens de Rabardel. Afin d'être en mesure de résoudre le problème qui lui est soumis, le sujet doit prendre en main l'environnement de programmation. Il nous semble donc nécessaire de mener une analyse de l'activité instrumentée, afin de distinguer ce qui, dans l'activité, relève de la maîtrise conceptuelle et ce qui relève de la maîtrise instrumentale. Pour cela, il nous faut définir les instruments et les genèses instrumentales attachés à l'environnement de programmation que nous utilisons pour nos expérimentations. En d'autres termes, nous nous demanderons, parmi les schèmes identifiés, lesquels sont des schèmes d'usage en précisant leur objet. Pour ce qui concerne les schèmes d'action instrumentée, nous déterminerons l'instrument et à quelle conceptualisation ils sont attachés.

Nous considérons les environnements de programmation par blocs comme des EIAH. En effet, ils sont destinés à l'apprentissage de notions de base de la programmation par un public novice. Dans l'environnement que nous utilisons, différents paramétrages sont possibles. Le concepteur des puzzles agit sur la situation à travers le choix des paramètres. Bien que l'unité d'analyse de notre étude soit le couple sujet / situation instrumentée, nous nous demandons quelle part de l'activité du sujet est dépendante du paramétrage de l'EIAH par le concepteur, dans quelle mesure le paramétrage de l'EIAH contraint le traitement de la situation par le sujet.

Un exemple de paramétrage que nous développerons est la contrainte en nombre de blocs pour l'édition du programme. Ce paramétrage de l'EIAH vise à contraindre le passage de la séquence d'instructions à la boucle. Quels sont les avantages et les inconvénients de cette démarche du point de vue des genèses conceptuelles et instrumentales ?

6.3 Reformulation de la problématique

Comme synthèse du présent chapitre, nous proposons une reformulation de notre problématique. Nous nous plaçons dans le contexte de l'initiation à la programmation de sujets de 7 à 15 ans, ce qui correspond à l'âge de l'école obligatoire en France et nous adressons la problématique suivante :

> Quelles sont les genèses conceptuelles en lien avec le concept de motif lors de l'initiation à la programmation dans un environnement de programmation par blocs, en particulier lors de la résolution de puzzles qui requièrent l'utilisation d'une boucle bornée ?

Pour adresser cette problématique, nous répondrons aux questions de recherche suivantes en nous plaçant dans le cas de la résolution de puzzles de programmation qui requièrent l'utilisation d'une boucle bornée.

- Quels indicateurs rendent compte de l'expertise du sujet dans la résolution de puzzles de programmation ? Peut-on définir des profils de résolution de ces puzzles en lien avec le cadre théorique de la théorie des champs conceptuels ?
- Quelles classes de situations attachées au concept de motif peut-on définir ?
- Quelles conceptualisations-en-acte peut-on identifier ? Quelles sont les erreurs récurrentes qui révèlent une conceptualisation défaillante ?

Dans notre contexte, accéder aux genèses conceptuelles du sujet ne nous semble réalisable que lorsque celui-ci a pris en main l'environnement de programmation. C'est pourquoi, nous aborderons aussi la question suivante avec une approche instrumentale :

- Quels sont les schèmes d'usage attachés à l'environnement de programmation par blocs qu'il est nécessaire de construire afin d'être en mesure de résoudre les puzzles proposés ?

Ainsi, comme décrit précédemment, nous mobiliserons la théorie des champs conceptuels de manière concomitante à des méthodes d'analyse de traces pour ce que cette démarche pourra produire comme résultats sur la connaissance des processus cognitifs des élèves. Nous compléterons par une réflexion sur l'approche que nous avons choisie d'utiliser : quels sont les apports réciproques des méthodes d'analyse de traces et de la théorie des champs conceptuels sur un plan théorique ? Comment ce questionnement s'inscrit-il dans le champ de la recherche en EIAH ?

Troisième partie

Méthodologie et cadre expérimental

Dans cette troisième partie, nous détaillons la méthodologie employée pour cette thèse. Notre protocole expérimental est adossé au concours national de programmation Algoréa que nous présentons dans le chapitre 7. Il repose pour une large part sur la collecte de traces lors de l'activité des sujets dans l'environnement de programmation (chapitre 8), puis sur un processus d'analyse de ces traces d'interaction associant traitement automatisé et travail d'expert ancré dans le cadre d'analyse de la théorie des champs conceptuels (chapitre 9).

Chapitre 7

L'environnement de programmation Algoréa, support de notre dispositif expérimental

Sommaire du présent chapitre

Dans ce chapitre, nous décrivons en détails l'environnement de programmation du concours Algoréa que nous avons choisi comme support de la partie expérimentale de notre étude. Il s'agit d'un environnement de programmation par blocs. Il en possède les caractéristiques, décrites dans le chapitre 2 et que nous contextualisons dans la première section de ce chapitre. Algoréa est l'environnement de programmation que nous avons déjà utilisé lors de nos expérimentations en amont de ce travail doctoral. Il est par ailleurs utilisé pour le concours de programmation en ligne Algoréa, que nous présentons brièvement dans une deuxième section, avant d'expliciter notre choix d'utiliser cet environnement de programmation et d'adosser notre protocole expérimental au concours Algoréa.

7.1 Les caractéristiques de l'environnement de programmation Algoréa

L'environnement de programmation Algoréa est développé par l'association France-ioi [1]. Cet environnement supporte une approche de la programmation par résolution de problème, ou approche algorithmique, sous forme de puzzles dans des micromondes au sens de Pelánek et Effenberger (2020). Pelánek et Effenberger ont catégorisé les fonctionnalités de ces environnements de programmation. Pour analyser l'environnement de programmation Algoréa, nous nous appuyons sur leurs critères : type de micromonde, label des blocs, manière d'accrocher les blocs, manière de mettre les blocs à disposition, fixation de limites, notions de programmation pouvant être abordées.

La programmation dans l'environnement Algoréa s'inscrit dans le paradigme de la programmation dite séquentielle ou impérative. Toutes les notions citées par Pelánek et Effenberger (2020, p.7) sont susceptibles d'être abordées dans l'environnement Algoréa : séquence d'instructions, répétition, structure conditionnelle, variable.

Les notions prises en charges dans l'environnement de programmation Algoréa recouvrent celles du programme scolaire de l'école élémentaire et du collège (1.2.1).

L'environnement Algoréa est assez générique dans le sens où les trois types de micromonde que Pelánek et Effenberger ont répertoriés y sont disponibles : *Robot sur une grille*, *Tortue graphique* et *Dessin de formes*. Il supporte en plus d'autres micromondes, non répertoriés par les auteurs, comme le micromonde *Quickpi* dédié à la simulation d'objets connectés.

1. Site de l'association France-ioi : https://www.france-ioi.org/

Nous détaillons l'interface pour le micromonde de type *Robot sur une grille* que nous mobilisons dans l'étude. Un micromonde de ce type est déterminé dans l'environnement Algoréa par le choix d'un *contexte* (voir Figure 7.2), qui comprend :

- un design visuel
- un type de robot (orienté ou non) et le système d'orientation associé
- une mission à faire réaliser au robot et une fonction de validation associée
- des commandes d'action
- des éléments de la grille : objets sur lesquels le robot devra agir, zones de dépôt, case à atteindre, obstacles, éléments de décor
- éventuellement des caractéristiques supplémentaires propres au micromonde (exemple : la gravité)

7.1.1 L'organisation de l'interface

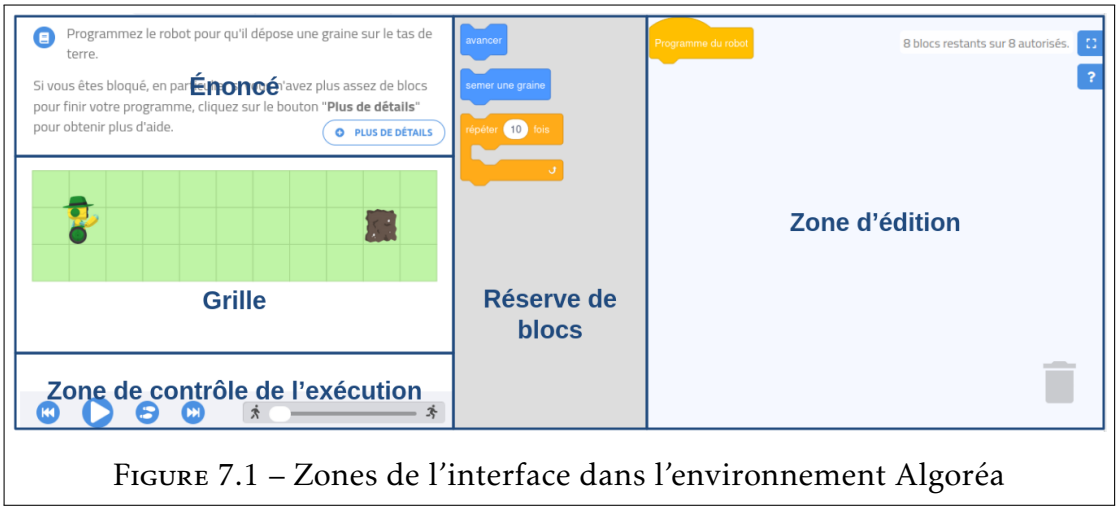
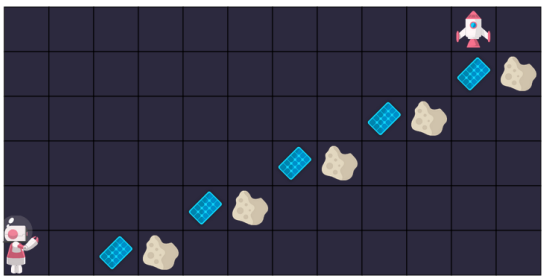


FIGURE 7.1 – Zones de l'interface dans l'environnement Algoréa

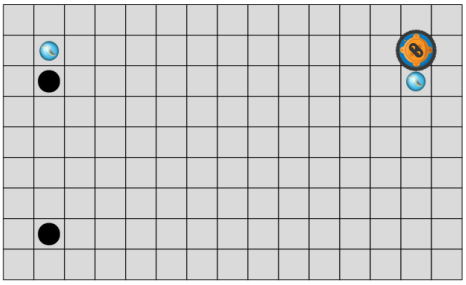
L'interface est toujours organisée de la même manière quel que soit le puzzle considéré. Elle comporte cinq zones, identifiées sur la figure 7.1, et que nous analysons en considérant les quatre concepts de l'informatique (DOWEK, 2011) et les critères de PELÁNEK et EFFENBERGER (2020).

7.1.1.1 L'énoncé

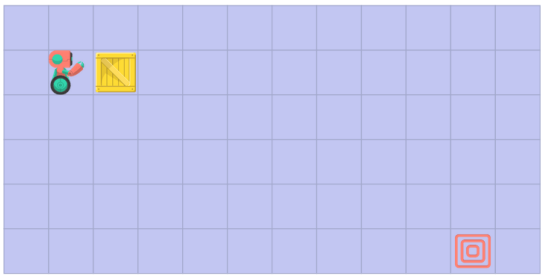
L'énoncé du problème est placé dans la zone en haut à gauche (Figure 7.1). Cet énoncé est court et redondant d'un problème à l'autre pour un même contexte. Il



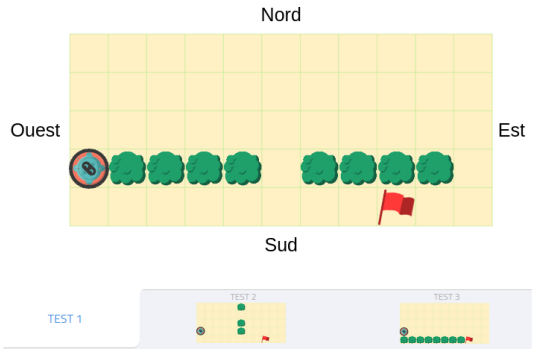
Grille n°1 - Contexte *Dans l'espace*
Tâche : programmez le robot pour qu'il ramasse les objets puis rejoigne la fusée.



Grille n°2 - Contexte *Ranger les billes*
Tâche : programmez le robot pour qu'il ramasse chaque bille et la dépose dans un trou.



Grille n°3 - Contexte *Sokoban*
Tâche : programmez le robot pour qu'il pousse la caisse sur la case marquée.



Grille n°4 - Contexte *Suivre le parcours*
Tâche : programmez le robot pour qu'il atteigne le drapeau.

FIGURE 7.2 – Exemples de grilles et d'énoncés pour le micromonde *Robot sur une grille*

définit la tâche assignée à l'utilisateur, qui est toujours de la forme « Programmez le robot pour qu'il... » (Figure 7.2). Le *faire faire* propre à la programmation est explicite dans chaque énoncé.

Des informations supplémentaires peuvent être accessibles en cliquant sur un bouton « Plus de détails » placé en-dessous de l'énoncé. Dans certains cas, ces informations complémentaires correspondent à un tutoriel qui introduit la notion algorithmique en jeu.

7.1.1.2 La grille

La zone en-dessous de l'énoncé est dédiée à une grille en deux dimensions. Les dimensions de la grille varient d'un problème à l'autre. Un robot virtuel, que nous détaillerons ultérieurement, est toujours présent sur cette grille. D'autres

éléments peuvent aussi se trouver sur la grille suivant le contexte et les besoins de la tâche à faire réaliser à ce robot. La figure 7.2 montrent quelques exemples représentatifs.

- Certains éléments sont des objets que le robot virtuel a pour mission de ramasser (Figure 7.2 : objets pour la grille n°1, billes pour la grille n°2). Pelánek et Effenberger parlent de *tokens*. Suivant les contextes, le robot peut porter un seul objet à la fois, ou plusieurs. Par exemple, dans le contexte *Ranger les billes*, le robot doit collecter une bille à la fois (Figure 7.2, grille n°2). D'autres objets ne sont pas collectés mais poussés par le robot virtuel. C'est le cas des caisses dans le contexte *Sokoban* (Figure 7.2, grille n°3).
- Certaines cases sont des zones de dépôt pour les objets collectés ou poussés. Par exemple, dans le contexte *Ranger les billes*, les cases avec un disque noir représentent un trou. Ce sont les cases où les billes doivent être déposées. Chaque trou ne peut contenir qu'une seule bille (Figure 7.2, grille n°2).
- Dans certains contextes, le robot doit atteindre une case cible qui est repérée par un élément visuel. Dans le contexte *Suivre le parcours*, la case cible contient un drapeau (Figure 7.2, grille n°4). Dans le contexte *Dans l'espace*, elle contient une fusée (Figure 7.2, grille n°1).
- Certains éléments, lorsqu'ils sont présents sur une case, rendent celle-ci inaccessible au robot. Ils constituent des obstacles (Figure 7.2 : cases contenant un astéroïde pour la grille n°1, cases contenant un buisson pour la grille n°4).

Pour certains problèmes, le programme doit fonctionner sur plusieurs grilles différentes. Cette fonctionnalité est mentionnée par Pelánek et Effenberger sans qu'elle constitue un critère de la catégorisation. Dans ce cas, les différentes grilles sont accessibles par des onglets en bas de la zone de la grille (Figure 7.2, grille n°4). Ces différentes grilles réfèrent au concept d'information, dans le sens où elles constituent les données d'entrée qui vont être traitées par le programme de l'utilisateur.

Pour conclure, nous insistons sur le rôle central de la grille dans les micromonde de type *Robot sur une grille*. En effet, la grille comporte toutes les informations dont l'utilisateur a besoin pour construire un programme qui résout le problème. C'est aussi sur cette grille que l'utilisateur visualise l'exécution du programme qu'il a conçu.

7.1.1.3 La réserve de blocs

La réserve de blocs contient les commandes disponibles pour la résolution du puzzle. Elle est à mettre en correspondance avec le concept de langage. Parmi

les critères de Pelánek et Effenberger, le label des blocs et la manière de mettre les blocs à disposition correspondent à cette zone de l'interface.

L'environnement de programmation Algoréa est paramétrable pour supporter trois langages différents : deux langages visuels (Blockly et Scratch) et un langage textuel (Python). Pour les deux langages par blocs, le label des blocs est textuel et la langue est paramétrable (anglais, français et quelques autres).

Types de commande Parmi les commandes disponibles, nous distinguons deux types. D'une part les commandes attachées aux concepts algorithmiques correspondent à des mots clés du langage de programmation et sont communes à tous les contextes (Figure 7.3). D'autre part, nous avons les commandes de déplacements, qui sont attachées au système d'orientation du robot, et les commandes spécifiques à un contexte (commandes d'action élémentaire, capteurs).

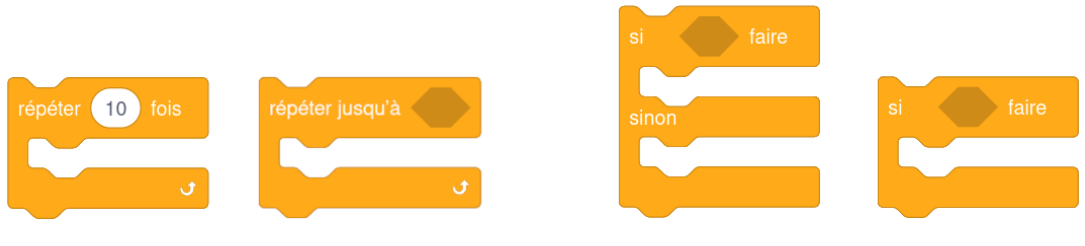


Figure 7.3 – Exemples de blocs attachés à des notions algorithmiques

Du point de vue informatique, ces commandes sont en fait des fonctions pré-programmées que l'utilisateur appelle lorsqu'il utilise le bloc correspondant dans son programme. Le système calcule alors l'évolution de variables en arrière-plan et en rend compte visuellement à l'utilisateur par une évolution de l'état de la grille (position et orientation du robot, présence ou absence de certains éléments sur la grille).

Mise à disposition des commandes Pelánek et Effenberger abordent le format de la mise à disposition de ces commandes à l'utilisateur. Les deux alternatives mentionnées, à plat et rangement dans un menu thématique, sont disponibles dans l'environnement Algoréa (Figure 7.4).

Le rangement à plat est plus adapté aux débutants, lorsqu'un très petit nombre de blocs est disponible. Le rangement en menu s'adresse à des utilisateurs plus avancés, lorsqu'un nombre important de blocs disponibles différents

À plat Menu thématique : une seule catégorie visible à la fois

FIGURE 7.4 – Options de mise à disposition des commandes à l'utilisateur

rend pénible le fait d'utiliser la barre de défilement pour atteindre un bloc. Ce mode de mise à disposition des commandes introduit un élément de complexité dans le sens où une seule catégorie de blocs est visible à la fois. Cela demande à l'utilisateur de se faire une représentation de la catégorie de chaque bloc pour savoir le retrouver rapidement, sans surcharge cognitive liée à cette tâche.

7.1.1.4 La zone d'édition

La zone d'édition est celle dans laquelle l'utilisateur manipule les blocs pour concevoir son programme (Figure 7.1). Cette zone est donc associée au concept d'algorithme. Dans les micromondes de type *Robot sur une grille*, l'algorithme est la suite d'actions à faire réaliser au robot virtuel. La description de cet algorithme est exprimée dans le langage de programmation par un agencement de blocs accrochés entre eux et au bloc initial *Programme du robot*.

En haut à droite de la zone d'édition figure le nombre de blocs disponibles pour résoudre le problème, ainsi que le nombre de blocs restant.

7.1.1.5 La zone de contrôle de l'exécution

La dernière zone de l'interface contient les boutons de contrôle de l'exécution (Figure 7.5).

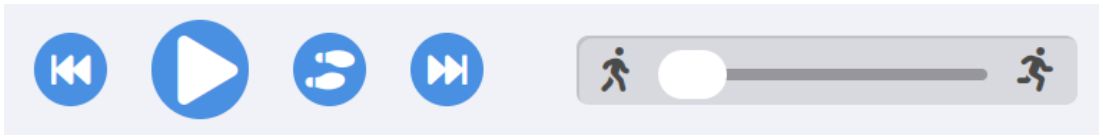


FIGURE 7.5 – Boutons de contrôle de l'exécution

Le bouton *play* permet à l'utilisateur de lancer l'exécution du programme qu'il a conçu. Des boutons complémentaires sont disponibles, dont un mode d'exécution *pas à pas* sur lequel il nous semble important d'attirer l'attention du lecteur. En effet, le mode *pas à pas*, qui consiste à exécuter le programme instruction par instruction, a pour finalité de permettre à l'utilisateur de localiser et détecter plus facilement les erreurs commises. Dans cet environnement de programmation destiné à des débutants, le mode *pas à pas* constitue une ressource pour les utilisateurs dans l'analyse et la correction de leurs programmes.

Lors de chaque clic sur le bouton *play*, le programme de l'utilisateur est évalué automatiquement par le système. Cette évaluation se fait sur la base d'états intermédiaires valides de la grille et d'un état final défini. Un *feedback* est renvoyé aussitôt à l'utilisateur, lui indiquant si son programme est valide, ou, dans le cas contraire, lui fournissant un indice sur la nature de son erreur. La validation automatique par le système est attachée à la définition que PELÁNEK et EFFENBERGER donnent d'un puzzle. Cependant, les auteurs ne détaillent pas les fonctions de contrôle de l'exécution.

7.1.2 Le robot virtuel

Le robot virtuel est l'élément central des micromondes de type *Robot sur une grille*. Il est à mettre en lien avec le concept de machine, dont il est une métaphore. En effet, aux yeux du jeune utilisateur, c'est le robot virtuel qui exécute son programme, en lieu et place du processeur de l'ordinateur. Cette métaphore permet une visualisation de l'exécution du programme.

Dans l'environnement Algoréa, trois systèmes d'orientation sont disponibles pour les robots virtuels. Chaque système d'orientation est associé à un robot d'aspect visuel spécifique. Des robots non orientés sont utilisés lorsque le système d'orientation est absolu alors que des robots orientés le sont dans le cas d'un système d'orientation relative ou d'un système que nous appelons hybride. Nous décrivons ces trois systèmes d'orientation.

FIGURE 7.6 – Exemple de robot et commandes de déplacement en orientation absolue

La figure 7.6 montre les commandes de déplacement associées au système d'orientation absolue (ou allocentrée), ainsi qu'un exemple de robot non orienté. Lorsque le puzzle est conçu avec ce système d'orientation, la grille est vue de dessus et les points cardinaux sont indiqués sur les côtés de la grille pour faciliter le repérage (exemple sur la figure 7.2, grille n°4).

FIGURE 7.7 – Exemple de robot et commandes de déplacement en orientation relative

La figure 7.7 montre les commandes de déplacement associées au système d'orientation relative (ou autocentrée), ainsi qu'un exemple de robot orienté. Comme pour le système d'orientation absolue, la grille est vue de dessus en orientation relative. Cependant, cette vue de dessus induit une distorsion de la perspective car le robot, lui, est vu de côté. C'est un défaut de l'interface Algoréa susceptible de poser des problèmes de prise en main.

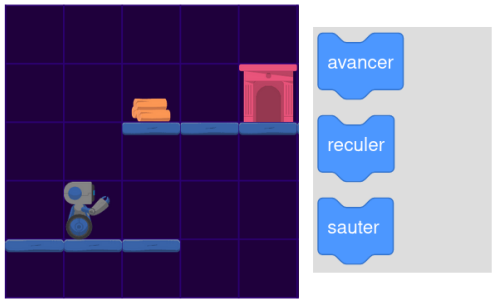


FIGURE 7.8 – Exemple de robot et commandes de déplacement en orientation hybride

Dans un troisième système d'orientation, dit *hybride*, la grille est vue de côté (Figure 7.8). Le robot est orienté mais le système d'orientation est absolu, dans le sens où il n'y a pas de changement d'orientation du robot, qui peut seulement avancer, reculer et sauter. Une caractéristique de ce micromonde est qu'il simule la gravité. Le déplacement vers le bas n'est donc pas disponible, le robot tombant automatiquement lorsqu'il ne repose pas sur une plateforme.

Les commandes de déplacement font partie des fonctions pré-programmées

fournies à l'utilisateur. Du point de vue informatique, ces fonctions font évoluer la valeur de variables qui représentent les coordonnées du robot sur la grille, et son orientation pour l'orientation relative. Lors de l'exécution du programme, le système calcule l'évolution de la valeur de ces variables, et en rend compte visuellement par l'évolution de la position du robot sur la grille, évolution qui est perçue comme un déplacement par l'utilisateur.

7.1.3 Synthèse

L'analyse des caractéristiques de l'environnement de programmation Algoréa au regard des critères de Pelánek et Effenberger (2020) montre que cet environnement est à la fois assez prototypique des environnements de programmation conçus pour la résolution de puzzles, et à la fois assez adaptable, car offrant de nombreuses possibilités de paramétrage.

7.2 Le concours national de programmation Algoréa

L'environnement de programmation décrit dans la section précédente est utilisé dans le cadre d'un concours national de programmation informatique, le concours Algoréa [2].

7.2.1 Présentation du concours, de son contexte, de ses objectifs et enjeux

Le concours Algoréa est un concours de programmation informatique en ligne organisé par France-ioi, qui répond au double objectif de l'association. D'une part, il vise à accompagner l'apprentissage des notions de base en algorithmique pour un très large public, essentiellement dans le cadre scolaire. D'autre part, il vise à identifier les jeunes passionnés par l'informatique afin de les amener vers l'excellence.

À sa création en 2013, année où l'Académie des sciences publie son rapport intitulé « L'enseignement de l'informatique en France. Il est urgent de ne plus attendre. » (Berry et al., 2013), le concours Algoréa attire seulement une centaine de participants, utilisateurs réguliers du site france-ioi.org pour la plupart. La programmation informatique est une activité peu répandue, qui est quasiment absente des programmes scolaires et souffre de représentations de trop grande complexité pour le grand public. À la même période, le concours Castor [3],

2. Site du concours Algoréa : https://algorea.org
3. Site du Concours Castor, co-organisé par France-ioi, l'INRIA et l'ENS Paris-Saclay : https://castor-informatique.fr/

concours de découverte de l'informatique et des sciences du numérique co-organisé par cette même association, rassemble déjà 170 000 participants.

Le concours Algoréa est volontairement accessible sans aucun pré-requis en programmation informatique, afin de lever l'obstacle de cette représentation de la programmation informatique comme une activité difficile. Il est ouvert pour une très large tranche d'âge, du CM1 à la terminale, de manière transversale par rapport aux curricula scolaires, afin d'accompagner chaque élève quel que soit son niveau en programmation et son rythme de progression.

La mise en lien avec le concours Castor réalisée en 2014 et l'effet du retour de la programmation informatique dans les programmes scolaires à partir de 2016 ont provoqué une progression exponentielle de la participation. Le concours Algoréa, qui comptait une centaine de participants en 2013, en compte désormais plus de 250 000 par an.

7.2.2 Organisation du concours

Le concours Algoréa, organisé annuellement, se déroule en deux phases (Figure 7.9). La première phase comprend trois tours étalés entre janvier et mai, qui sont ouverts à tous dans une perspective d'accompagnement de l'apprentissage, sans aucune sélection. La seconde phase, dite phase finale, est réservée aux participants qui ont atteint un certain niveau en algorithmique à l'issue de la première phase. Cette seconde phase vise à identifier les jeunes qui participeront à la préparation des compétitions internationales d'informatique l'année suivante. Dans le cadre de cette recherche, nous nous intéressons seulement à la première phase du concours Algoréa.

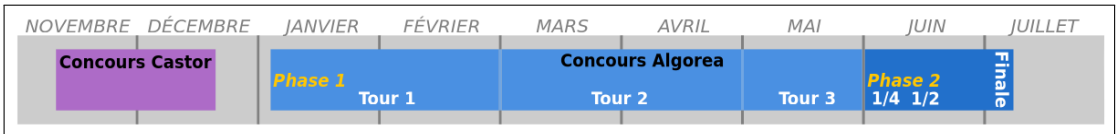


FIGURE 7.9 – Chronologie du concours Algoréa

Bien que l'organisation du concours ne soit pas sous la tutelle de l'institution scolaire, celle-ci a été pensée pour s'insérer dans le contexte scolaire. Les épreuves sont disponibles en ligne, sans autre installation qu'un navigateur. La procédure d'inscription a été étudiée pour ne pas être chronophage pour les enseignants, tout en leur fournissant du *feedback* sur l'activité de leur classe. Les épreuves ont lieu en temps limité, chaque épreuve durant 45 minutes, soit approximativement la durée d'un cours dans le secondaire. Plusieurs langages de programmation (Scratch, Blockly, Python) sont disponibles, afin de s'adapter aux choix effectués par l'institution scolaire pour les différents cycles et classes :

le langage de programmation par blocs est préconisé à l'école élémentaire et au collège. Au lycée, c'est le langage textuel Python qui est recommandé.

7.2.3 Progression pédagogique

Le concours Algoréa cible précisément l'apprentissage des notions de base en algorithmique, que l'on retrouve désormais dans tous les programmes scolaires de l'école élémentaire au lycée. L'ensemble du concours se présente sous la forme d'une série de puzzles de programmation, à résoudre dans une interface toujours organisée de la même façon, ce qui lui donne une cohérence globale : quel que soit le niveau de pratique, quel que soit le langage de programmation choisi, il s'agit de concevoir un programme pour résoudre un problème posé, en contrôlant un robot virtuel.

Bien que l'existence du concours Algoréa soit antérieure à la réintroduction de l'informatique dans les curricula scolaires à la rentrée 2016, l'objectif du concours est en adéquation avec les programmes scolaires des cycles 3 et 4 (1.2.1). L'extrait du programme de cycle 3 suivant mentionne directement le contexte de déplacement d'un robot que l'on retrouve dans les puzzles du concours Algoréa : « Une initiation à la programmation est faite à l'occasion notamment d'activités de repérage ou de déplacement (programmer les déplacements d'un robot ou ceux d'un personnage sur un écran) ». Pour le cycle 4, la compétence attendue en fin de cycle d'« Écrire, mettre au point (tester, corriger) et exécuter un programme en réponse à un problème donné. » décrit exactement l'activité de l'utilisateur lors de la résolution des puzzles du concours Algoréa.

La figure 7.10 montre la progression dans les notions algorithmiques établie pour le concours Algoréa. En plus d'être compatible avec les programmes scolaires de l'école élémentaire et du niveau secondaire, cette progression pédagogique est par ailleurs congruente avec celle présentée par Pelánek et Effenberger (2020, p.7). Les participants sont répartis dans les catégories (repérées par une couleur) en fonction de leur niveau en algorithmique. Le passage d'une catégorie à la suivante se fait en fonction d'un seuil de score obtenu lors d'une épreuve.

Chaque catégorie est constituée d'un parcours composé de 6 problèmes, chaque problème comportant lui-même 4 puzzles, ce qui fait donc 24 puzzles par parcours. Les 4 puzzles, accessibles par des onglets au sein d'un même problème, sont de difficulté croissante. La progression dans la difficulté, associée à des *étoiles*, peut être caractérisée comme suit :

- La version * introduit la notion algorithmique visée, ou un pré-requis. Pour cette version, un tutoriel d'aide pas à pas est accessible en cliquant sur le bouton *Plus de détails*.

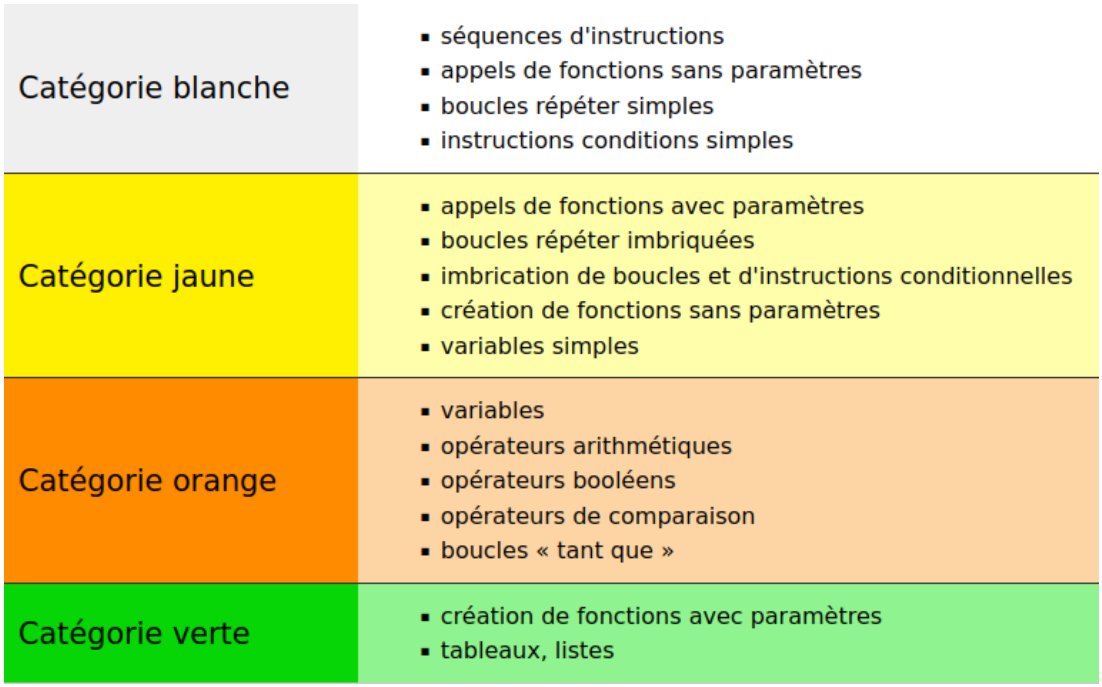


FIGURE 7.10 – Progression pédagogique attachée aux catégories de la première phase du concours Algoréa

- La version ** permet de mobiliser la même notion que dans la version *, dans un cas simple, mais désormais sans tutoriel.
- Dans la version ***, il est nécessaire de mobiliser la notion algorithmique visée en l'articulant avec plusieurs notions déjà introduites. Les organisateurs considèrent que la notion algorithmique est acquise lorsque cette version est résolue. C'est pourquoi le seuil de passage d'une catégorie à la suivante correspond à la résolution de ces versions ***.
- La version **** est volontairement plus difficile. Il s'agit d'un challenge dans lequel les notions algorithmiques visées doivent être mobilisées de manière astucieuse ou dans un contexte plus complexe.

Il ressort de la description précédente qu'une spécificité de la progression pédagogique du concours est sa structuration selon deux axes : une progression dans les notions algorithmiques d'un problème à l'autre et d'une catégorie à l'autre, et une progression dans la complexité pour une même notion au sein de chaque problème. Cette caractéristique de la progression pédagogique permet d'ouvrir le concours à une large tranche d'âge, tout en garantissant que chaque élève trouve des puzzles à résoudre qui correspondent à son niveau de

compétence. En ce sens, les versions au sein de chaque problème constituent une différenciation pédagogique auto-portée, certains élèves d'une classe pouvant chercher les versions une et deux étoiles, pendant que les plus avancés se confrontent aux versions quatre étoiles, avant de changer de catégorie le tour suivant.

7.3 La conception de puzzles de programmation dans l'environnement Algoréa

L'auteure de ce travail de recherche est impliquée dans l'organisation du concours Algoréa depuis sa création et conçoit les puzzles pour les catégories blanche et jaune du concours depuis 2018. C'est pourquoi, il nous paraît intéressant de porter un regard du point de vue du concepteur et d'expliquer en quoi cette fonction constitue un point d'appui essentiel dans la mise en place du dispositif expérimental de cette recherche. Nous abordons la nature de l'ingénierie des puzzles, à la confluence de l'ingénierie logicielle et de l'ingénierie didactique, avant de préciser les principaux choix de conception, et le contexte particulier de la situation de concours comme situation didactique au sens de Brousseau (1998).

7.3.1 Nature de l'ingénierie dans l'environnement Algoréa considéré comme un EIAH

Notre travail de recherche s'inscrit dans le champ des EIAH, dont l'objet est d'« étudier les situations pédagogiques informatisées et les logiciels qui permettent ces situations » (Tchounikine, 2009, p.13). Les puzzles de programmation proposés au concours Algoréa correspondent d'ailleurs assez bien à la situation prototypique mentionnée par Tchounikine où « l'apprenant doit réaliser une tâche au sein d'un environnement informatique et où les propriétés de la tâche, le contexte de réalisation de la tâche et en particulier l'environnement informatique (le ou les logiciels) proposé à l'apprenant pour la réaliser ont été étudiés, définis et construits en fonction des objectifs d'apprentissage visés » (Tchounikine, 2009, p.13).

Tchounikine aborde deux points de vue pour l'étude des situations pédagogiques informatisées. D'une part, plusieurs disciplines des Sciences Humaines et Sociales, dont les didactiques, se saisissent de problématiques relatives à l'enseignement/apprentissage. D'autre part, des chercheurs en informatique abordent des problématiques relatives à la conception de l'EIAH en tant que logiciel. L'approche didactique de notre travail de recherche nous situe princi-

palement du côté des Sciences Humaines et Sociales, ce travail doctoral étant inscrit en Sciences de l'Éducation et de la Formation. Nos questions de recherche, relatives à la genèse conceptuelle en lien avec le concept de motif, concernent les situations de résolution de problème auxquelles les utilisateurs de l'EIAH sont confrontées. Elles sont donc relatives aux usages de l'EIAH. Cependant, nous intervenons aussi directement sur le code informatique de l'EIAH lorsque nous concevons les puzzles du concours. À partir de là, comment positionner l'activité de conception des puzzles du concours Algoréa par rapport aux enjeux de ce travail de recherche ? S'agit-il d'ingénierie de l'EIAH au sens de Tchounikine ?

Le code informatique de l'environnement de programmation Algoréa considéré comme EIAH est structuré en trois couches. La première couche concerne les fonctionnalités de la plateforme et de l'environnement de programmation. Nous considérons que cette couche correspond à ce que Tchounikine entend par la conception du logiciel. Nous n'intervenons pas du tout sur cette couche. La deuxième couche consiste en une librairie de fonctions Javascript qui gère le micromonde *Robot sur une grille*. Nous pouvons intervenir à la marge sur cette couche, dont nous avons participé à la conception, en étroite relation avec l'informaticien qui l'a implémentée. La troisième couche gère le contenu de chaque puzzle. C'est sur cette dernière couche, constituée d'un fichier HTML et d'un fichier Javascript par puzzle, que nous intervenons lors de la conception des puzzles du concours Algoréa.

Notre activité de conception consiste essentiellement à effectuer tous les paramétrages qui définissent la situation de programmation. Elle relève autant de l'ingénierie didactique au sens de Artigue (1988) que de l'ingénierie de logiciel. En effet, l'ingénierie didactique est instrumentée par l'EIAH, le code informatique étant construit comme instrument au service de cette ingénierie didactique. Celle-ci respecte les phases définies par Artigue, si nous la considérons globalement, c'est à dire de manière itérative sur les éditions successives du concours.

La phase de conception comporte une analyse préalable, la conception proprement dite sur laquelle nous reviendrons dans la section suivante, et une analyse a priori. L'analyse préalable a consisté à ajuster le dispositif au contexte scolaire et aux enjeux de l'association, comme décrit dans la section 7.2. L'analyse a priori consiste à anticiper les stratégies des élèves pour résoudre les puzzles, afin de bloquer celles qui ne mobilisent pas la notion algorithmique visée. Cette analyse a priori est confrontée à des tests d'usage en amont de l'ouverture officielle du concours. Elle bénéficie aussi de l'expérience accumulée au fil des années. Il peut arriver qu'un participant au concours réussisse un puzzle avec un programme qui ne mobilise pas la notion algorithmique prévue, mais cela devient de plus en plus marginal. Si cela devait se produire pour les puzzles que

nous avons sélectionné pour ce travail de recherche, nous serions en mesure de le détecter par la consultation des programmes soumis pour valider les puzzles. Après chaque tour du concours, une analyse des taux de réussite pour chaque puzzle constitue la phase d'analyse a posteriori et amène un ajustement pour les tours suivants. Au fil des tours du concours, quinze entre 2018 et 2022, ce cycle d'ingénierie didactique a renforcé notre compréhension des facteurs qui influencent la difficulté des puzzles. Cette compréhension est pointée comme un point clé par Pelánek et Effenberger (2020, p.4).

Bien que nous nous appuyions sur cette ingénierie didactique, elle n'est pas en tant que telle l'objet de ce travail de recherche. Cependant, un aspect nous semble essentiel à développer pour préciser le cadre théorique de notre travail, il s'agit de la conception proprement dite des puzzles de programmation considérés comme des situations didactiques (Brousseau, 1998).

7.3.2 Choix de conception

La phase de conception de l'ingénierie didactique propre au concours Algoréa consiste à concevoir des puzzles de programmation qui nécessitent d'acquérir et de mobiliser les notions algorithmiques visées. Les choix de conception peuvent être divisés en deux parties.

Une première partie concerne les choix globaux, qui ont un effet sur l'ensemble des puzzles. Notamment, les contextes conçus pour le concours Algoréa sont volontairement épurés, c'est à dire que, sur la grille, il y a très peu d'éléments purement décoratifs, sans utilité dans la résolution du puzzle. Le texte est réduit au minimum, formulé avec une syntaxe similaire d'un puzzle à l'autre. L'objectif est de rendre le micromonde facilement compréhensible par des utilisateurs novices, en particulier par des élèves à partir de l'école élémentaire.

La conception de chaque puzzle et la mise en parcours de ceux-ci respectent les principes de design identifiés par Linehan et al. (2014) et reprises par Pelánek et Effenberger (2020, p.5).

- Chaque notion algorithmique est introduite de manière séparée. Lors de l'introduction d'une notion, seules les commandes utiles pour la résolution du problème sont disponibles.
- Chaque notion est introduite par une tâche basique. C'est l'objet des versions une étoile, qui sont accompagnées d'un tutoriel de prise en main de la commande correspondante.
- L'utilisateur a la possibilité de pratiquer (version 2 étoiles), et d'articuler la notion avec celles précédemment introduites (version 3 étoiles).

- La complexité des puzzles sur la même notion augmente avant que la notion suivante soit introduite. Dans cet item, nous retrouvons le principe de progression selon deux axes expliqué dans la section 7.2.3.

Une pluralité de paramétrages, qui sont autant de moyens technologiques au service de l'ingénierie didactique, rendent possibles la conception de puzzles variés, précisément calibrés, et en adéquation avec les principes de *design* précédemment énoncés :

- Le choix des dimensions de la grille et le placement des éléments sur celle-ci.
- Les types de bloc mis à disposition (blocs d'action, bloc de répétition, bloc de structure conditionnelle, bloc pour créer ses propres variables…).
- La manière de mettre les blocs à disposition de l'utilisateur (à plat, ou via un menu).
- Le nombre de blocs disponibles pour concevoir le programme, éventuellement un nombre de blocs disponibles par type de bloc.
- La pluralité des tests sur lesquels le programme est exécuté. Ce paramétrage est mobilisé en particulier pour amener l'utilisation de structures conditionnelles et de variables.
- La rédaction d'un feedback contextuel, pour les différents cas d'erreur détectés par la fonction de validation.
- L'ajout d'un tutoriel, ou de précisions accessibles par un clic sur un bouton *Plus de détails*.

Parmi ces paramétrages, l'un en particulier nécessite quelques précisions, car il a une fonction particulière pour notre travail de recherche. Il s'agit de la limitation du nombre de blocs dont l'utilisateur dispose pour concevoir son programme. Nous mobilisons cette limitation en nombre de blocs afin de contraindre le passage de la séquence d'instructions à la boucle. Cette limite est *dure* au sens de Pelánek et Effenberger (2020, p.10), c'est à dire que le programme ne peut pas être exécuté lorsque cette contrainte n'est pas respectée.

7.3.3 Situation didactique au sens de Brousseau

Cette section a pour objectif d'interpréter les choix de conception décrits dans la section précédente dans les termes du cadre théorique de la théorie des situations didactiques (Brousseau, 1998). En quoi la conception des puzzles de programmation du concours Algoréa s'apparente-t-elle à la conception de situations didactiques au sens de Brousseau ? Quelles en sont les spécificités ?

Comment ce cadre théorique éclaire-t-il notre choix d'adosser notre protocole expérimental au concours Algoréa ?

Pour Brousseau, le sens du mot *situation* est beaucoup plus large que pour Vergnaud pour qui le mot situation a le sens de tâche. Brousseau définit une situation comme « un modèle d'interaction d'un sujet avec un certain milieu qui détermine une connaissance donnée comme moyen, pour le sujet, d'atteindre ou de conserver dans ce milieu un état favorable » (Brousseau, 2012). Pour Brousseau, une connaissance est une manière régulière de traiter un ensemble de situations, un « moyen optimal et autonome de solution » dans une situation (Brousseau, 1998, p.10).

Parmi les situations, Brousseau distingue d'une part les *situations didactiques*, où « un actant, un professeur, par exemple, organise un dispositif qui manifeste son intention de modifier ou de faire naître les connaissances d'un autre actant, un élève par exemple et lui permet de s'exprimer en actions », et d'autre part les *situations non didactiques* où « l'évolution de l'actant n'est soumise à aucune intervention didactique directe ». Brousseau ajoute une troisième modalité, les situations *a-didactiques*, combinaison des deux premières, dans le sens où l'intention didactique est présente mais ne se traduit pas par des interventions directes (Brousseau, 2002).

Nous nous demandons ce qui est constitutif de la situation dans le contexte du concours Algoréa. Le milieu, au moment où le sujet participe au concours, comprend son environnement physique et social d'une part, et l'EIAH accessible à travers la technologie informatique (ordinateur ou tablette) d'autre part. Dans la très grande majorité des cas, le sujet se trouve dans l'environnement scolaire. Cependant, quelques spécificités sont relatives au dispositif de concours. Nous avons déjà questionné quelques caractéristiques de l'interface directement en lien avec ce dispositif de concours (Léonard, 2020). Nous nous attardons ici sur les rôles respectifs de l'enseignant et du concepteur lors la participation des élèves au concours Algoréa. Dans une situation d'enseignement classique, il revient à l'enseignant de concevoir (ou au moins de contextualiser à sa classe) et de mettre en œuvre les situations didactiques. En d'autres termes, les intentions didactiques émanent essentiellement de l'enseignant. A contrario, la situation de concours induit provisoirement une mise en retrait de l'enseignant de ce point de vue. L'enseignant se trouve alors dans une position de médiateur entre ses élèves et la plateforme de concours. Son rôle consiste pour une large part à mettre ses élèves dans les conditions de pouvoir participer : inscrire la classe au concours, garantir l'accès au matériel informatique, guider pour la connexion à la plateforme du concours. Ensuite, en principe, selon les règles définies pour le concours, il n'intervient plus, les élèves réalisant l'épreuve en autonomie. C'est à notre sens une caractéristique importante de la situation particulière

du concours, qui en fait un dispositif privilégié pour notre travail de recherche. En effet, la mise en retrait de l'enseignant au moment de la participation de ses élèves nous permet de nous focaliser sur l'unité d'analyse sujet / situation (au sens de tâche dans le cadre théorique de Vergnaud) sans que les interventions de l'enseignant ne viennent provoquer trop de perturbations dans l'activité spontanée du sujet.

Dans le dispositif de concours, c'est le concepteur qui a choisi les puzzles auxquels les élèves sont confrontés, et non l'enseignant. Le concepteur n'est bien sûr pas présent lorsque les élèves participent au concours (sauf dans certaines conditions particulières sur lesquelles nous reviendrons), il ne fait pas partie du milieu. Cependant, le parcours de programmation porte intrinsèquement les choix de conception explicités dans la section précédente. En quoi ces choix de conception peuvent-ils être considérés comme des intentions didactiques ?

Si nous nous référons à nouveau à la définition d'une situation didactique énoncée par Brousseau, le concepteur crée le milieu sur lequel l'élève devra agir pour aboutir à un état favorable, en l'occurrence une validation par une fonction de validation du programme qu'il exécute. Le nombre de blocs autorisés pour concevoir le programme, la composition de l'ensemble de blocs mis à disposition, les feedbacks et la mise à disposition de tutoriel sont destinés à guider ou contraindre l'activité du sujet et n'auraient pas leur place dans une situation de programmation ordinaire. C'est pourquoi, ces choix lors de la conception constituent des intentions didactiques de la part du concepteur. Dans la mesure où ils ne constituent pas une intervention directe mais seulement une action sur le milieu, la situation peut être catégorisée comme a-didactique au sens de Brousseau (2002).

Nous pouvons aller plus loin dans l'analyse du puzzle de programmation comme situation didactique. En effet, Brousseau caractérise plusieurs types de situation a-didactique (Brousseau, 1986, pp.98-99) :

- Les *situations d'action* où le sujet « exprime ses choix et ses décisions sans aucun codage linguistique, par des actions sur le milieu ».
- Les *situations de formulation* où le sujet « agit en émettant un message à l'intention du milieu antagoniste sans que ce message signifie l'intention d'émettre un jugement. [...] L'information ainsi donnée est supposée changer au moins l'incertitude du milieu et en général son "état" ».
- Les *situations de validation* où « les messages échangés avec le milieu sont des assertions, des théorèmes, des démonstrations, émises et reçues comme telles ».

Dans la mesure où une situation de programmation consiste à décrire un processus, elle est par essence une situation de formulation. Le passage du *faire*

au *faire faire* qui caractérise l'informatique en tant que science du traitement automatique de l'information porte le passage de l'action à la formulation décrit par Brousseau. La spécificité de la situation de programmation est que le destinataire de la formulation est une machine, et non un sujet. Dans le micromonde *Robot sur une grille*, c'est le robot qui remplit cette fonction. En conséquence, cette formulation doit être exprimée dans un langage interprétable par la machine, le langage de programmation, les connaissances en jeu étant les notions algorithmiques.

« La formulation d'une connaissance correspondrait à la capacité du sujet à la reprendre (la reconnaître, l'identifier, la décomposer et la reconstruire dans un système linguistique) »(Brousseau, 1998, p.7). Dans l'apprentissage des notions de base en algorithmique via des puzzles de programmation, c'est cette compétence à formuler efficacement un processus avec une certaine stabilité dans le temps et dans des situations similaires qui est visée.

Cependant, il est à noter que le dispositif de concours ne comprend pas intrinsèquement de phase d'*institutionnalisation*, qui visent à donner aux connaissances le statut de *savoirs*, entendus comme « les moyens sociaux et culturels d'identification, d'organisation, de validation et d'emploi des connaissances » (Brousseau, 1998, p.10). Il revient à l'enseignant d'inclure la situation de concours dans un dispositif plus large, qui, lui, comprend une ou plusieurs phases d'institutionnalisation. Dans ce sens, nous pouvons considérer que lors de la participation au concours Algoréa, et à d'autres concours similaires, les *connaissances* sont prises en charge par le concepteur du contenu du concours, alors que les *savoirs* restent de la responsabilité de l'enseignant.

7.4 Le dispositif expérimental de notre recherche

Notre travail de recherche vise l'identification des genèses conceptuelles en lien avec le concept de motif lors de l'initiation à la programmation avec un langage par blocs. Dans cette optique, le dispositif du concours Algoréa nous semble adapté.

- La mise en retrait de l'enseignant nous permet de nous focaliser sur l'unité d'analyse sujet/situation qui est celle du cadre théorique de Vergnaud que nous mobilisons pour l'étude, tout en restant dans les conditions d'un milieu écologique.

- La conception des puzzles suivant les principes de Linehan et al. (2014) et la contrainte en nombre de blocs pour concevoir le programme induisent des conditions très favorables pour une étude sur l'identification de motifs.

En effet, lors de l'introduction de la notion de boucle bornée, le bloc *répéter* est le seul bloc de structure de contrôle disponible. Le sujet induit rapidement qu'il se trouve dans la situation où il a besoin d'utiliser ce bloc. De plus, le nombre de blocs autorisé pour concevoir un programme est limité, ce qui contraint l'utilisation de ce bloc *répéter*. En conséquence, l'identification de motifs devient la tâche principale lors de la confrontation à ces puzzles.

Dans ce qui suit, nous analysons les autres avantages et les contraintes que ce choix implique, et nous définissons le périmètre de notre protocole expérimental. Premièrement, adosser notre protocole expérimental au concours Algoréa présente de nombreux avantages.

- Nous nous appuyons sur un EIAH éprouvé, qui est en production depuis une dizaine d'années et a fait l'objet d'améliorations successives. Si nous nous référons au nombre important de participants au concours, nous pouvons considérer son acceptabilité comme correcte, au moins du point de vue des enseignants qui inscrivent leurs classes sur la base du volontariat.
- Nous profitons d'une connaissance très fine du dispositif du concours Algoréa de part notre implication dans celui-ci. Les paramétrages que nous réalisons au niveau des puzzles nous laissent une latitude suffisante pour intégrer les motifs auxquels nous souhaitons que les sujets soient confrontés.
- Les puzzles du concours sont identiques pour tous les niveaux de classe que nous considérons dans l'étude, ce qui donne la possibilité d'appréhender la progression des sujets sur un temps long comme le préconise Vergnaud.
- Le dispositif de concours nous permet de collecter des données à l'échelle nationale, tout en garantissant, en situation écologique, une certaine uniformité dans les conditions de collecte. La taille de l'échantillon est à cette échelle très large, de l'ordre de plusieurs dizaines de milliers de participations par niveau de classe. Travailler sur un tel volume de données nous semble constituer un apport par rapport aux travaux antérieurs mobilisant la théorie des champs conceptuels (Vergnaud, 1991).
- Le nombre d'essais pour résoudre chaque puzzle n'est pas limité, ce qui ouvre des possibilités d'observation du processus de résolution.

Ce choix d'adosser notre protocole expérimental à un concours dont l'existence et le fonctionnement pré-existent à l'étude implique aussi un certain nombre de contraintes.

- Nous n'avons pas le choix du type de micromonde. Nous sommes contraints de restreindre les puzzles étudiés à ceux du micromonde *Robot sur une grille*. Nous renonçons de ce fait à poursuivre les investigations de nos expérimentations en amont dans l'environnement *Motif art* qui en est une adaptation.
- La séquentialité de la mise à disposition des puzzles relatifs aux motifs est imbriquée dans la progression définie pour le concours Algoréa (section 7.2.3). Nous ne pouvons pas changer l'ordre dans lequel le système présente automatiquement les puzzles, par exemple pour comparer les effets respectifs de progressions différentes.
- La temporalité de l'étude est contrainte par le déroulement du concours. Cela implique de nous organiser pour nous caler sur la chronologie du concours, notamment sur les tours de la première phase (section 7.2.2, Figure 7.9).

Ces avantages et contraintes étant explicitées, nous définissons le périmètre du dispositif expérimental de notre étude au sein de celui, plus large, du dispositif du concours Algoréa.

Ainsi, comme nous ciblons l'étude de sujets d'école élémentaire et de collège, et comme c'est le langage par blocs qui est préconisé pour ces niveaux de scolarité, nous restreignons, parmi les trois langages disponibles pour le concours Algoréa, notre étude au langage Scratch.

Dans la mesure où nos questions de recherche sont resserrées sur l'étude du concept de motif lors de l'initiation à la notion de répétition, nous nous concentrons sur quelques puzzles du parcours en catégorie blanche (sur fond bleu dans le tableau 7.1). Leur point commun est que la séquence d'actions à faire exécuter au robot virtuel comporte de la régularité, qu'il est nécessaire d'identifier pour résoudre le problème. La solution de référence implique donc une boucle ou plusieurs boucles en séquence (elle ne nécessite pas de boucles imbriquées en catégorie blanche). De plus, nous ne retenons que les puzzles comportant une seule grille, afin de ne pas introduire de complexité sans rapport immédiat avec notre sujet d'étude. Étant donnée la place des puzzles retenus au sein du parcours, lorsque les sujets abordent ces puzzles qui concernent le motif, ils ont déjà pris en main l'interface. Nous nous gardons en revanche la possibilité d'étudier certains puzzles de la catégorie jaune, où des motifs sont présents dans des situations plus complexes (par exemple boucles imbriquées, procédures).

Pour faciliter la référence aux puzzles dans la suite du document, nous attribuons à chaque puzzle un identifiant constitué de l'année, du tour du concours, de la place du problème dans le parcours et de la version du problème. Par exemple, le puzzle 2022t1p4v2 correspond à la version 2 du problème 4 pour le premier tour de l'édition 2022 du concours Algoréa. Dans le tableau 7.1, seul le problème et la version sont mentionnés.

	Version 1 étoile	Version 2 étoiles	Version 3 étoiles	Version 4 étoiles
Problème 1 : séquence	**p1v1 - Séquence** 3 à 5 instructions Tutoriel vidéo de prise en main de l'éditeur	**p1v2 - Séquence** 6 à 10 instructions	**p1v3 - Boucle (une seule instruction)** Le nombre de blocs laisse la possibilité d'une séquence.	**p1v4 - Séquence de boucles (une seule instruction) avec instructions intercalées** ou Boucle (plusieurs instructions)
Problème 2 : tests multiples	**p2v1 - Séquence** 5 à 8 instructions	**p2v2 - Séquence** 5 à 8 instructions Programme devant fonctionner sur 3 tests différents	**p2v3 - Boucle (une seule instruction)** Programme devant fonctionner sur 3 tests différents	**p2v4 - Boucle (plusieurs instructions)** Programme devant fonctionner sur 3 tests différents
Problème 3 : boucle (une seule instruction)	**p3v1 - Boucle (une seule instruction)** Tutoriel	**p3v2 - Boucle (une seule instruction)** Sujet proche du tutoriel, pas de possibilité de séquence	**p3v3 - Séquence de boucles (une seule instruction)** 2 à 5 instructions intercalées	**p3v4 - Séquence de :** **-boucles (3 ou 4 instructions)** **-boucles simples - instructions intercalées**
Problème 4 : boucle (plusieurs instructions) Pas de rotation du robot	**p4v1 - Boucle (2 instructions)** Tutoriel (2021 : possibilité d'une séquence ; 2022 : non)	**p4v2 - Boucle (plusieurs instructions)** (2021 : 2 instructions ; 2022 : plus que deux)	**p4v3 - Boucle (plusieurs instructions)**	**p4v4 - Séquence de boucles (plusieurs instructions) instructions intercalées**
Problème 5 : boucle (plusieurs instructions) Orientation relative	**p5v1 - Boucle (plusieurs instructions)** Tutoriel	**p5v2 - Boucle (plusieurs instructions)**	**p5v3 - Séquence de boucles (plusieurs instructions)**	**p5v4 - Séquence de boucles (plusieurs instructions) Instructions intercalées**
Problème 6 : structure conditionnelle	**p6v1 - Structure conditionnelle** tutoriel	**p6v2 - Structure conditionnelle** Nécessité de placer le robot sur la bonne case pour faire le test	**p6v3 - Séquence de boucle(s) et d'alternative**	**p6v4 - Imbrication d'une structure conditionnelle dans une boucle**

TABLEAU 7.1 – Progression pédagogique de la catégorie blanche du concours Algoréa

Chapitre 8

Une collecte de données structurée selon plusieurs dimensions

Sommaire du présent chapitre

8.1 Notre démarche de collecte de données

Comme expliqué dans le chapitre précédent, nous avons adossé notre protocole expérimental au concours national de programmation Algoréa. Plus précisément, nous nous intéressons aux puzzles de programmation de la première phase du concours en catégorie blanche dans lesquels le concept de motif est impliqué. Nous restreignons de plus nos investigations au langage Scratch, qui est le langage de programmation préconisé par l'institution scolaire pour les niveaux de l'école élémentaire et du collège, ciblés dans cette recherche. En conséquence, nous organisons aussi une large part de notre collecte de données autour de ce concours. Notre objectif est de collecter des données qui nous permettent d'étudier l'activité du sujet lorsqu'il est confronté à ces puzzles de programmation.

> Nous visons de combiner des analyses quantitatives et des analyses qualitatives afin d'être en mesure d'établir des résultats à la fois précis et robustes statistiquement.

Dans cette optique, nous nous demandons quelle données il est possible de collecter directement à partir de l'EIAH et de quelles données supplémentaires nous avons besoin. Nous considérons donc et discutons, au regard de nos objectifs de recherche, plusieurs aspects de la démarche de collecte de données : la taille de l'échantillon, la nature et la granularité des données collectées.

Pelánek et Effenberger dressent une liste des informations à collecter dans le cas particulier de la résolution de puzzles dans un environnement de programmation par blocs (Pelánek & Effenberger, 2020). Les auteurs rangent ces informations en quatre catégories : celles sur le contexte, celles sur le sujet, celles sur le puzzle, et enfin celles sur l'évènement.

Pour la très grande majorité des sujets, nous ne disposons pas de données sur le contexte dans lequel ils ont participé au concours Algoréa. Nous ne connaissons ni le lieu, ni le moment, ni les conditions précises de passation. La seule information à laquelle nous avons accès pour tous les sujets est la classe dans laquelle ils sont scolarisés. Il en ressort que nous ne sommes en mesure de collecter que très peu d'informations pour les deux premières catégories définies par Pelánek et Effenberger.

Par contraste, en tant que conceptrice des puzzles, nous avons à notre disposition toutes leurs caractéristiques et en avons une connaissance très fine. Pour cette raison, les informations concernant les puzzles sont renseignées du point de vue de l'expert. Une description des paramètres disponibles pour la conception de ces puzzles a été présentée dans le chapitre 7. Les puzzles retenus

pour l'étude feront l'objet d'une analyse a priori dans le chapitre 11.

L'objet de notre étude nous amène à focaliser notre recueil de données sur l'activité du sujet. Une partie de cette activité consiste à interagir avec l'environnement de programmation : déplacer le curseur de souris, déplacer et accrocher entre eux des blocs, cliquer pour lancer l'exécution d'un programme, cliquer pour accéder à un contenu... Cette activité peut être collectée sous la forme d'évènements, c'est-à-dire d'actions du sujet sur l'interface, qui sont captées et enregistrées automatiquement par le système. Ces évènements, dernière catégorie définie par Pelánek et Effenberger, sont des données discrètes dont nous discutons la granularité dans la section 8.2.

Une autre modalité est de collecter l'activité du sujet sous la forme du flux continu d'un enregistrement vidéo. Nous expliquons pourquoi mobiliser cette modalité de manière complémentaire nous est nécessaire. La mise en œuvre particulière à notre dispositif expérimental est détaillée dans la section 8.3.

La démarche de collecte que nous venons d'introduire est centrée sur l'observation du sujet, directe ou au travers de traces d'interaction, comme nous avons vu que le préconise Bationo Tillon et Rabardel (2015) dans le chapitre 5 pour des sujets jeunes. Cela dit, appréhender la conceptualisation-en-acte du sujet et comprendre la logique de ses actions, qui peut être totalement différente de celle de l'expert, peut s'avérer ardu. Dans cette optique, recueillir les énonciations verbales du sujet peut apporter de précieuses indications. C'est pourquoi, nous investiguons les méthodes de recueil du point de vue subjectif du sujet dans la section 8.4. Nous présentons aussi dans cette section une expérimentation complémentaire que nous avons mis en place afin de mieux appréhender ce point de vue subjectif du sujet.

En combinant toutes ces modalités et les possibilités de collecte dans notre contexte, nous construisons le protocole de recueil de données de manière concomitante au protocole expérimental de cette étude. Nous en présentons la structuration suivant plusieurs dimensions dans la section 8.5.

8.2 La collecte de traces d'interaction

Dans la mesure où nous considérons que les actions du sujet sur l'interface de l'environnement de programmation révèlent une part de sa conceptualisation-en-acte, nous cherchons à collecter des traces d'interaction. Lors de ce type de collecte, les questions qui se posent concernent la nature et la granularité des évènements collectés (Gao et al., 2021).

8.2.1 Granularité de la collecte de traces d'interaction

Parmi les six niveaux de granularité pour la collecte de données identifiés par Ihantola et al. (2015) dans le cadre de la programmation textuelle, trois sont relevés par Pelánek et Effenberger (2020) comme pertinents dans le contexte de la programmation par blocs : seulement la solution finale soumise, toutes les exécutions du code, tous les changements opérés dans le code.

Nous ne retenons pas le premier niveau de granularité mentionné car nous estimons celui-ci trop grossier au regard de nos objectifs de recherche. En effet, nous serions contraints d'inférer la manière dont le sujet a opéré à partir de la solution finale soumise, alors que notre recherche est centrée sur le processus de résolution. De plus, la solution finalement validée est très souvent la bonne dans notre contexte étant donné que le nombre d'essais n'est pas limité. Enregistrer seulement la solution finale soumise limiterait donc fortement le nombre de programmes exploitables pour l'analyse.

La collecte de tous les programmes exécutés correspond au deuxième niveau sur six de l'échelle de Ihantola et al. . Ce niveau de granularité permet de rendre compte plus précisément du processus de résolution et des erreurs commises, ce qui est notre objectif. Il nous semble donc un bon compromis entre une granularité assez fine pour mener les analyses prévues d'une part, et un volume de stockage et une capacité de traitement de ces données d'autre part.

Le cinquième niveau de granularité sur six de l'échelle de Ihantola et al. est l'édition de ligne de code dans un environnement de programmation textuel. Dans un environnement de programmation par blocs, nous considérons que cette granularité correspond à un changement opéré dans le code (ajout, déplacement, modification ou suppression d'un bloc). Ce niveau de granularité aurait pu nous intéresser, mais il n'est pas disponible pour l'environnement de programmation Algoréa.

En revanche, nous avons disposé d'une possibilité de collecte à un niveau de granularité encore plus fin à partir de l'été 2022. Un module de collecte, développé dans le cadre du projet xCALE [1], permet d'enregistrer toutes les interactions de l'utilisateur avec l'interface (mouvements de souris, clics, glisser-déposer de blocs, touches pressées au clavier) ainsi que les caractéristiques de cette interface (dimensions de la fenêtre et des différentes zones de l'interface de programmation). Ce niveau de granularité englobe le niveau de granularité le plus fin de Ihantola et al. qui ne comprend que les touches pressées au clavier. De ce niveau de granularité, nous pourrions déduire si besoin les modifications de programmes qui sont les informations catégorisées dans le niveau de granularité précédent.

1. Site du projet xCALE : https://www.imt-atlantique.fr/fr/recherche-innovation/collaborer/projet/xcale

Pour Pelánek et Effenberger, la captation des mouvements de souris relève plus de l'analyse des Interactions Humain-Machine (IHM) que de l'activité de résolution de problème (Pelánek & Effenberger, 2020). Pour notre part, dans la mesure où le schème est intrinsèquement lié au geste, nous faisons l'hypothèse que les mouvements de souris participent à rendre visible le raisonnement sous-jacent, ce que nous développons par la suite (chapitre 10).

8.2.2 Format de collecte des programmes édités

Les programmes exécutés par les sujets occupent une place centrale dans notre collecte de données. Un point de questionnement est relatif au format de collecte de ces programmes édités. Si des captures d'écran à partir de sessions enregistrées en vidéo peuvent être envisagées pour des analyses qualitatives à l'échelle réduite de quelques sujets comme dans l'étude de Fernandez et al. (2022), des analyses quantitatives à plus large échelle imposent un format textuel de collecte, ce que permet la collecte de traces à partir de l'environnement Algoréa.

Pelánek et Effenberger listent plusieurs possibilités de format textuel pour la collecte des programmes de l'utilisateur, parmi lesquels figure le format XML (Pelánek & Effenberger, 2020). Si l'enregistrement des programmes dans un format textuel permet des traitements à large échelle, il entraîne une nécessité de se doter d'outils pour regénérer les programmes Scratch sous forme d'images, ceci afin d'être en mesure de construire des visualisations facilement interprétables. Ce point est développé dans le chapitre 9.

8.3 La collecte de l'activité en flux continu

La collecte de l'activité en flux continu s'effectue par enregistrement vidéo.

Une première modalité consiste à placer la caméra dans la même pièce que le sujet, idéalement de manière à capter ses gestes, sa voix et son écran. Cette modalité a l'avantage de permettre d'enregistrer une large part de l'activité, mais elle présente aussi quelques inconvénients. Elle contraint l'expérimentateur à se trouver dans la même pièce que le sujet, ou à demander à celui-ci d'installer le matériel de captation.

Une seconde modalité, qui nous intéresse particulièrement, consiste à enregistrer l'activité du sujet à travers son écran. Avec cette modalité, l'environnement habituel du sujet n'est pas modifié par la présence de l'expérimentateur. L'écran occupe toute la place dans l'enregistrement, et nous sommes sûr que la captation en est de bonne qualité, sans reflets par exemple. En revanche, un inconvénient

de cette modalité est l'absence de la captation des gestes du sujet qui ne se traduisent pas par une modification de l'écran. Les enregistrements vidéo d'écran sont couramment utilisés dans le domaine des IHM pour évaluer l'utilisabilité d'un logiciel ou l'expérience utilisateur (Lazar et al., 2017).

Krieter compare l'enregistrement vidéo de l'écran de l'utilisateur avec la collecte automatique de traces d'interactions (Krieter, 2020). Par rapport à l'enregistrement de traces d'interaction, qui permet de traiter un volume important de données, l'enregistrement vidéo de l'écran de l'utilisateur ne peut s'envisager que pour des expérimentations à petite échelle en raison du temps considérable qui est requis pour l'analyse de ces données. Nous revenons sur ce point dans le chapitre suivant. Cette modalité de collecte de donnée est assez facile à mettre en œuvre techniquement dans le sens où elle ne nécessite pas d'intervenir sur le code du système informatique, mais utilise un logiciel d'enregistrement indépendant de ceux avec lesquels l'utilisateur interagit. Tang et al. pointent le caractère potentiellement intrusif dans la vie privée de cette modalité de collecte, qui peut poser des problèmes de confidentialité (Tang et al., 2006). Les auteurs rapportent que cette potentielle atteinte à la vie privée entraîne des problèmes de recrutement de participants lors des expérimentations. Ils mentionnent qu'une telle collecte de données par enregistrement de l'écran des participants nécessite un niveau élevé de confiance entre ceux-ci et l'équipe de recherche.

Pour notre étude, c'est la modalité d'enregistrement vidéo d'écran qui s'est imposée à cause du contexte de la pandémie de COVID en 2020 et 2021. Cela dit, elle nous semble un bon compromis en vue des analyses que nous projetons de réaliser.

8.4 La collecte du point de vue subjectif du sujet

8.4.1 Méthodes de recueil du point de vue du sujet

Parmi les méthodes de recherche qualitatives, nous nous intéressons aux méthodes de verbalisation de l'action, dont Forget dresse un panorama (Forget, 2013).

D'une part, la méthode de pensée à voix haute (*think aloud*) permet de recueillir des verbalisations du sujet au moment où il effectue une tâche. Dans le domaine des IHM, ce protocole est utilisé de manière concomitante à l'enregistrement de l'écran de l'utilisateur (Kushniruk & Patel, 2004 ; Planas & Cabot, 2020).

D'autre part, parmi les méthodes répertoriées par Forget, plusieurs sont indiquées pour l'investigation des connaissances-en-acte (Rix-Lièvre & Lièvre, 2012). Celles-ci ont pour point commun de postuler que l'observation du compor-

tement du sujet ne suffit pas pour saisir ses conceptualisations-en-acte et qu'en conséquence, il convient de recueillir le point de vue subjectif du sujet pour comprendre la logique de son action. Cette démarche, qui vise l'explicitation de sa pratique par le sujet lui-même, nécessite une mise en œuvre spécifique. Dans une filiation à Piaget qui explique la différence entre réussir et comprendre (Piaget, 1974b), des auteurs tels que Vermersch (2019) et Theureau (2010) ont mis au point des outils de verbalisation provoquée afin d'investiguer la conceptualisation-en-acte du sujet.

Nous explorons succinctement le cadre qui prévaut à ces méthodes en vue de les mobiliser pour notre étude. Notre objectif est de nous doter d'outils de nature à faciliter la formulation des concepts en jeu lors de la confrontation à des puzzles de programmation, afin d'accéder à la compréhension du raisonnement des sujets, tant pour documenter les procédures expertes que pour comprendre ce qui sous-tend les erreurs commises.

8.4.1.1 La méthode de la pensée à voix haute ou *think aloud*

La méthode de la pensée à haute voix (*think aloud*) est une technique utilisée en psychologie cognitive et en recherche en éducation pour étudier et comprendre les processus de pensée d'une personne pendant qu'elle effectue une tâche. Analysée par Ericsson et Simon en 1993, cette méthode consiste à encourager le sujet à exprimer verbalement ses actions, réflexions, stratégies, prises de décision au fur et à mesure qu'il opère (Ericsson et Simon, 1993 cité par Jääskeläinen, 2010). La méthode de pensée à voix haute vise à comprendre les processus mentaux des sujets en analysant ce raisonnement verbalisé. Elle permet notamment d'identifier les modes de pensée experts et les erreurs de raisonnement. Lorsqu'elle est utilisée pour des tâches qui ne requièrent pas elles-mêmes l'usage de la parole, la méthode de pensée à voix haute ne dénature pas la pensée du sujet. En revanche, l'avancée de la tâche est ralentie par cette verbalisation concomitante à l'action.

Nous pensons que la méthode de pensée à voix haute est une manière d'enrichir la collecte de traces d'interaction en les complétant par un raisonnement verbalisé du sujet au cours de l'activité de programmation. C'est pourquoi, nous mobilisons cette méthode lors des enregistrements vidéo de sessions selon la modalité décrite dans la section 8.3.

8.4.1.2 L'entretien d'explicitation

L'entretien d'explicitation est une méthode de verbalisation rétrospective, c'est-à-dire que l'entretien se déroule après l'action proprement dite. Cette méthode s'appuie sur le modèle de la prise de conscience selon Piaget (Piaget,

1974a), qui décrit les étapes de la prise de conscience, c'est-à-dire de la conceptualisation. Pour Vermersch, qui a élaboré la méthode de l'entretien d'explicitation, l'accès au processus de conceptualisation ne peut pas se faire de manière directe (Vermersch, 2019). Il nécessite une stratégie indirecte, à laquelle répond techniquement l'entretien d'explicitation.

L'entretien d'explicitation vise à amener le sujet à une « position de parole incarnée », lors de laquelle celui-ci est invité à se remémorer son vécu, à décrire la succession de ses actions, avec une granularité de plus en plus fine. L'accent est mis sur la description de la dimension procédurale de l'action. Il est demandé au sujet de décrire ses actions mais aussi ses prises d'information.

En décrivant son action de façon précise, le sujet dévoile la cohérence de ses prises de décision à l'expérimentateur. Celui-ci amène le sujet à verbaliser certaines composantes des schèmes mobilisés dans la situation qu'il se remémore : règles d'action, de prise d'information et de contrôle.

L'entretien d'explicitation, dont le dispositif est léger du point de vue de la mise en oeuvre, doit obéir à certaines règles. Le rôle de l'expérimentateur est d'amener le sujet à décrire son activité, en posant des questions sur le « comment », en encourageant à poursuivre avec des marques d'écoute du type « mm », « oui » ou des relances vides de contenu comme « et ensuite ? ». En revanche, des demandes d'explication directe commençant par « pourquoi » sont à proscrire afin de ne pas provoquer une reconstruction a posteriori.

Certains aspects de l'entretien d'explicitation sont bien adaptés à notre recherche : microtemporalité, absence de groupe témoin, compatibilité avec la démarche itérative. En revanche, le choix laissé au sujet du moment précis à se remémorer nous semble difficile à tenir dans notre contexte. En effet, notre objectif est de cibler notre recueil de verbalisations sur les moments où le sujet s'est trouvé en difficulté pendant la session. Nous souhaitons nous assurer que ces moments soient abordés, tout en garantissant un temps raisonnable d'entretien en raison du jeune âge des sujets impliqués.

8.4.1.3 Les techniques d'autoconfrontation

Theureau développe l'entretien d'autoconfrontation afin de recueillir des données verbales et gestuelles de manière différée (Theureau, 2010). L'entretien d'autoconfrontation consiste à faire verbaliser le sujet à partir de traces de son activité, souvent un enregistrement vidéo. Un dispositif alternatif consiste à remettre le sujet en situation en s'appuyant sur des traces matérielles de son activité.

Theureau mène ces entretiens à deux niveaux. Dans un premier temps, le sujet est invité à décrire simplement son activité comme s'il la revivait. Dans un second temps, il lui est demandé de changer de posture, c'est-à-dire de prendre

son activité comme objet d'analyse, de donner des clés de compréhension à propos de sa manière d'opérer. Selon Theureau, il convient de bien séparer ces deux types d'entretien. En cas d'entrelacement entre les deux postures, posture de remise en situation et posture analytique, Theureau invite à être vigilant à ce que le sujet ne reconstruise pas son activité passée en fonction de son analyse présente.

Contrairement à l'entretien d'explicitation de Vermersch, l'entretien d'autoconfrontation n'est pas basé sur les relances de l'expérimentateur, qui visent à faire décomposer son activité au sujet. Ce sont l'enregistrement vidéo ou les traces de l'activité qui assurent l'essentiel du guidage. En conséquence, le sujet ne choisit pas l'objet de la verbalisation, celui-ci étant fortement contraint par le support.

L'entretien d'autoconfrontation répond à notre besoin de focaliser l'attention du sujet sur des moments particuliers de la session, afin de mieux comprendre ce qui a déterminé ses actions au moment où il s'est trouvé en difficulté dans la résolution de certains puzzles.

8.4.2 Catégorisation de situations du point de vue du sujet

Pour notre analyse des situations de programmation, nous mobilisons la théorie des champs conceptuels de Vergnaud. Dans ce cadre d'analyse, Vergnaud préconise deux catégorisations :

- Une catégorisation des situations du point de vue de l'expert : pour notre étude, cela correspond au travail théorique de construction d'un champ conceptuel dans lequel figure le concept de motif (chapitre 11).
- Une catégorisation du point de vue du sujet : d'une part cette catégorisation a lieu de manière implicite, selon la manière dont le sujet traite la situation. D'autre part, nous cherchons à faire expliciter cette catégorisation par le sujet.

Afin de faire émerger les classes de situation du point de vue du sujet, nous explorons les tâches de catégorisation, dont Tijus et Cordier réalisent une description formelle et relèvent les méthodes expérimentales associées (Tijus & Cordier, 2003). Cette étude est centrée sur la catégorisation d'objets, mais les auteurs mentionnent à plusieurs reprises des images. Ils mentionnent aussi l'étude d'un dispositif informatique dans le domaine de l'ergonomie cognitive, en complément du recueil automatique de données. Nous en déduisons que nous pouvons nous appuyer sur ce travail pour guider notre démarche méthodologique.

Tijus et Cordier reprennent la description que donnent Mervis et Rosch (1981) : « Une catégorie existe dès lors que deux ou plusieurs objets qui peuvent être distingués sont traités de manière équivalente. ». Une catégorie peut être définie en extension, par l'énumération de ses éléments, ou par compréhension, par description de propriétés qui la caractérise. Tijus et Cordier construisent un arbre qui décrit de manière exhaustive toutes les tâches de catégorisation envisageables.

Dans cette arborescence, nous repérons la tâche qui correspond à notre objectif de faire expliciter aux sujets les catégories qu'il a construites par rapport aux situations de programmation rencontrées. Cette tâche consiste à présenter un groupe d'objets, de demander au sujet de « mettre ensemble ce qui va ensemble », de relever les regroupements qu'il effectue spontanément, puis de lui demander de justifier ses regroupements, soit en dénommant la catégorie créée, soit en en décrivant les propriétés. Les auteurs indiquent que cette tâche permet d'étudier le processus de construction de catégories en faisant varier le degré de similarité entre les objets présentés.

Ce type de tâches de catégorisation est couramment utilisé en psychologie cognitive, et notamment en psychologie du développement. Il est donc bien adapté au public cible de notre étude. Ces tâches sont de nature à nous informer sur le degré de conceptualisation du sujet, sur la construction progressive de classes par un mécanisme d'abstraction de la propriété des objets. L'exemple de la résolution successive de problèmes qui sont isomorphes du point de vue de leur structure, mais présentent un habillage différent est cité. Les critères de catégorisation sont de deux types : des critères de surface qui sont en rapport avec la perception, et des critères de structure qui sont attachés à une conceptualisation plus avancée.

Dans notre contexte, le sujet a traité des situations de programmation les unes après les autres sans que nous lui demandions explicitement d'établir de liens entre elles. Cette tâche complémentaire de catégorisation l'amène à chercher des liens entre les situations, à les organiser les unes par rapport aux autres et à verbaliser cette organisation.

Cette tâche questionne la généralisation des compétences acquises dans le cadre très restreint de la résolution de puzzles de programmation, le passage de la connaissance opératoire à une conceptualisation plus large.

8.5 Structuration de notre protocole expérimental

Après avoir présenté les différentes dimensions de notre collecte de données, nous détaillons la structure du protocole expérimental que nous mettons en place.

8.5.1 Trois échelles de collecte de données adossées au concours Algoréa

Pelánek distingue des données locales (*local data*) et des données globales (*global data*). Les données locales concernent des individus ou des items particuliers alors que les données globales décrivent toute la population ou l'ensemble du domaine (Pelánek, 2017, p.323). Notre objectif est de disposer de ces deux types de données sur les mêmes puzzles de programmation afin de pouvoir comparer ou relier entre elles ces données.

Dans notre contexte, les données globales concernent l'ensemble des participants au concours. Il s'agit d'une *échelle nationale*. Les données locales concernent des individus particuliers. Nous parlons d'*échelle du sujet*. Entre ces deux échelles, nous en intercalons une troisième, que nous appelons *échelle des classes*, et qui nous permet d'ajuster notre collecte à nos questions de recherche et à nos capacités de traitement et d'analyse. Les données sont collectées dans les mêmes conditions quelques soient l'échelle et la granularité considérées. En particulier, chaque parcours est réalisé lors d'une session en temps limité, d'une durée de 45 minutes. Nous décrivons ci-après chacune de ces échelles de collecte.

8.5.1.1 L'échelle nationale

Le concours national de programmation Algoréa implique annuellement environ 250 000 participants du CM1 à la terminale (de 9 à 18 ans). Dans le cadre de cette étude, nous nous intéressons seulement aux résultats individuels en langage Scratch pour les élèves du CM1 à la 3ème (de 9 à 15 ans), ce qui représente entre 2 700 et 92 400 participants en catégorie blanche suivant les tours (Figure 8.1). Adosser notre protocole expérimental au concours Algoréa nous permet donc de disposer de données à large échelle, que nous considérons comme des données globales au sens de Pelánek (2017).

Les participants sont répartis sur les 6 niveaux scolaires étudiés, avec une surreprésentation des élèves de collège. Aussi, nous ne maîtrisons pas la taille de l'échantillon étudié qui varie suivant les tours, mais reste conséquente. De plus, la situation est totalement écologique, la passation du concours ayant lieu en milieu scolaire ou à la maison. Nous ne disposons donc d'aucune information sur le contexte de passation. Toutefois, nous considérons que la taille conséquente des échantillons compense les variations de ce contexte.

Pour chaque tour du concours Algoréa entre 2018 et 2022, une extraction de données agrégées nous a été fournie par l'association France-ioi, à la fois pour ce travail de recherche, mais aussi pour le calibrage des sujets dans le cadre d'un cycle itératif d'ingénierie didactique (7.3.1). Pour chaque puzzle, nous disposons du pourcentage de réussite par niveau de classe. Ces pourcentages de

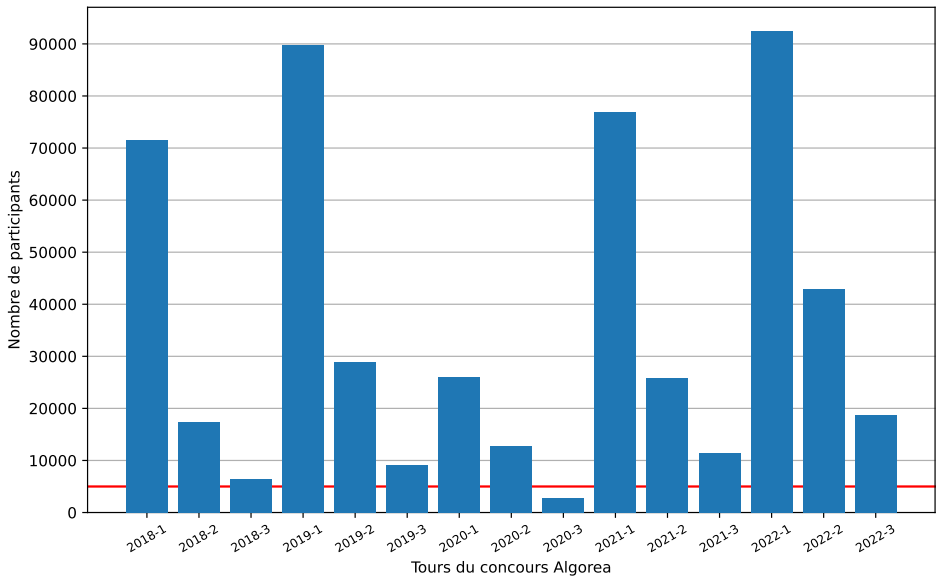


FIGURE 8.1 – Nombre de participants individuels pour les classes du CM1 à la 3ème, en catégorie blanche, utilisant le langage Scratch, pour les différents tours des éditions 2018 à 2022 du concours Algoréa

réussite, collectés auprès de plusieurs milliers d'élèves, nous donnent des repères quantitatifs à large échelle. Ils agissent comme une norme sur la population globale des participants au concours, norme à laquelle nous serons en mesure de comparer les résultats obtenus sur des échantillons plus restreints.

8.5.1.2 L'échelle des classes

Entre l'échelle nationale et l'échelle du sujet, qui réfèrent respectivement à des données globales et des données locales selon PELÁNEK (2017), nous intercalons une échelle intermédiaire qui correspond en contexte scolaire à des groupes classes. En effet, nous sommes en mesure de collecter pour cette échelle des données beaucoup plus fines qu'à l'échelle nationale, sans toutefois pouvoir recueillir des données aussi précises qu'à l'échelle individuelle.

Données collectées

À l'échelle des classes, les élèves participent au concours sur une plateforme dédiée à la recherche, `chticode.algorea.org`. Cette plateforme est une duplica-

tion de la plateforme du concours officiel concours.castor-informatique.fr. Du point de vue de l'interface avec laquelle interagissent les élèves, les deux plateformes sont strictement identiques. La différence se situe au niveau de la précision des traces collectées, qui sont beaucoup plus détaillées sur la plateforme chticode.algorea.org. En effet, tous les programmes exécutés sont collectés au format XML et horodatés, ce qui correspond au deuxième niveau de granularité de la gradation de Ihantola et al. (2015) présentée dans la section 8.2. Ces enregistrements sont stockés dans une base de données, qui est éprouvée dans la mesure où les données issues des expérimentations précédentes y ont déjà été collectées.

Plusieurs évènements déclenchent un enregistrement dans la base de données :

- Lorsque que l'utilisateur clique sur le bouton de contrôle *play* pour lancer l'exécution de son programme
- Lorsque l'utilisateur clique sur un onglet de version pour changer de puzzle
- Lorsque l'utilisateur clique sur le retour au menu (page d'accueil)

Le type d'enregistrement horodaté qui nous intéresse le plus pour notre étude est celui déclenché lors de l'exécution d'un programme. Il comprend principalement

- le nom du problème et son ID
- le type d'évènement, dont on déduit si le puzzle a été réussi ou pas
- le score obtenu dans le cas où le puzzle a été réussi, dont on déduit la version du problème, et donc précisément de quel puzzle il s'agit
- le programme exécuté, au format XML
- la date d'enregistrement

Le lecteur trouvera en annexe 1 le détail des champs présents dans la table lors d'une extraction de la base de données bebras.chticode et le type de données stocké dans chaque champ.

Après nettoyage, une session de 45 minutes comporte en moyenne 104 enregistrements, ce qui correspond à environ 2,3 enregistrements par minute.

Au regard de la catégorisation de Pelánek et Effenberger pour cette échelle de collecte, nous ne collectons pas d'informations concernant le contexte sur la plateforme chticode.algorea.org. Les seules informations concernant le puzzle sont son nom et son ID, données qui permettent de l'identifier, mais n'en donnent aucune caractéristique. Concernant les sujets, les informations collectées sont un identifiant de session et le niveau de classe.

Cette collecte est donc uniquement centrée sur les actions de l'utilisateur, en particulier sur les exécutions de programme. Nous ne sommes pas en mesure de collecter directement les états successifs du système. Ce que nous enregistrons est un ensemble d'actions de l'utilisateur qui ont pour conséquence de faire évoluer l'état du système. Celui-ci est néanmoins possible à reconstituer à partir du programme exécuté par l'utilisateur. Cette limitation n'est donc pas gênante au regard de nos objectifs de recherche.

Composition de l'échantillon

À l'échelle des classes, nous disposons de données collectées en 2021 et 2022. Pour l'édition 2021, la collecte a été perturbée par les conditions sanitaires. Nous disposons de données seulement pour le tour 1, issues de classes d'écoles élémentaires (école du bassin minier dans le Pas-de-Calais pour 2 classes de CE et une classe de CM, école de Paris pour une classe de CE et une classe de CM) et d'une classe de 6ème (très peu nombreuse, située à Paris). Pour l'édition 2022, des données ont été collectées dans deux collèges de la métropole lilloise (2 classes de 6ème et 2 classes de 3ème pour l'un et 3 classes de 5ème, 1 classe de 4ème et 1 classe de 3ème pour l'autre). Le détail de la composition de l'échantillon est présenté dans le tableau 8.1.

Classe - Tour	2021-1	2022-1	2022-2	2022-3
CE	3 classes (B :66)			
CM	2 classes (B :41)			
6ème	1 classe (B :8)	2 classes (B :43)	2 classes (B :39 ; J :6)	2 classes (B :16 ; J : 30)
5ème		3 classes (B :49)	3 classes (B :58 ; J :18)	3 classes (B :64 ; J :28)
4ème		1 classe (B :23)	1 classe (B :17 ; J :8)	1 classe (B :19 ; J :8)
3ème		3 classes (B :49 ; J :21)	3 classes (B :56 ; J :17)	1 classe (B :15 ; J :10)

Tableau 8.1 – Composition de l'échantillon à l'échelle des classes

Entre parenthèses figurent la catégorie du concours (B pour Blanche, J pour jaune) et le nombre d'élèves qui ont participé lors de chaque tour. Pour les collégiens, une information sur la participation à l'expérimentation a été adressée

aux parents des élèves par le professeur de la classe, via l'ENT (Annexe 2). Les élèves d'école élémentaire ont participé dans le cadre d'une autre recherche doctorale au sein du projet IE-CARE, pour lequel une autorisation avait déjà été obtenue.

8.5.1.3 L'échelle du sujet

Données collectées

À l'échelle du sujet, nous collectons les mêmes données quantitatives qu'à l'échelle des classes. Nous complétons par une approche qualitative qui consiste à enregistrer l'écran du sujet pendant la session du concours Algoréa et à recueillir son point de vue subjectif.

Nous avons utilisé la modalité d'enregistrement vidéo d'écran présentée dans la section 8.3. Les sessions sont réalisés à distance, en visioconférence avec l'expérimentateur. La présence de l'expérimentateur par visioconférence n'est pas mentionnée dans la littérature. Elle nous permet d'intégrer le recueil du point de vue subjectif du sujet à notre protocole.

Le logiciel Zoom est utilisé. Le sujet active son partage d'écran, son micro et éventuellement sa vidéo. De son côté, l'expérimentateur active l'enregistrement de la session. Après quelques essais, il est apparu qu'activer la vidéo ne constituait pas un apport substantiel, seul le haut du corps des sujets étant visible. En conséquence, afin de capturer un minimum de données sensibles et dans un souci de confidentialité (point de vigilance pointé dans la section 8.3), nous avons choisi de n'enregistrer que l'écran et la voix du sujet. Les gestes que le sujet pourrait effectuer devant sa machine restent donc inexplorés dans notre recherche.

Cette modalité permet une observation directe du comportement. L'enregistrement vidéo rend le flux de collecte continu et rend possible le visionnage du comportement a posteriori à des fins d'analyse. Nous collectons ces données qualitatives en vue d'appréhender finement le processus de résolution mis en œuvre par les sujets pour résoudre les puzzles de programmation. Nous recherchons en particulier à identifier des schèmes que le sujet mobilise pour traiter les situations auxquelles il est confronté, et la conceptualisation-en-acte sous-jacente.

Cependant, l'observation du comportement, directe ou à travers des traces d'interaction, n'est pas suffisante pour appréhender la conceptualisation-en-acte. C'est pourquoi, nous adjoignons un dispositif de collecte des verbalisations du sujet à notre protocole expérimental.

Protocole expérimental pour la collecte des verbalisations du sujet au cours de l'action

Nous demandons explicitement aux sujets dont nous enregistrons la session en vidéo de dire ce qu'ils sont en train de faire, selon la méthode de pensée à voix haute mentionnée précédemment. Nous leur expliquons que c'est important pour nous de comprendre leur raisonnement. Nous formulons notre demande au moment de commencer la session, mais ne contraignons pas les sujets. Après parfois une relance, si le sujet ignore notre demande, nous n'insistons pas. Pour les sujets qui accèdent à cette demande, nous nous permettons de les inviter à continuer à verbaliser leurs actions si ceux-ci arrêtent de le faire. Nous nous permettons aussi d'effectuer des demandes d'explicitation en situation.

Protocole expérimental pour la collecte des verbalisations du sujet a posteriori

Lorsque la durée de 45 minutes dédiée à la session est écoulée, l'interface se bloque, mettant un terme à l'activité du sujet. Cependant, l'enregistrement vidéo se poursuit et nous collectons des verbalisations a posteriori par rapport à l'action en cours de session.

En effet, nous ménageons un moment juste après la fin de la session que nous appelons *debrief*. Ce temps vise à confronter le sujet aux puzzles pour lesquels il a été en difficulté au cours de l'épreuve, dans une démarche qui se rapproche d'une remise en situation par les traces matérielles de l'activité (Theureau, 2010).

Bien que nous disposions de l'enregistrement vidéo de la session, nous choisissons une remise en situation à partir d'une image fixe, un programme conçu par le sujet au cours de la session. Deux raisons ont guidé ce choix. La première est pragmatique. Dans la mesure où l'entretien se déroule immédiatement après la session, il aurait été très compliqué de sélectionner les passages de la vidéo, à moins de décaler temporellement l'entretien. La durée d'entretien doit rester raisonnable en regard de la capacité de concentration de nos jeunes sujets. Il n'est pas question de durée de plusieurs heures comme mentionné dans l'exemple de l'article de Theureau (2010). Theureau invite à faire attention à la mise en scène, de nature à faciliter ou non la remise en situation. Dans notre cas de remise en situation à l'issue de la session, nous restons dans le même espace (même ordinateur, même interface), ce qui nous semble particulièrement favorable.

Deuxièmement, comme nous choisissons des moments où le sujet s'est trouvé bloqué, il pourrait être douloureux pour lui de revivre ce blocage dynamiquement. Revivre le blocage serait aussi de nature à renforcer un peu plus le processus qui y amène. Nous avons préféré partir d'une situation statique en

amont du blocage afin d'essayer de comprendre ce qui l'a provoqué, mais aussi amener le sujet à franchir le palier en étayant la résolution du problème. Cet étayage est proscrit par Theureau qui préconise d'éviter les prises de conscience et l'apprentissage pendant les entretiens d'autoconfrontation (Theureau, 2010). Il n'est en revanche pas exclu dans le cadre de l'entretien d'explicitation au sens de Vermersch (2019).

Du point de vue de la mise en œuvre, nous préparons la mise en scène du futur entretien pendant que le sujet est en activité : nous relevons au fur et à mesure du déroulement de la session les moments où le sujet bloque sur un puzzle. Nous ouvrons chacun de ces puzzles sur notre propre machine (dotée d'un double écran) et reproduisons dans l'éditeur le programme du sujet, programme auquel nous souhaitons le confronter à l'issue du temps imparti pour la session.

Ce qui est important de noter, c'est que lors de la remise en situation à partir de cette trace reconstruite, l'action revient à l'expérimentatrice, qui se laisse guider par les verbalisations du sujet. Cela permet à l'expérimentatrice de feindre de ne pas comprendre, afin d'inciter le sujet à être plus précis dans la description de l'action à lui faire réaliser. Si le début de l'entretien correspond à un entretien de remise en situation à partir des traces matérielles de l'activité (Theureau, 2010), il évolue ensuite vers une situation de formulation selon Brousseau (2011) : le sujet commande l'action de l'expérimentatrice qui agit, indépendamment de l'action antérieure lors de la session. Cette manière de faire permet à l'expérimentatrice de suivre les détours du raisonnement, que la situation de formulation contraint le sujet de verbaliser.

Du point de vue du sujet, ce moment de *debrief* permet de mener la résolution à son terme avec étayage. Il lui est présenté comme tel. De notre côté, cet entretien est de nature à amener le sujet à expliciter son raisonnement à des moments qui nous intéressent particulièrement au regard des objectifs de notre étude. Ainsi, notre protocole mobilise de manière hybride les apports de plusieurs méthodes d'investigation de la conceptualisation-en-acte.

Composition de l'échantillon

Pour cette échelle, nous avons eu recours à un échantillon que l'on peut qualifier d'*opportuniste*. Les sujets ont été recrutés par connaissance via différents réseaux : environnement professionnel de l'autrice et de son conjoint, association sportive, famille éloignée. Cependant, nous avons fait attention à ne solliciter que des sujets qui ne nous sont pas proches affectivement. Pour la plupart d'entre eux, nous n'avions jamais eu de contact direct avec eux avant l'expérimentation, ce sont les parents que nous connaissons.

	2021-1	2021-2	2021-3	2022-1	2022-2	2022-3	2022-4
CP			1				
CE1	2	2	3	2	2	2	
CE2		1	1	2	2	2	2
CM1	2	2	2	3	3	3	3
CM2	4	5	5	3	3	3	3
6ème	1	1	1	5	5	5	5
5ème				2	2	2	2
4ème	1	1	1	1	1	1	1
3ème	1	2	2	1	1	1	1
TOTAL	11	14	16	19	19	19	17

Tableau 8.2 – Composition de l'échantillon à l'échelle du sujet pour les tours du concours Algoréa de 2021 et 2022

Ce mode de recrutement présente plusieurs avantages et inconvénients. Parmi les avantages, nous relevons :

- Le côté pragmatique que ce mode de recrutement permet : nous avons commencé la collecte de données fin 2020, pendant la pandémie de COVID 19, au moment où l'accès aux établissements scolaires était quasiment exclu. Le mode de recrutement que nous avons adopté est avant tout une adaptation à ce contexte particulier.
- La mise en confiance d'emblée : comme nous connaissons tous les parents des sujets que nous avons recrutés pour les sessions enregistrées par vidéo, la confiance s'est installée très rapidement avec les sujets, qui se sont sentis à l'aise pour s'exprimer. Une autre conséquence est que nous n'avons eu aucun souci pour recueillir le consentement des parents pour ce recueil de données (document de consentement en annexe 3).
- La disponibilité sur un temps long : dans la mesure où l'expérimentation se déroule sur un temps long, de l'ordre de deux années pour certains sujets, recruter par connaissance est de nature à minimiser le risque d'abandon en cours d'expérimentation. Nous avons pu aussi ajouter une expérimentation complémentaire, non prévue à l'origine, sans que cela pose le moindre souci.

Nous sommes conscient que ce type d'échantillon présentent aussi quelques inconvénients :

- L'introduction d'un biais en termes de composition sociologique de l'échantillon : les sujets que nous avons recrutés sont tous issus d'un milieu

privilégié culturellement. Les sujets ont un bon niveau de langage et sont bien équipés en matériel.
- Une difficulté à recruter le même nombre de sujets pour tous les niveaux scolaires étudiés : certains niveaux scolaires sont surreprésentés, alors que nous n'avons qu'un seul représentant pour d'autres niveaux.

L'échantillon à l'échelle du sujet a été composé en plusieurs vagues. Le tableau 8.2 montre, suivant les tours du concours Algoréa, la composition de l'échantillon.

8.5.2 Collecte de données complémentaire par rapport au concours Algoréa

Après nos premières analyses et suite à des lectures complémentaires, il nous a semblé nécessaire de mettre en place une expérimentation supplémentaire afin d'évaluer l'effet des sessions précédentes :

- Est-ce que les schèmes identifiés lors des sessions du concours Algoréa sont mobilisés dans une situation avec des contraintes allégées ?
- Quelles sont les classes de situation construites du point de vue du sujet ?

Lors de cette expérimentation complémentaire, deux tâches sont soumises au sujet, qui visent respectivement à répondre aux deux questions ci-dessus. La première est un parcours similaire aux parcours du concours Algoréa, avec quelques paramétrages modifiés. La seconde est une tâche de catégorisation de grilles de puzzle.

8.5.2.1 Collecte de données lors de sessions sur un parcours avec contraintes allégées

Les puzzles du concours Algoréa sont paramétrés d'une manière telle que l'activité du sujet est fortement guidée. Seul un petit nombre de blocs est mis à disposition de l'utilisateur, et une limite en nombre de blocs pour éditer le programme est imposée. Ces deux paramétrages, que l'on peut considérer comme des artifices didactiques, contraignent la solution de référence ou une solution équivalente. Étant donné que la fonction d'évaluation évalue les états de la grille et non directement les programmes, procéder de cette manière assure que les participants au concours résolvent les puzzles en mobilisant les notions algorithmiques visées. En revanche, ces contraintes nous empêchent d'appréhender les stratégies que les sujets mettraient en œuvre spontanément.

C'est pourquoi, à l'échelle du sujet, nous mettons en place une expérimentation complémentaire qui vise à appréhender quel est l'effet de la suppression

des contraintes propres au concours. Pour le parcours de cette expérimentation, des situations similaires à celles du concours Algoréa sont reprises, mais avec un environnement légèrement modifié :

- Le nombre de blocs pour concevoir le programme n'est plus limité. Ainsi, les problèmes peuvent être validés avec des procédures non expertes.
- Le sous-ensemble de blocs disponibles est plus large. En particulier, les deux structures de contrôle, répétition et structure conditionnelle sont systématiquement présentes.
- À partir des versions 2 ou 3 étoiles, les blocs sont rangés dans un menu.

Dans la mesure où la fonction d'évaluation ne donne pas d'indication sur le programme qui est exécuté pour valider le puzzle, une classification manuelle des programmes corrects est requise.

Collecte de traces d'interaction complémentaires : captation des mouvements de souris

Pour le parcours complémentaire avec contraintes allégées décrit dans la section précédente, nous avons été en mesure de mettre en place une captation des mouvements de souris, rendue possible par la mise à disposition du module développé dans le cadre du projet xCALE (8.2.1).

Une difficulté de paramétrage de ce module a consisté à choisir une échelle d'échantillonnage adaptée. Plusieurs essais ont été nécessaires afin de trouver le compromis suivant, entre volume de données et précision nécessaire afin de reconstituer les interactions du sujet avec l'interface :

- Tout événement de clic est enregistré.
- En cas de déplacement de la souris, il est enregistré si : cela fait plus de 100ms depuis le dernier déplacement, et on se déplace de plus de 7 pixels par rapport aux coordonnées de la souris enregistrés lors de l'événement précédent.

Dans ces conditions, une session de 45 minutes génère en moyenne de l'ordre de 10000 enregistrements, c'est à dire entre 3 et 4 enregistrements par seconde.

8.5.2.2 Collecte de verbalisations au cours d'une tâche de catégorisation de grilles de puzzles

Les puzzles qui sont positionnés au même niveau dans chaque parcours de programmation ont la même structure algorithmique. Par exemple, la solution de

référence de tous les puzzles p4v1 comprend une boucle avec deux instructions. Il nous semble intéressant d'évaluer à l'issue de plusieurs participations à des parcours du concours Algoréa si les sujets repèrent ces structures algorithmiques à partir de la grille, ou s'ils catégorisent ces grilles selon d'autres critères. En d'autres termes, nous souhaitons connaître les classes de situations du point de vue subjectif du sujet par rapport à la prise d'information sur la grille en vue de programmer le robot virtuel.

La tâche demandée au sujet est une tâche de catégorisation parmi celles décrites par Tijus et Cordier (2003). Nous présentons au sujet des planches de six situations identiques ou similaires à celles qu'il a rencontrées lors des parcours de programmation. Nous lui demandons quelles situations il rangerait ensemble et d'expliquer son choix. En d'autres termes, nous lui demandons de créer des catégories, et de les définir d'une part en extension en listant les situations qu'il range dedans, et d'autre part en compréhension lorsque nous lui demandons de justifier son choix.

Nous avons construit 13 planches, en fonction de critères que nous anticipons a priori. Nous présentons à chaque sujet un sous-ensemble de ces planches, en fonction des critères de catégorisation que le sujet exprime. Dans la mesure du possible, lorsqu'un critère est donné par le sujet, celui-ci ne peut pas s'appliquer pour distinguer les situations de la planche suivante, afin que le sujet élargisse ses critères de catégorisation. Une liste des planches avec une analyse a priori des critères de catégorisation possibles est disponible en annexe 12.

À travers les regroupements de situations que le sujet propose et les arguments qu'il donne pour justifier ces regroupements, nous visons à identifier les classes de situations de son point de vue.

8.5.3 L'organisation temporelle de la collecte de données

Notre collecte de données sur trois échelles est réalisée sur les années 2021 et 2022 (Figure 8.2). À l'échelle nationale, cela représente six tours, en mauve sur la figure, annotés par l'année suivie d'un numéro d'ordre. Les collectes aux deux autres échelles sont calées autant que possible sur les tours du concours Algoréa. Sur la figure, chaque collecte est annotée avec le label du tour auquel elle correspond. À l'échelle des classes, la collecte se déroule de manière concomitante au concours officiel. Une exception est notable en avril-mai 2021. Ces données proviennent de l'expérimentation d'Isabelle Vandevelde, doctorante dans la même équipe qui fait une thèse sur un sujet connexe. Comme ces sujets n'ont pas participé aux autres tours du concours Algoréa, il n'y a pas d'interférences, juste un décalage temporel non significatif de notre point de vue. À l'échelle du sujet, la collecte commence en amont du concours Algoréa car les sessions servent aussi de test d'usage à petite échelle avant l'ouverture du parcours à

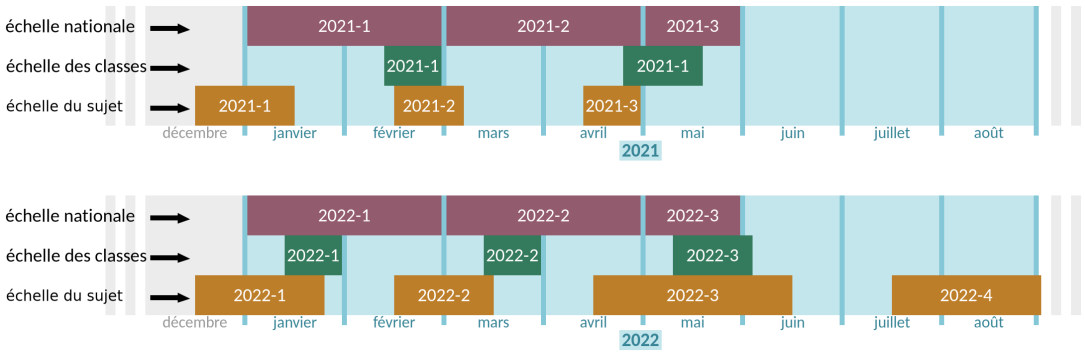


FIGURE 8.2 – Frise chronologique de la collecte de données

large échelle. Une collecte supplémentaire, en-dehors du concours Algoréa, s'est déroulée pendant l'été 2022. Cette collecte particulière (contraintes allégées et catégorisation de grilles de puzzles), dénommée 2022-4, a été décrite dans la section 8.5.2.

Pour l'échelle des classes et l'échelle du sujet, nous collectons les données lors de plusieurs tours pour les mêmes sujets, sans que cela soit systématique. La collecte a lieu à quelques mois d'intervalle sur des parcours conçus pour être sensiblement équivalents (voir tableau 7.1). En conséquence, il est possible de réaliser une analyse longitudinale pour certains sujets. Pour un puzzle placé au même niveau du parcours, il est notamment possible d'observer l'évolution des stratégies de résolution et l'évolution des erreurs commises.

8.6 Synthèse

Notre protocole expérimental consiste à collecter des données sur les mêmes puzzles, mais à des échelles différentes, et à combiner plusieurs types de collecte de données, quantitatives et qualitatives (tableau 8.3). L'uniformité des conditions de collecte rend possible la comparaison de ces données.

Notre démarche s'apparente à de la triangulation au sens que lui donne FLICK (2011). La triangulation consiste à adopter différentes perspectives sur un objet étudié. Dans les perspectives mentionnées figurent la combinaison de méthodes, de types de données, de cadres théoriques. Notre protocole inclut les deux premiers cités. Lorsque des méthodes quantitatives et qualitatives sont combinées, ce qui est notre cas, la méthode est dite mixte. Selon FLICK, la triangulation permet d'acquérir des connaissances à différents niveaux, d'aller plus loin que ce qui serait possible avec une seule approche, ce qui est bien notre objectif.

échelle\type de données collectées	Données agrégées	Traces d'interactions		Captations vidéo (écran, verbalisations)
		Programmes exécutés	Clics, mouvements de souris	
Nationale (ordre de grandeur de la dizaine de milliers)	✓			
Classes (ordre de grandeur de la centaine)		✓		
Sujets (entre 12 et 19)		✓	✓	✓

Tableau 8.3 – Synthèse du protocole de collecte de données

Ainsi, en mettant en place cette méthode mixte de collecte de données, nous visons à la fois des analyses très précises et une robustesse statistique en établissant une correspondance entre des analyses aux trois échelles de collecte. Notre protocole de collecte de données constitue un compromis entre précision des données, largeur de l'échantillon étudié et volume de données collecté, stocké et possible à traiter dans notre contexte (Figure 8.3).

Figure 8.3 – Caractéristiques des échelles de collecte

Du point de vue de la théorie des champs conceptuels :

- L'analyse des puzzles et les données quantitatives décrivant globalement l'ensemble de la population confrontée à ces puzzles renseignent l'axe des situations. Nous visons de caractériser des classes de situations, notamment pour les puzzles impliquant la notion de boucle bornée.

- Les données qualitatives à l'échelle du sujet renseignent l'axe des schèmes. Notre objectif est, à partir de ces données, d'identifier des schèmes mobilisés lors de la confrontation du sujet à des puzzles de programmation. Nous projetons de documenter les différentes composantes de ces schèmes au regard de la définition analytique d'un schème que donne Vergnaud (4.4).

Nous visons que les données quantitatives à l'échelle du sujet (traces d'interaction) adossées aux données qualitatives nous permettent d'identifier des indicateurs dans les traces d'interaction qui révèlent la mobilisation des schèmes identifiés. Dans la mesure où nous disposons des mêmes données quantitatives à l'échelle des classes et à l'échelle du sujet, nous visons une généralisation de nos résultats et une robustesse statistique à cette échelle.

Chapitre 9

Méthodes de traitement et d'analyse des données

Sommaire du présent chapitre

Notre jeu de données, dont nous avons présenté la collecte dans le chapitre précédent, est composé de données de natures différentes, qui nécessitent des traitements et analyses spécifiques. L'objet de ce chapitre est d'expliciter nos choix méthodologiques et de montrer que l'ensemble des méthodes retenues forme un tout cohérent, dans le but d'étudier le comportement, la conceptualisation-en-acte, et les stratégies des élèves lors de leur confrontation à des puzzles de programmation.

Une difficulté de notre travail de traitement et d'analyse concerne le volume important des traces collectées, qui exclut une exploration exhaustive à la main. L'enjeu est donc de se doter d'une stratégie pour combiner des analyses quantitatives basées sur l'automatisation des traitements avec des analyses qualitatives d'expert. Une autre difficulté est d'automatiser l'analyse lorsqu'on se place du point de vue de la sémantique, et non plus d'indicateurs purement quantitatifs.

Dans les sections qui suivent, nous expliquons les traitements et outils construits pour chaque nature de données, en précisant l'enjeu de ces traitements par rapport à notre cadre d'analyse. Nous menons des traitements et analyses à des granularités de plus en plus fines. Dans une première section, nous présentons comment nous combinons une analyse des situations a priori avec le traitement et l'analyse des données agrégées de l'échelle nationale. Les traitements suivants consistent, pour l'échelle des classes, à considérer la validité des réponses et le temps passé sur les puzzles, afin de construire un indicateur et des profils attachés à la confrontation sujet/puzzle. Nous nous intéressons ensuite aux programmes enregistrés, afin d'analyser les solutions valides et les erreurs commises. Nous complétons par la description des traitements sur les mouvements de souris, le niveau de granularité le plus fin dans notre étude pour ce qui concerne les traces d'interaction. Enfin, nous analysons qualitativement les enregistrements vidéo de sessions, en y relevant notamment les verbatim.

Pour chaque type d'analyse, nous précisons les données mobilisées, nous nous appuyons sur la littérature existante ou/et sur le cadre d'analyse, nous

détaillons les traitements et analyses mis en place, et nous expliquons l'apport pour l'étude et l'articulation avec les autres types d'analyse.

9.1 Échelle nationale : analyse quantitative de données agrégées en appui de l'analyse a priori

À l'échelle nationale, nous disposons de données agrégées, en l'occurrence de la fréquence de réussite pour chaque puzzle, par niveau de classe. Ces données viennent en appui de l'analyse épistémologique des situations, qui est le premier axe préconisé par Vergnaud (1991) pour l'étude d'un champ conceptuel. Dans notre contexte, une situation de programmation correspond à un puzzle. Les données dont nous disposons nous permettraient de renseigner l'ensemble des notions de base de l'algorithmique pour la programmation impérative présentées au chapitre 1. Cependant, dans le cadre de ce travail doctoral, nous nous limitons à réaliser cette analyse épistémologique pour des situations qui mettent en jeu le concept de motif. Leur point commun est que la séquence d'actions à faire exécuter au robot virtuel comporte des répétitions, qu'il est nécessaire d'identifier pour résoudre le problème. La solution de référence implique donc une boucle ou plusieurs boucles en séquence (mais elle ne nécessite pas de boucles imbriquées pour les puzzles sur lesquels nous focalisons notre attention). En prenant en compte cette caractérisation, nous retenons 139 puzzles en catégorie blanche, issus des éditions 2018 à 2022.

Nous réalisons une analyse *a priori* de ces puzzles, qui est développée dans le chapitre 11. Notre travail d'analyse débute par l'identification de différents critères de catégorisation de ces puzzles, c'est à dire de variables de situation. Cette identification est étroitement liée à l'ingénierie didactique de notre travail de conception des puzzles. Il s'agit ensuite de renseigner manuellement les valeurs de ces variables de situation pour tous les puzzles étudiés.

Dans un second temps, nous confrontons cette catégorisation a priori avec une étude de statistique descriptive, en prenant le puzzle comme individu statistique. Chaque puzzle est associé à sa fréquence de réussite lors du concours Algoréa, global ou par niveau de classe, fréquence de réussite qui constitue la variable statistique étudiée.

Pour ce travail, nous nous inspirons de l'étude de Vergnaud et Durand qui concerne les structures additives en mathématiques (Vergnaud & Durand, 1976). Dans cette étude, les auteurs ont donné à résoudre des problèmes additifs dont la réponse est strictement la même numériquement, mais pour lesquels la formulation du problème induit un raisonnement différent. Afin de caractériser des paliers de difficulté, ils ont catégorisé les problèmes en classes de situations

qu'ils ont désignées par les prénoms utilisés dans l'énoncé des problèmes : les problèmes dits *Pierre* qui sont résolus rapidement par une majorité d'élèves, les problèmes dits *Paul* qui résistent plus longtemps, car moins intuitifs et les problèmes dits *Jacques* qui restent difficiles même pour des adultes.

Vergnaud incite à envisager la progressivité de la conceptualisation en la considérant sur un temps long, de l'ordre de plusieurs années. Ainsi, Vergnaud et Durand ont mené leur expérimentation avec 28 élèves de chaque niveau du CP au CM2. Leur approche consiste à mesurer le décalage temporel entre la résolution des différents types de problème. La figure 9.1 montre les résultats de l'expérimentation pour un problème de type *Pierre* et un problème de type *Paul*. Le décalage temporel dans la résolution de ces deux problèmes est représenté par le décalage des courbes sur l'axe horizontal (flèche orange que nous avons ajoutée au diagramme).

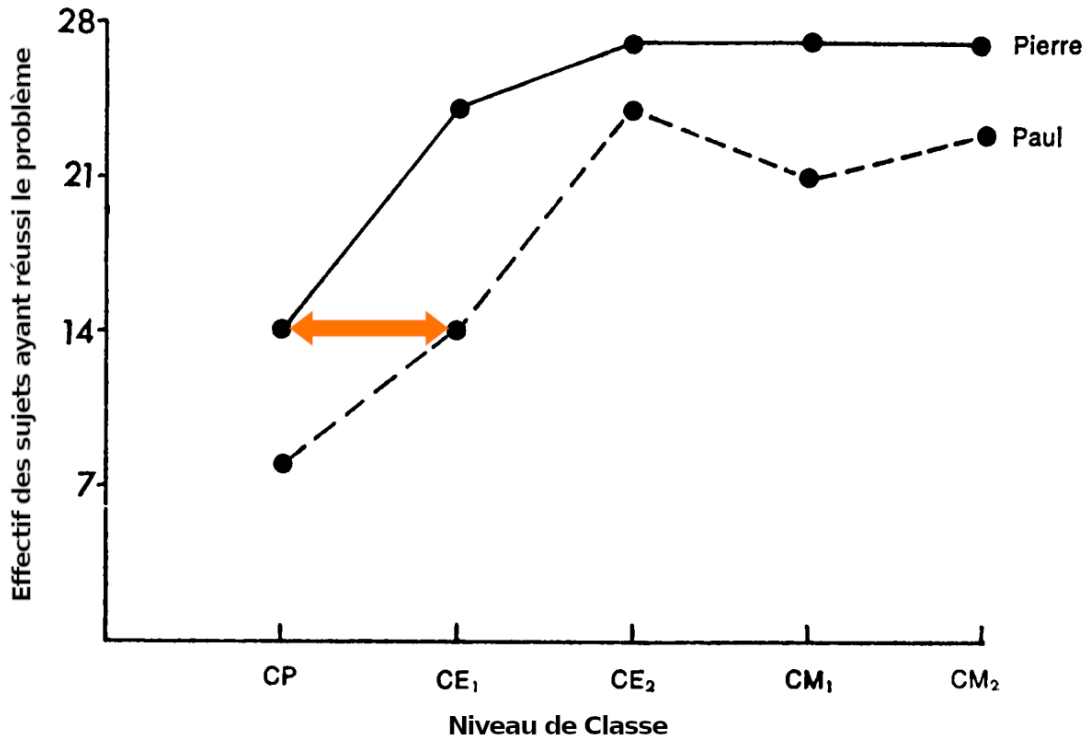


Figure 9.1 – Comparaison des résultats aux problèmes *Pierre* et *Paul* issus de l'étude de Vergnaud et Durand sur la complexité psychogénétique des structures additives

Nous retenons le principe de considérer la performance de sujets dans la réalisation de la tâche pour appuyer la catégorisation de situations, et de présenter

les résultats par niveaux de classe afin de rendre compte de la progression sur plusieurs années. En effet, les fréquences de réussite à l'échelle nationale nous permettent de mener le même type d'analyse que Vergnaud et Durand, avec cependant quelques différences.

D'une part, l'étude de Vergnaud et Durand a eu lieu dans le cadre d'une expérimentation contrôlée, pendant laquelle chaque sujet était interrogé individuellement par un expérimentateur. Lors du concours Algoréa, les résultats sont collectés automatiquement via l'environnement de programmation. L'avantage est que notre échantillon est beaucoup plus large que celui de Vergnaud et Durand, plusieurs milliers de sujets par niveau de classe contre vingt-huit. L'inconvénient est que nos données sont collectées dans une situation écologique, dont nous ne contrôlons pas les conditions de passation hormis la durée de la session.

D'autre part, dans l'étude de Vergnaud et Durand, la solution attendue est strictement identique pour tous les problèmes soumis aux sujets. Ce n'est pas le cas dans notre contexte. Les situations de programmation sont similaires mais la solution attendue en termes de programme édité n'est pas strictement identique. Elles sont similaires dans le sens où ce sont toutes des situations de programmation d'un robot virtuel sur une grille qui requièrent l'utilisation du bloc *répéter*. La solution attendue a toujours la même structure du point de vue algorithmique, mais sans être strictement identique. Ce point introduit un élément de complexité supplémentaire pour l'interprétation des résultats.

Nous reprenons le même type de graphique pour la présentation de nos résultats (Figure 9.2).

Les niveaux de classe que nous étudions vont du CM1 à la 3ème. Sur l'axe des ordonnées, nous remplaçons l'effectif de l'échantillon par la fréquence de réussite. Dans le cas où nous étudions un ensemble de puzzles qui présentent une caractéristique commune, nous retenons la médiane, la moyenne et la zone interquartile comme indicateurs de la distribution des données. La figure 9.2 montre un exemple d'un tel graphique, pour un puzzle particulier (en orange), et pour un ensemble de puzzles (en bleu).

Avec cette étude, nous espérons vérifier nos hypothèses sur la catégorisation a priori des situations. Nous visons des repères globaux sur les situations de programmation qui mettent en jeu le concept de motif. Les résultats obtenus, parce qu'ils le sont à large échelle, sont en mesure d'avoir une robustesse statistique qui nous permet de les considérer comme des repères auxquels nous référer lors de nos analyses aux échelles plus réduites des classes et des sujets. Nous mettons en œuvre des tests de comparaison de moyennes afin d'établir le caractère significatif ou non des résultats obtenus.

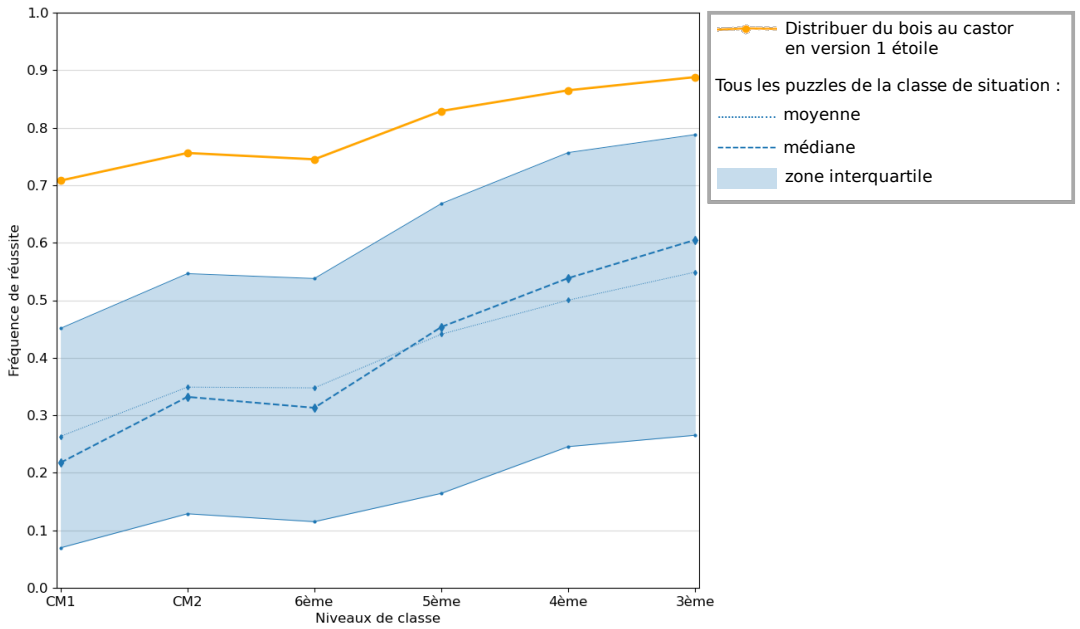


FIGURE 9.2 – Exemple de diagramme montrant l'évolution de la fréquence de réussite en fonction du niveau de classe

Du point de vue méthodologique, l'apport de notre travail se situe sur le passage à l'échelle du type d'expérimentation mené par VERGNAUD et DURAND. La collecte automatique des données rend possible d'obtenir des résultats sur de larges échantillons, et donc de conférer une robustesse statistique à ceux-ci.

Pour aller plus loin, nous serions en mesure d'adjoindre un aspect prédictif à notre étude. En effet, les variables de situation que nous avons renseignées pourraient constituer les données d'entraînement d'algorithmes de *machine learning* qui viseraient à prédire le taux de réussite d'un nouveau puzzle. Ce travail est initié par la construction d'un modèle de régression linéaire mais sera surtout poursuivi au-delà de cette thèse.

9.2 Échelle des classes : analyse quantitative à partir de la validité des réponses et du temps passé sur les puzzles

Dans cette section, nous nous situons au niveau du deuxième axe préconisé par Vergnaud pour étudier la conceptualisation-en-acte du sujet, celui de l'activité du sujet. Nous visons de définir une méthode qui nous permette, à partir des traces d'interaction, d'étudier les compétences des élèves à résoudre des puzzles dans un environnement de programmation par blocs. Les détails de cette méthode de traitement sont consultables dans l'annexe 6 (préparation des données) et dans l'annexe 7. Les questions de recherche que nous adressons dans cette section sont :

- Quels indicateurs construits à partir de traces d'interaction rendent compte de l'existence de compétences à résoudre des puzzles dans un environnement de programmation par blocs ?
- Quels profils relatifs au degré de compétence à résoudre des puzzles dans un environnement de programmation par blocs peut-on définir à partir de ces indicateurs ?

9.2.1 Ancrage théorique : compétence et schème dans la théorie des champs conceptuels

De nombreuses acceptions du terme de *compétence* existent (Crahay, 2006), ce qui nous incite à situer ce terme dans le cadre d'analyse de Vergnaud. Selon Vergnaud, « Sans conceptualisation, il n'y a pas d'activité opératoire possible et pas de compétence » (Vergnaud, 2003). Par compétence, Vergnaud désigne « la forme opératoire de la connaissance, qui permet de faire et de réussir » (Vergnaud, 2012). Pour Vergnaud, le concept de compétence présente une limite, dans la mesure où il « renvoie d'abord au résultat de l'activité et insuffisamment à l'organisation de l'activité elle-même. » (Vergnaud, 1996). Malgré cela, il convoque ce concept à plusieurs reprises, mentionnant par exemple que « A est plus compétent que B, s'il s'y prend d'une meilleure manière. Le comparatif "meilleure" suppose des critères supplémentaires : rapidité, fiabilité, économie, élégance, compatibilité avec la manière de procéder des autres, etc... » (Vergnaud, 2012). Concernant l'organisation de l'activité, elle est prise en charge par le concept de schème, défini comme une « organisation invariante de la conduite pour une classe donnée de situations » (Vergnaud, 1991). En d'autres termes, le concept de schème explique la stabilité et la permanence des compétences.

Dans l'article de 1991 intitulé « La théorie des champs conceptuels », Vergnaud catégorise les situations (ou tâches) en deux classes :

> 1. des classes de situations pour lesquelles le sujet dispose dans son répertoire, à un moment donné de son développement et sous certaines circonstances, des compétences nécessaires au traitement relativement immédiat de la situation ;
> 2. des classes de situations pour lesquelles le sujet ne dispose pas de toutes les compétences nécessaires, ce qui l'oblige à un temps de réflexion et d'exploration, à des hésitations, à des tentatives avortées, et le conduit éventuellement à la réussite, éventuellement à l'échec.
>
> Le concept de « schème » est intéressant pour l'une et l'autre classes de situations, mais il ne fonctionne pas de la même manière dans les deux cas. Dans le premier cas on va observer pour une même classe de situations, des conduites largement automatisées, organisées par un schème unique ; dans le second cas, on va observer l'amorçage successif de plusieurs schèmes, qui peuvent entrer en compétition et qui, pour aboutir à la solution recherchée, doivent être accommodés, décombinés et recombinés ; ce processus s'accompagne nécessairement de découvertes.(Vergnaud, 1991, p.136)

Nous retenons deux éléments essentiels de ce passage sur lequel nous fondons une part de notre méthode d'analyse. D'une part, la durée nécessaire au traitement de la situation est la variable déterminante pour distinguer ces deux classes de situations. D'autre part, Vergnaud passe

- du traitement immédiat d'une situation, induit par l'existence de compétences, à l'organisation de la conduite selon un schème unique dans un cas,
- et d'un temps de traitement substantiellement plus long pour traiter la situation, induit par l'absence d'une ou plusieurs compétences, à la mise en défaut et la restructuration sous-jacente du système de schèmes du sujet (que nous résumons sous le terme accommodation) dans l'autre cas.

En conséquence, nous cherchons un ou plusieurs indicateurs construits à partir des traces d'interaction dont nous disposons, qui permettent de rendre compte de la durée irréductible à l'exécution de la tâche (traitement immédiat révélateur de la mobilisation d'un schème unique) et du temps éventuellement nécessaire à un processus d'accommodation. En d'autres termes, nous cherchons à modéliser le traitement de la situation par le sujet en nous fondant sur la

distinction opérée par Vergnaud, afin d'inférer si les compétences sous-jacentes sont présentes. Dans ce but, nous cherchons en premier lieu une visualisation qui nous permette d'identifier ces deux classes de situations à partir de nos données. Nous construisons alors notre modélisation avec comme objectif de déterminer automatiquement des profils, que nous appelons profils de confrontation sujet/situation, fondés sur le traitement immédiat ou non du puzzle de programmation.

9.2.2 Appui sur la littérature existante en analyse de traces

Pelánek dresse un panorama des aspects à prendre en considération lors de la modélisation d'un domaine et la modélisation d'un processus d'apprentissage, en fonction du contexte et de l'objectif de la modélisation (Pelánek, 2017).

Parmi les travaux qui concernent la modélisation du processus d'apprentissage, beaucoup ne prennent en considération que la réponse finale, juste ou fausse, ou l'état de réalisation final du programme, par exemple ceux de Moreno-León et al. pour l'environnement Scratch (Moreno-León et al., 2015). Ces travaux induisent des éléments de processus de réalisation à partir de l'état final. De notre côté, nous souhaitons nous concentrer sur la conceptualisation-en-acte des sujets, c'est à dire sur le cœur du processus de résolution des situations de programmation. Nous ne retenons donc pas ce type de travaux, qui manquent de précisions au regard de ce que nous visons d'analyser.

En revanche, une première étape de notre stratégie d'analyse nous amène à considérer la validité des réponses associée au temps nécessaire pour résoudre la tâche et à l'historique des réponses soumises, qui, selon Pelánek, sont indiqués pour mesurer l'aisance (*fluency*) lors d'un processus d'apprentissage (Pelánek, 2017).

Selon Pelánek, le temps nécessaire pour résoudre la tâche est potentiellement une source d'information, mais la manière de l'exploiter nécessite des recherches approfondies (Pelánek, 2017). Pelánek explore ce temps pour résoudre une tâche, associé à la nature des réponses fausses (Pelánek, 2018). Il construit un indicateur pour normaliser le temps de réponse par rapport à la difficulté de la tâche. Cet indicateur reste cependant dépendant de l'échantillon considéré. En combinant temps nécessaire pour résoudre la tâche et catégorisation des erreurs, Pelánek établit quatre *profils de performance* : réponse correcte, réponse fausse dans un temps très court, erreur commune sans que le temps soit court, autre erreur (Pelánek, 2018). Dans le cas de problèmes de difficulté croissante, Pelánek pointe la difficulté à distinguer ce qui relève de la montée en difficulté du problème et ce qui relève de l'apprentissage du sujet (Pelánek, 2017).

Declercq et Zeyringer proposent deux autres indicateurs construits pour caractériser le processus de résolution d'une situation de programmation. Le

premier indicateur, que les auteurs nomment *taux d'anticipation*, est le rapport du nombre d'ajout d'instructions sur le nombre total d'actions (ajouts, suppressions, exécutions). Cet indicateur, plus élaboré que le nombre d'essais, témoigne des hésitations du sujet pendant le processus de résolution. Le second indicateur, appelé *taux de d'efficacité*, est le rapport entre le nombre d'instructions de la solution de référence, et le nombre d'instructions utilisé par le sujet. Il mesure à quel point la solution de celui-ci est proche de la solution optimale (Declercq & Zeyringer, 2018). Pour ces deux indicateurs, l'aspect temporel n'est pas directement pris en compte.

Une manière d'établir des profils attachés à la résolution d'un problème de programmation, alternative à la construction d'indicateurs, est proposée par Jiang et al., en utilisant une méthode de *clustering* (Jiang et al., 2022). Les auteurs caractérisent quatre profils à partir des programmes intermédiaires soumis : ceux qui abandonnent rapidement (*quitters*), ceux qui sont loin d'une solution valide lors des premiers essais, s'en approchent plus ou moins sans aboutir (*approachers*), ceux qui sont loin d'une solution valide lors des premiers essais puis s'en approchent jusqu'à trouver (*solvers*), ceux qui soumettent un programme valide dès les premiers essais (*knowers*).

Pour notre recherche, nous retenons l'idée de combiner plusieurs indicateurs pour établir des profils concernant la confrontation d'un sujet avec un puzzle. Nous cherchons une combinaison qui prend en compte à la fois l'aspect temporel et le nombre d'essais. Notre contribution porte sur la mise en correspondance des indicateurs et profils définis avec le cadre d'analyse de la théorie des champs conceptuels (Vergnaud, 1991).

9.2.3 Visualisation sous forme de frise chronologique et identification de profils

Nous avons montré dans la section 9.2.1 que l'aspect temporel est essentiel à prendre en compte dans l'analyse de la résolution de puzzles de programmation. En conséquence, nous cherchons une visualisation qui le met en évidence. Parmi les modalités de visualisation de données répertoriées par Khan et Khan (2011), nous retenons la frise chronologique. Ce type de visualisation est déjà utilisé dans le contexte de la programmation, plutôt pour l'étude du processus de résolution d'un problème particulier (Chevalier et al., 2020 ; Fernandez et al., 2022 ; Toll & Wingkvist, 2018).

Nous avons donc conçu une fonction qui permet, pour chaque sujet, de générer automatiquement une frise chronologique de sa session à partir des évènements horodatés. La figure 9.3 montre un extrait de frise chronologique obtenue.

FIGURE 9.3 – Visualisation de la session d'un sujet sous forme de frise chronologique (zoom sur les puzzles 4 et 5 du parcours, entre les 6ème et 18ème minutes)

Cette visualisation procure une vue synthétique dotée des principales informations sur le déroulé temporel d'une session. L'axe horizontal est un axe temporel qui porte les différents évènements survenant pendant la session : changements de puzzle et exécutions. Chaque problème est représenté par une couleur dédiée, chaque version de problème (puzzle) par une nuance et une hauteur : plus clair et plus bas pour la version la plus facile jusqu'à plus foncé et plus haut pour la version la plus difficile de chaque problème. Chaque exécution d'un programme invalide est représentée par un trait noir, chaque soumission d'un programme valide par un trait blanc.

La visualisation sous forme de frise chronologique nous permet d'observer différents profils construits à partir de la distinction de VERGNAUD (1991) à propos du traitement immédiat ou non d'une situation (Figure 9.4). Un profil est associé à un couple sujet/puzzle. Différents profils se succèdent pour un même sujet au cours d'une session.

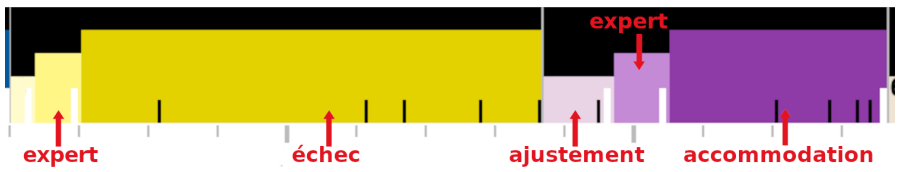


FIGURE 9.4 – Identification de profils sur la frise chronologique d'une session (zoom)

- Profil **expert** : le sujet résout le puzzle très rapidement en un seul essai. La résolution du puzzle est organisée par un schème unique ou un système de schèmes, opérationnel dans la situation.
- Profil **ajustement** : le sujet résout le puzzle rapidement après une ou deux soumissions ratées. Il s'organise autour d'un schème (ou d'un système de schèmes) opérationnel pour traiter la situation, mais avec des approximations lors de la mise en œuvre.
- Profil **accommodation** : le sujet passe un temps conséquent sur le problème

et fait plusieurs soumissions ratées avant de soumettre un programme valide. Il ne dispose pas de schème directement opérationnel pour traiter la situation. Il entre dans un processus d'accommodation qui aboutit positivement.

- Profil **échec** : le sujet est en échec sur le problème. Comme pour le profil précédent, le sujet entre dans un processus d'accommodation, mais celui-ci ne permet pas d'aboutir à un programme valide pour résoudre le puzzle.

Nous identifions par ailleurs deux autres profils, non visibles sur l'exemple de frise de la figure 9.4.

- Profil **passé** : le sujet ouvre brièvement un puzzle sans lancer d'exécution de programme.
- Profil **indéterminé** : le sujet résout le puzzle en un seul essai mais avec un temps long, qui ne nous permet pas de conclure sur la mobilisation d'un schème opérationnel. Nous observons plusieurs cas sur nos enregistrements vidéos où le sujet suspend momentanément sa recherche pour faire autre chose, mais aussi des cas où le sujet manipule longuement des blocs dans l'éditeur avant de lancer une exécution et des cas où le sujet consulte le tutoriel disponible.

9.2.4 Construction d'un indicateur de rapidité normalisée

En nous appuyant sur les éléments théoriques présentés dans la section 9.2.1, nous cherchons à déterminer automatiquement, à partir des traces d'interaction, si le sujet réussit un puzzle de programmation de manière experte, en mobilisant un schème opérationnel (validation rapide et quasiment sans se tromper), ou s'il s'engage dans un processus d'accommodation.

À cette fin, nous calculons le temps passé sur chaque puzzle et le nombre d'essais réalisés avant d'aboutir à une solution valide ou à un abandon. En utilisant ces données, notre objectif est de déterminer un seuil en temps et en nombre d'essais au-delà duquel nous considérons qu'un processus d'accommodation est engagé, qu'il aboutisse à une résolution du problème (profil *accommodation*) ou non (profil *échec*).

Nous fixons le seuil en nombre d'essais à trois, ce qui inclut la nuance d'un schème opérationnel qui fait l'objet d'une ou deux erreurs de mise en œuvre, et qui est ajusté aussitôt (profil *ajustement*). En effet, nous avons remarqué à de nombreuses reprises que dans ce cas, le programme soumis est corrigé très rapidement, puis à nouveau exécuté et validé par le système.

Comme argumenté dans la section 9.2.1, la vitesse de résolution du problème est essentielle pour déterminer le degré de compétence d'un sujet face à une

situation. Cette vitesse dépend du nombre de blocs à déplacer pour obtenir le programme solution et de la dextérité du sujet. Un sujet plus jeune peut disposer d'un schème opérationnel, mais une moindre dextérité avec la souris induit un temps de résolution plus long. Notre objectif est de construire un indicateur qui ne dépende plus ni du nombre de blocs à déplacer pour obtenir la solution de référence, ni de la dextérité du sujet.

9.2.4.1 Temps moyen individuel pour placer un bloc

Comment rendre compte de la dextérité du sujet à travers les traces ?

La dextérité du sujet relève d'une certaine maîtrise instrumentale, à travers le maniement de la souris comme instrument pour déplacer les blocs par glisser-déposer. Cette dextérité est à distinguer de la maîtrise des fonctionnalités de l'environnement de programmation, qui est un autre aspect de la maîtrise instrumentale sur laquelle nous reviendrons.

La dextérité du sujet dans le maniement de la souris se manifeste en grande partie par la vitesse à laquelle il déplace et accroche les blocs les uns aux autres. D'où l'idée de chercher à calculer un temps moyen individuel pour placer un bloc. Placer un bloc dans l'éditeur est une action élémentaire du point de vue de la programmation par blocs, et le temps nécessaire à cette action élémentaire nous donnerait une bonne indication de la dextérité du sujet.

Cependant, sauf pour la dernière expérimentation, nous ne disposons pas précisément des logs de pression et de relâchement du clic gauche de la souris, qui nous auraient permis de calculer exactement ce temps moyen. Donc une mesure aussi précise n'est pas à notre portée. Nous cherchons donc plutôt une approximation acceptable de ce temps moyen.

Dans cette optique, nous choisissons de ne considérer que les premières soumissions de chaque confrontation sujet/puzzle. Deux raisons à cela :

- Lors des premières soumissions, sauf exception, il n'y a que des ajouts de blocs, à partir de l'éditeur vide. Alors que pour les soumissions suivantes, il faudrait aussi considérer les suppressions et les remplacements, ce qui engendre une marge plus grande d'erreur.
- La première soumission correspond au schème mobilisé en première intention pour traiter la situation, que celui-ci soit opérationnel ou non.

En première approximation, nous calculons donc un temps moyen individuel pour placer un bloc dans l'éditeur t_{bloc} de la manière suivante :

$$t_{bloc} = \frac{\sum_i t_{prem}(i)}{\sum_i N_{prem}(i)} \tag{9.1}$$

où $\sum_i t_{prem}(i)$ est la somme sur tous les puzzles du parcours des durées des premiers essais et où $\sum_i N_{prem}(i)$ est le nombre de blocs déplacés lors de l'ensemble des premiers essais.

Le problème de cette première approximation est que des valeurs aberrantes de durée (par exemple si le sujet a suspendu son activité, ou est allé consulter longuement le tutoriel,..) ont un impact non négligeable sur le résultat obtenu.

C'est pourquoi nous affinons le processus, en calculant d'abord le temps moyen pour placer un bloc non plus globalement, mais puzzle par puzzle. Nous calculons ensuite la moyenne et l'écart type de ces temps moyens par puzzle. Nous éliminons alors les valeurs aberrantes de temps moyen attaché à un puzzle, en fixant un seuil à trois écarts type de la moyenne, pratique courante en statistiques.

Nous réitérons alors le calcul 9.1 mais cette fois en ayant écarté les enregistrements qui correspondent aux valeurs aberrantes identifiées.

9.2.4.2 Indicateur de rapidité normalisée

Nous construisons ensuite un indicateur de rapidité r_i attaché à un puzzle i (formule 9.2). Celui-ci prend en compte le nombre de blocs à déplacer, en nous basant sur le nombre de blocs de la solution de référence $N_{ref}(i)$ pour ce puzzle i et sur t_i qui est le temps passé sur ce puzzle. Dans le cas d'un expert qui soumet la solution de référence en un seul essai, cet indicateur correspond au temps moyen nécessaire pour placer un bloc. À ce stade, cet indicateur ne tient pas encore compte de la dextérité du sujet.

$$r_i = \frac{t_i}{N_{ref}(i)} \tag{9.2}$$

Nous calculons enfin un indicateur de rapidité normalisée R_i (formule 9.3) pour le puzzle i. Pour cela, nous centrons l'indicateur de rapidité r_i autour de 0 en prenant l'écart entre r_i et t_{bloc} et nous le réduisons en divisant par le temps moyen mis pour placer un bloc. Nous obtenons ainsi un indicateur qui est indépendant de la dextérité du sujet et du nombre de blocs de la solution de référence.

$$R_i = \frac{r_i - t_{bloc}}{t_{bloc}} \tag{9.3}$$

Si cet indicateur R_i est négatif, cela signifie que le sujet a résolu plus rapidement le puzzle i que la moyenne de ses premières soumissions. Dans ce cas, il y a absence d'accommodation, le sujet mettant en œuvre un schème unique si l'on se réfère au cadre d'analyse de la section 9.2.1. Plus cet indicateur est élevé,

plus le sujet a passé de temps d'accommodation sur le puzzle i relativement à sa dextérité et au nombre de blocs nécessaires pour résoudre ce puzzle.

9.2.5 Détermination de profils

Combiné au nombre d'essais, l'indicateur de rapidité normalisée précédemment calculé nous permet de déterminer automatiquement les profils identifiés visuellement sur les frises (9.2.3). À cette fin, il est nécessaire de définir un seuil pour l'indicateur de rapidité normalisée. En effet, ce seuil détermine la répartition entre les profils expert et indéterminé d'une part, ajustement et accommodation d'autre part. Nous incluons pour ce calcul l'ensemble de nos données de 2022 à l'échelle des classes pour les catégories blanche et jaune, ce qui représente 12340 confrontations d'un sujet avec un puzzle.

Nous visualisons la courbe de la fréquence des profils expert et ajustement en fonction du seuil considéré pour l'indicateur de rapidité normalisée (Figure 9.5).

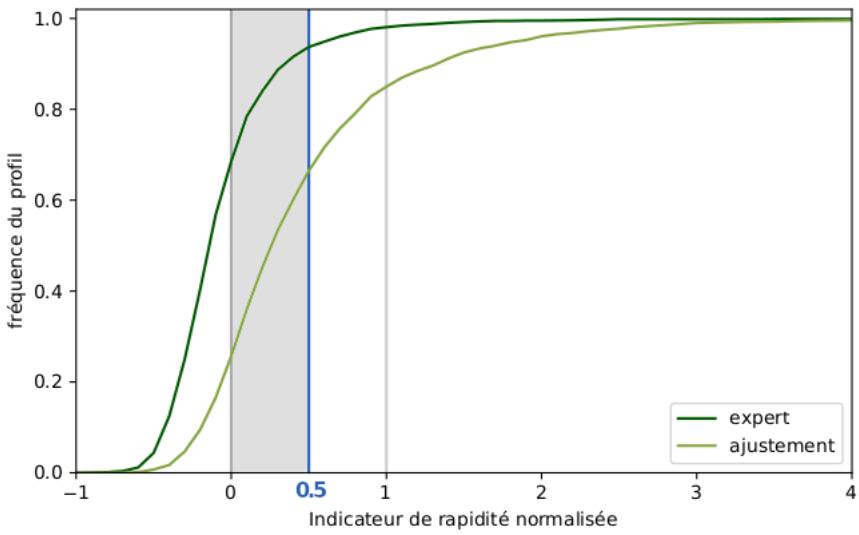


FIGURE 9.5 – Fréquence des profils expert (réussite en une seule tentative) et ajustement (réussite en deux ou trois tentatives) en fonction du seuil retenu pour l'indicateur de rapidité normalisée

De cette courbe, nous déduisons un intervalle de valeurs dans lequel nous situons le seuil recherché et concentrons nos analyses. La valeur 0 pour l'indicateur de rapidité normalisée équivaut à une résolution avec un temps égal au temps moyen des premiers essais du sujet. Lorsque la valeur de l'indicateur de

rapidité normalisée vaut 0,5 cela signifie que le sujet met 1,5 fois plus de temps à résoudre le puzzle que le temps moyen de ses premiers essais.

Nous avons fait varier le seuil entre 0 et 0,5 et étudié l'incidence sur la répartition des profils. Nous avons finalement retenu un seuil à 0,5. La fixation du seuil à cette valeur comporte certes une part d'arbitraire. Elle permet cependant un équilibre satisfaisant entre :

- le résidu des validations en 1 essai (profil indéterminé) qui ne sont pas catégorisées dans le profil expert
- une distinction entre les profils ajustement et accommodation concordante dans une majorité de cas avec nos observations sur les enregistrements vidéo

En effet, nous avons confronté pour chaque couple sujet/puzzle le profil déterminé automatiquement à partir d'un seuil à 0,5 avec le profil attribué suite à une analyse de l'enregistrement vidéo (pour 73 cas sur 1277, soit 5,7%, le profil reste indéterminé ou non pris en charge par la catégorisation). Nous obtenons une concordance de 98% pour le profil échec, 92% pour les profils expert et accommodation, 70% pour le profil ajustement. Le profil indéterminé calculé à partir du seuil correspond, sur la base de l'analyse vidéo, pour 29% à un profil expert, 6% à un profil ajustement, 50% à un profil accommodation et pour 15% à un cas non pris en charge par la présente catégorisation (notamment suspension temporaire de l'activité).

En considérant ce seuil à 0,5, nous visualisons un histogramme des profils identifiés selon la valeur de l'indicateur de rapidité normalisée (Figure 9.6). Ainsi, sur les 12340 confrontations de sujet à un puzzle de programmation, nous obtenons :

- 2819 occurrences (soit 22,8%) de profil *expert* : puzzle résolu en un seul essai et $R_i \leq 0.5$
- 1728 occurrences (soit 14,0%) de profil *ajustement* : puzzle résolu en 2 ou 3 essais et $R_i \leq 0.5$
- 4049 occurrences (soit 32,8%) de profil *accommodation* : puzzle résolu en 2 ou 3 essais et $R_i > 0.5$ ou résolution en plus de 3 essais.
- 2407 occurrences (soit 19,5%) de profil *échec* : puzzle non résolu, toutes les soumissions sont invalides.
- 1067 occurrences (soit 8,6%) de profil *passé* : puzzle ouvert mais pas de programme soumis.
- 270 occurrences (soit 2,2%) de profil *indéterminé* : un seul essai mais $R_i > 0.5$

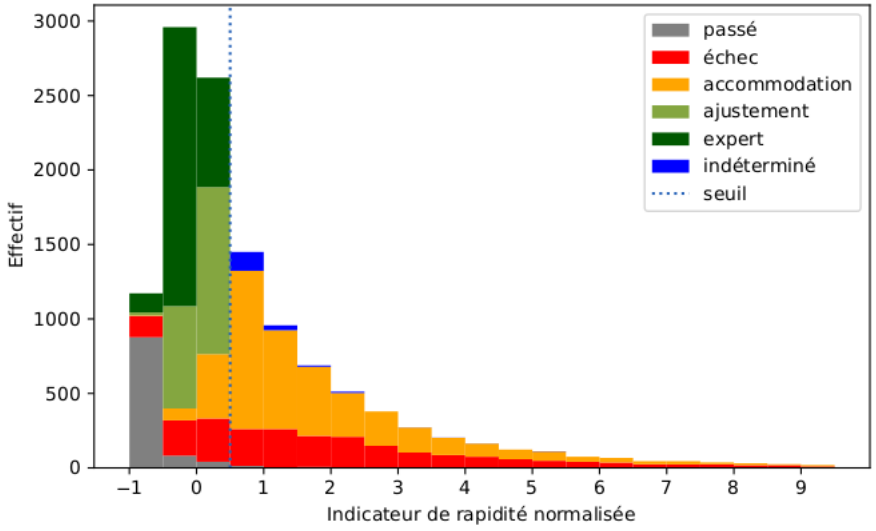


FIGURE 9.6 – Histogramme des profils avec un seuil de l'indicateur de rapidité normalisée à 0.5 pour la détermination des profils expert et ajustement

9.2.6 Synthèse

La contribution de cette partie porte sur la mise en relation d'éléments de la théorie des champs conceptuels de VERGNAUD avec l'analyse de traces d'interaction dans le but de quantifier le caractère immédiat ou non du traitement d'une situation et de caractériser automatiquement des profils dans le contexte de la résolution de puzzles dans un environnement de programmation par blocs. À cette fin :

> Nous avons calculé un indicateur de rapidité normalisée attaché à un couple sujet/puzzle indépendant de la dextérité du sujet et de la solution de référence pour le puzzle. Nous combinons cet indicateur avec le nombre d'essais et la validité ou non des programmes exécutés pour établir des profils relatifs au degré de compétence d'un sujet lors de la résolution d'un puzzle.

Dans la mesure où nous n'avons pris en compte que la validité des réponses, le temps de réponse et le nombre d'essais, indépendamment du contenu des puzzles, cette méthode pourrait être généralisée à des tâches pour lesquelles il est possible de définir une action élémentaire (ici placer un bloc dans l'éditeur). Nous notons que PELÁNEK a déjà procédé à ce type de généralisation pour des données de différentes natures (PELÁNEK, 2018). C'est aussi la raison pour

laquelle, pour ce traitement, nous utilisons des données plus larges que les seules traces issues des puzzles retenus pour l'étude du concept de motif. Cela nous permet de comparer la répartition des profils pour un puzzle particulier à la répartition pour l'ensemble des puzzles.

Associé à une identification précise des compétences en jeu dans chaque puzzle, nous pensons que cette détermination de profils sur la base de l'indicateur de rapidité normalisée et du nombre d'essais peut aider à automatiser la validation de compétences en programmation par blocs. Néanmoins, dans notre travail, les profils ont d'abord un statut d'outil. La démarche constitue une étape de nature à faciliter l'étude analytique des schèmes en jeu. Ces profils calculés automatiquement nous guident pour identifier les puzzles pour lesquels une difficulté particulière est rencontrée par un nombre significatif de sujets, ce qui nous permet de nous focaliser sur les données qualitatives correspondantes sans perdre de temps à analyser l'exhaustivité de nos données. Suite à notre analyse a priori du chapitre 11, nous mobilisons ces outils pour les puzzles retenus pour notre étude sur l'identification de motifs lors de l'initiation à la notion de boucle.

9.3 Échelle des classes : analyse quantitative à partir de l'enregistrement des programmes exécutés

Dans cette section, nous visons à définir une méthode pour investiguer le processus de résolution de manière plus précise, afin de mieux appréhender la conceptualisation-en-acte sous-jacente.

9.3.1 Appui sur le cadre théorique

La détermination automatique de profils expliquée dans la section précédente nous fournit une indication globale sur la manière dont un sujet a résolu (ou non) un puzzle. Cette démarche est intéressante, notamment parce qu'elle peut être mise en œuvre à large échelle et qu'elle présente un degré de généralité qui dépasse le cadre restreint des puzzles de programmation. Cependant, elle ne permet pas d'investiguer l'organisation invariante de l'activité, centrale dans le concept de schème. Nous déduisons seulement l'existence d'un schème unique organisateur de la conduite dans le cas des résolutions expertes. Or selon Vergnaud, « le concept de compétence appelle une analyse de l'activité en termes de "schèmes" » (Vergnaud, 2013b).

Afin d'approfondir l'analyse, nous nous appuyons sur l'extrait suivant de « La théorie des champs conceptuels » dans lequel Vergnaud explique comment le concept de schème est impliqué lorsque le sujet entre dans un processus

d'accommodation :

> Le concept de schème s'applique facilement à la première catégorie de situations [...], celles pour lesquelles le sujet dispose des compétences nécessaires, et moins à la seconde catégorie puisque le sujet hésite et tente plusieurs approches. Pourtant l'observation des élèves en situation de résolution de problème, l'analyse de leurs hésitations et de leur erreurs montrent que les conduites en situation ouverte sont également structurées par des schèmes. Ceux-ci sont empruntés au vaste répertoire des schèmes disponibles, et notamment à ceux qui sont associés aux classes de situations qui paraissent avoir une parenté avec la situation actuellement traitée. Simplement comme la parenté n'est que partielle et éventuellement illusoire, les schèmes sont seulement esquissés, et les tentatives souvent interrompues avant d'avoir été menées à leur terme ; plusieurs schèmes peuvent être évoqués successivement, et même simultanément dans une situation nouvelle pour le sujet (ou considérée par lui comme nouvelle). (Vergnaud, 1991, p.139-140)

Dans notre contexte, nous avons accès aux erreurs des sujets, à travers l'enregistrements des programmes exécutés. Nous retenons qu'analyser ces erreurs peut amener des éléments de compréhension de leur conceptualisation-en-acte. Un aspect intéressant est que nous disposons d'un volume conséquent de programmes soumis par les sujets. Nous sommes donc en mesure d'étudier, à travers leur fréquence, le caractère récurrent de certaines erreurs, et nous tenterons de les interpréter en termes d'organisation invariante de l'activité, c'est à dire de schèmes ébauchés ou mobilisés à mauvais escient.

Ainsi, à travers l'étude des programmes exécutés au cours des confrontations sujet/puzzle, nous cherchons à répondre aux questions suivantes : quelles sont les erreurs fréquentes ? Que signifient-elles en termes de conceptualisation-en-acte ? Que nous apprennent-elles sur la construction du champ conceptuel par le sujet ?

9.3.2 Appui sur la littérature en analyse de traces

Pour cette section, nous recherchons des travaux qui étudient les programmes exécutés.

Peláneк catégorise les erreurs de manière binaire en commune et non commune en définissant un seuil d'occurrences par rapport au total des réponses fausses (Peláneк, 2018). Il montre que cette catégorisation, bien que très sommaire, est utile pour la modélisation de l'apprenant. Peláneк donne une piste

pour aller plus loin dans cette classification des erreurs, en perdant le caractère général et au prix d'un travail d'expert important. Il donne des exemples de catégories possibles : représentation très erronée (*important misconception*), erreur type et erreur de calcul. Nous proposons d'établir une telle classification pour les erreurs relevées dans les programmes soumis par les élèves.

Une autre approche est celle de Gao et al. qui consiste à identifier des séquences d'événements redondants (Gao et al., 2021). Les auteurs ont procédé à partir d'enregistrements datés issus de l'environnements de programmation par blocs *Snap !* (106 étudiants novices en programmation). Ils utilisent une méthode de *machine learning* qu'ils nomment *differential sequence mining*. Les auteurs corrèlent ces séquences avec une catégorisation de l'expertise en programmation en faible niveau d'expertise et haut niveau d'expertise, qu'ils déterminent en utilisant la médiane des résultats au parcours de programmation. Ils relèvent ensuite les séquences les plus fréquentes dans chacun des deux groupes constitués, haut niveau et faible niveau d'expertise. Trois séquences relevées sont présentées :

- une séquence correspondant à l'utilisation d'un bloc de procédure avant de le compléter. Cette séquence révèle les compétences de décomposition et de représentation mentale de l'exécution, d'anticipation.
- une séquence alternant insertion d'un bloc et exécution du programme, qui correspond à une stratégie par essai-erreur, et qui est corrélée à un faible niveau d'expertise.
- une séquence correspondant au test de la fonctionnalité d'un bloc avant de l'utiliser. Cette séquence est reliée à un niveau faible d'expertise.

Dans la mesure où nous considérons que l'activité du sujet est constituée de ses interactions avec l'interface, nous retenons cette approche pour le caractère invariant et récurrent des séquences d'évènements qui nous semble pertinent pour révéler la présence de schème, dans son aspect organisation invariante de l'activité du sujet. Cela dit, mettre en œuvre cette approche à l'échelle des classes représenterait un travail trop conséquent dans le cadre de cette thèse. C'est pourquoi, nous mobilisons cette approche seulement pour des schèmes déjà repérés à l'échelle du sujet (9.4). Une perspective de travail ultérieur est d'automatiser pour pouvoir généraliser le repérage de séquences d'évènements.

9.3.3 Construction d'outils visant à analyser les programmes exécutés

À l'échelle des classes, nous disposons de tous les programmes exécutés par les sujets. D'un côté, comme mentionné par Peláne k (2018), travailler sur les

erreurs commises nécessite un travail d'expert conséquent. D'un autre côté, le volume conséquent de données exclut une analyse exhaustive des programmes à la main. Il s'agit donc de trouver un compromis entre analyse d'expert et automatisation du traitement des données. Nous expliquons les étapes de ce travail, au cours duquel traitements automatisés et analyses d'expert s'articulent. Le détail des traitements est consigné dans l'annexe 8.

Dans une première étape, nous construisons une table qui répertorie tous les programmes exécutés pour chaque puzzle. Nous réalisons ensuite une première catégorisation de ces programmes, de manière automatisée. Enfin, nous construisons une visualisation qui synthétise les principales informations.

9.3.3.1 Table des programmes prototypiques

Lors de l'exécution d'un programme, l'enregistrement contient le programme exécuté, mais aussi sa position dans l'éditeur, et éventuellement des blocs détachés qui y ont été laissés.

Une question concerne le statut de ces blocs détachés. D'un côté, ils peuvent être utiles à l'analyse au niveau individuel car ils constituent une trace des manipulations effectuées avant l'exécution du programme. D'un autre côté, ils perturbent la fusion des enregistrements de solutions identiques.

Afin de réduire le nombre d'entrées dans la table, nous choisissons de normaliser les programmes, c'est à dire de supprimer les éventuels blocs détachés et de replacer le programme aux coordonnées d'origine de l'éditeur. Nous appelons *programme prototypique* un tel programme et ce sont ces programmes que nous conservons comme entrées dans la table documentant les programmes exécutés. Nous gardons toutefois la référence au programme initial dans un champ de cette table, afin d'être en mesure de mener des analyses plus précises à l'échelle du sujet.

Dans les paragraphes suivants, nous détaillons les champs de cette table des programmes prototypiques :

Caractérisation d'un programme par rapport à un puzzle

L'analyse d'un programme exécuté est attachée à la résolution d'un puzzle. Un même programme peut éventuellement avoir été soumis sur des puzzles différents, mais il peut ne pas être considéré de la même manière sur ces différents puzzles. Par exemple, il peut constituer une solution valide pour un puzzle et une erreur pour un autre. Trois champs déterminent de manière unique le statut d'un programme exécuté pour la résolution d'un puzzle :

- L'identifiant du programme prototypique

- Le nom du problème pour lequel le programme a été soumis
- La version de ce problème

Des indications concernant la fréquence de ce programme prototypique

- Le nombre d'occurrences pour l'ensemble des programmes collectés au cours de l'étude (échelle des classes et échelle du sujet)
- La proportion que ce programme recouvre, respectivement par rapport à l'ensemble des programmes valides ou invalides.
- Parmi les utilisateurs qui ont soumis un programme pour ce puzzle, la fréquence des utilisateurs ayant soumis un programme correspondant à ce programme prototypique.

Les indicateurs calculés dans la section précédente

Nous considérons toutes les confrontations sujet/puzzle qui contiennent le programme soumis et nous calculons :

- la moyenne et l'écart type de l'indicateur de rapidité normalisée
- la moyenne et l'écart type du nombre d'essais
- la répartition des profils (expert/ajustement/accommodation/indéterminé dans le cas d'un programme valide, ajustement/accommodation/échec dans le cas d'un programme invalide)

Le lien avec l'échelle du sujet

Afin d'approfondir l'analyse en vue de la documentation des schèmes identifiés, nous renseignons un champ qui contient, lorsqu'il y en a, les identifiants des sujets suivis individuellement qui ont soumis le programme considéré, avec leur profil attaché à cette confrontation.

Nous conservons aussi un champ avec la liste des identifiants de programmes initiaux, afin de pouvoir nous y référer si besoin.

9.3.3.2 Une première catégorisation réalisée de manière automatisée

La catégorisation des programmes soumis vise l'identification et la caractérisation des schèmes mobilisés par les sujets. Cette catégorisation pose des questions en termes de méthodologie. En effet le volume important de programmes soumis, plusieurs dizaines de milliers à l'échelle des classes, nécessite d'automatiser le traitement, mais le travail d'expert reste indispensable pour identifier ces catégories. L'objet de cette section est de mettre en place une automatisation pour alléger l'important travail d'expert mentionné par Pelánek (2018).

L'enjeu de cette catégorisation est multiple :

- Identifier les solutions valides qui ont été soumises de manière alternative à la solution de référence
- Séparer les programmes incomplets des véritables erreurs. En effet, ce qui se joue au niveau cognitif n'est pas de même nature : règle de vérification pour les programmes incomplets, théorème-en-acte erroné pour les erreurs.

Nous réalisons une première catégorisation des programmes prototypiques de la manière suivante :

- Solution optimale : solution de référence ou programme valide qui a le même nombre de blocs que la solution de référence
- Solution non optimale : programme valide mais dont le nombre de blocs est strictement supérieur à celui de la solution de référence
- Programme incomplet qui constitue le début d'une solution optimale
- Programme incomplet qui constitue le début d'une solution non optimale
- Erreur

Nous intégrons cette catégorisation comme champ de la table des programmes prototypiques dont nous avons détaillé la construction. Concernant les programmes erronés, ils font en plus l'objet d'une table dédiée, avec une catégorisation plus précise, qui est présentée plus loin.

9.3.3.3 Une visualisation synthétique des programmes fréquemment soumis

Avec les données de la table des programmes prototypiques, nous construisons une visualisation des programmes exécutés, sur la base de leur nombre d'occurrences. L'enjeu de ce graphique est de visualiser rapidement les programmes les plus fréquemment soumis pour un puzzle donné (exemple figure 9.7).

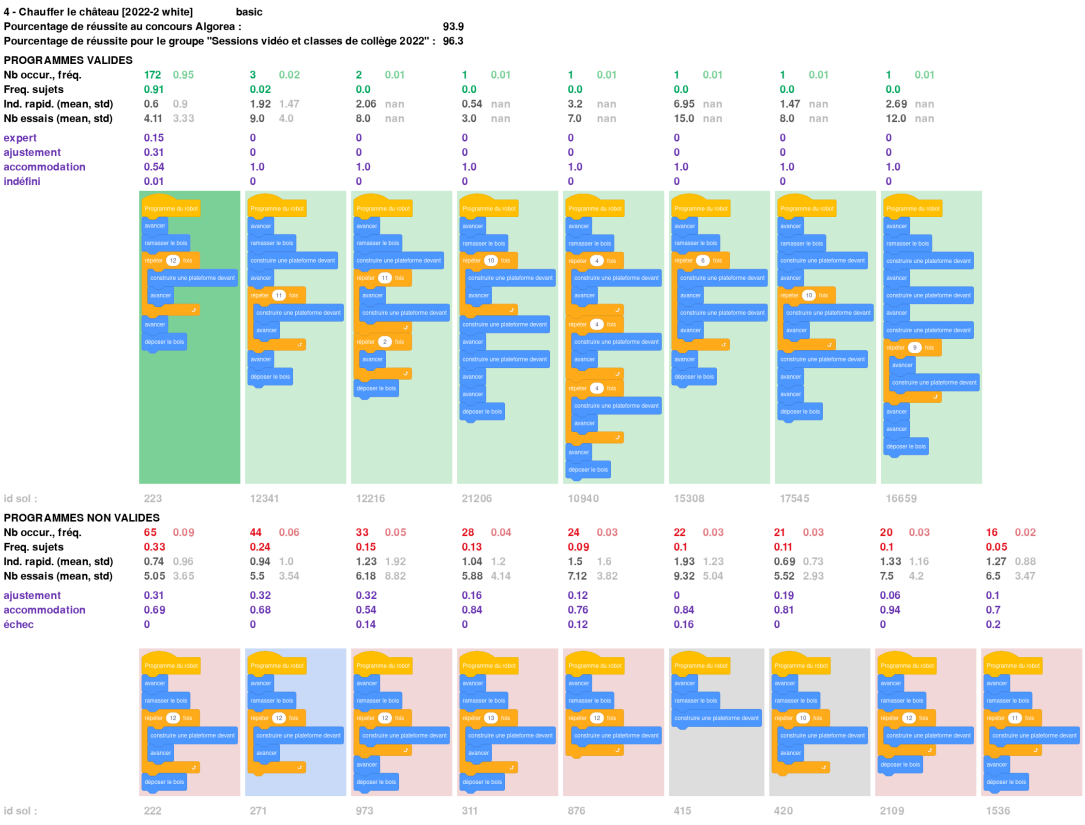


FIGURE 9.7 – Exemple de visualisation des programmes soumis par ordre décroissant de fréquence

Nous séparons les programmes valides et les programmes erronés. Les programmes valides sont en haut de la figure. La solution de référence ainsi qu'éventuellement les solutions qui comportent le même nombre de blocs (appelées solutions optimales) sont sur fond vert soutenu. Les autres solutions valides, non optimales, sont sur fond vert clair.

Pour les programmes non valides, nous distinguons les débuts de solution optimale sur fond bleu clair, les débuts de solution non optimale sur fond gris clair et les erreurs sur fond rosé.

Pour chaque catégorie, programmes valides et programmes non valides, nous montrons les programmes soumis par ordre décroissant de fréquence d'apparition. Nous accompagnons chaque image de programme en langage Scratch des informations complémentaires suivantes :

- le nombre d'occurrences pour le groupe considéré
- la proportion que ce programme recouvre, respectivement par rapport à

l'ensemble des programmes valides ou à l'ensemble des erreurs.

- pour toutes les confrontations sujet/problème qui contiennent le programme soumis :
  - la moyenne et de l'écart type de l'indicateur de rapidité normalisée
  - la moyenne et de l'écart type du nombre d'essais
  - la répartition des profils

À partir de cette visualisation, nous sommes en mesure de répondre aux questions suivantes pour chaque puzzle :

- Quelle est la répartition des profils pour la solution de référence ?
- Quelles sont les solutions alternatives valides ? Quelle est leur fréquence ?
- Certaines erreurs sont-elles corrélées à un profil de résolution ?
- Quelles sont les erreurs qui amènent plus fréquemment à un échec ?

À l'échelle des classes, cette visualisation constitue une étape importante dans notre processus d'analyse de la conceptualisation-en-acte. En effet, en fonction de la réponse aux questions précédentes, nous n'étudions pas tous les puzzles de la même manière. Notamment, les puzzles pour lesquels nous repérons des erreurs qui amènent fréquemment à un échec, et qui indiquent un blocage, sont étudiés en priorité pour identifier le raisonnement sous-jacent.

9.3.4 Création d'une table documentant les erreurs commises

Nous nous intéressons de manière plus approfondie aux programmes erronés, dans la mesure où nous considérons les erreurs comme une conceptualisation-en-acte non adaptée. Nous concentrons donc notre travail d'expert spécifiquement sur ces programmes erronés, que nous cherchons à caractériser au-delà du contexte d'un puzzle particulier. À nouveau, nous présentons les étapes que nous suivons afin de rendre compatible ce travail d'expert avec le volume important de données.

9.3.4.1 Caractérisation manuelle des erreurs les plus fréquemment commises

Pour chaque puzzle, nous générons une visualisation des 20 programmes de type *erreur* les plus fréquents. Cela correspond aux programmes sur fond rosé sur la visualisation précédente.

Nous étudions manuellement chacun de ces programmes. Cette analyse consiste à caractériser le type d'erreur commise, et à déterminer des indicateurs qui permettent d'identifier ce type d'erreur automatiquement à partir des programmes au format xml.

Nous avons deux types d'indicateurs :

- un premier indicateur est le nombre de blocs total, ou le nombre de blocs d'un type particulier
- un autre indicateur est un sous-programme qui doit faire partie du programme enregistré, et qui est caractéristique de l'erreur commise.

Dans certains cas, nous pouvons combiner ces deux types d'indicateurs.

9.3.4.2 Automatisation de la caractérisation des erreurs pour l'ensemble des programmes erronés

En utilisant ces indicateurs construits à partir des 20 programmes erronés les plus fréquemment enregistrés, nous trouvons automatiquement les autres programmes, moins fréquents mais qui présentent les mêmes caractéristiques.

Ce processus nous permet de construire une table des types d'erreur commises.

Cette table comprend d'une part des champs qui spécifient le puzzle et d'autre part des champs qui décrivent le type d'erreur :

- un label qui caractérise le type d'erreur
- la fréquence des sujets qui ont commis ce type d'erreur sur ce puzzle, parmi les sujets qui ont soumis au moins un programme pour ce puzzle
- le nombre de sujets auquel cette fréquence correspond
- le nombre de programmes prototypiques dans lesquels a été trouvé le type d'erreur
- la liste des identifiants de ces programmes prototypiques afin de pouvoir y accéder rapidement en cas de besoin de vérification ou d'analyse plus approfondie
- la liste des identifiants des sujets qui ont commis ce type d'erreur et pour lesquels nous possédons un enregistrement vidéo de la session, en vue d'établir le lien avec l'échelle du sujet

Cette manière de procéder évite l'investissement excessif en temps que demanderait une analyse exhaustive, tout en préservant le travail d'expert nécessaire pour entrer dans une analyse approfondie des erreurs commises. Nous

ne sommes pas certains de répertorier ainsi la totalité des erreurs commises. Cependant, nous sommes en mesure de donner une bonne approximation de la fréquence d'apparition pour les erreurs les plus communes.

De plus, le type d'erreur est dans la plupart des cas indépendant du contexte narratif (aspect visuel des éléments de la grille) du puzzle.

9.4 Échelle du sujet : mise en correspondance des traces d'interaction et de l'analyse qualitative des enregistrements vidéo d'écran

L'objectif du niveau d'analyse à l'échelle du sujet est de cerner finement, par des analyses qualitatives, ce qui se passe lorsque celui-ci est en action. Plus précisément, nous visons l'identification et la documentation des schèmes mobilisés par le sujet lors de sa confrontation à un puzzle de programmation. Les données dont nous disposons à cette échelle pour mener à bien nos investigations sont constituées des traces d'interaction, des enregistrements vidéo d'écran, et des verbalisations collectées en situation et a posteriori.

9.4.1 Appui sur le cadre théorique

Dans cette section, nous prenons appui sur la définition analytique d'un schème énoncée par Vergnaud :

> Le schème est formé nécessairement de quatre composantes :
>
> - un but, des sous buts et anticipations ;
> - des règles d'action, de prise d'information et de contrôle ;
> - des invariants opératoires : concepts-en acte et théorèmes en acte ;
> - des possibilités d'inférence en situation.
>
> (Vergnaud, 2013a, p.138)

Identifier quels schèmes les sujets mobilisent lors de la confrontation à des puzzles de programmation est de nature à nous donner accès à leur conceptualisation. En effet, « le schème n'organise pas que la conduite observable, mais également l'activité de pensée sous-jacente » (Vergnaud, 2013a, p.138)

En analysant les enregistrements vidéo d'écran et les verbalisations du sujet, nous visons de renseigner les schèmes mobilisés lors des procédures expertes, mais aussi et surtout à identifier les conceptualisations erronées sous-jacentes aux erreurs fréquemment commises.

9.4.2 Appui sur la littérature

Nous cherchons dans la littérature des travaux dont le protocole expérimental s'appuie sur des enregistrements vidéo d'écran, éventuellement combinés avec d'autres natures de données, afin d'analyser le comportement de sujets. Nous mobilisons d'abord des travaux assez généraux qui nous semblent présenter des caractéristiques intéressantes du point de vue méthodologique, puis des travaux plus ciblés sur l'initiation à la programmation informatique.

McMillan et al. décrivent les enregistrement vidéo d'écran comme des données qui, au niveau micro, prennent en charge l'analyse des interactions avec un système informatique via son interface. Les auteurs, dont les expérimentations concernent des appareils mobiles, triangulent les données avec d'autres sources telles que des fichiers journaux pour recouper les enregistrements d'écran avec des informations complémentaires (McMillan et al., 2015). Nous retenons qu'une utilisation de ces données supplémentaires est de réduire le nombre de vidéos à analyser. Les auteurs donnent l'exemple des données de localisation, qui permettent de cibler l'analyse vidéo sur les seuls endroits qui présentent un intérêt pour l'analyse.

Si nous contextualisons à notre étude, les enregistrements datés et la visualisation sous forme de frise chronologique que nous générons à partir de ces données constituent un moyen d'accéder directement aux extraits de l'enregistrement vidéo d'écran que nous souhaitons analyser.

Krieter propose une méthode pour cumuler la densité d'informations que permet une analyse manuelle des enregistrements vidéo d'écran à petite échelle avec le passage à l'échelle que permet la collecte de traces d'interactions alliée à une automatisation des traitements. Cette méthode s'appuie sur des techniques de vision artificielle et d'apprentissage machine. Elle consiste à définir des évènements, qui sont ensuite recherchés automatiquement dans les enregistrements vidéos, et collectées dans un fichier journal en vue d'une analyse quantitative (Krieter, 2020). Nous retenons de cette étude la visée de passage à l'échelle des analyses réalisées à partir des enregistrements vidéo d'écran.

En plus de ces deux études de portée générale, nous nous focalisons sur des travaux qui concernent l'initiation à la programmation et qui combinent des natures différentes de données, dont des enregistrements vidéo d'écran.

Grover et al. combinent l'analyse de traces d'interaction et l'analyse d'enregistrements vidéo pour étudier le comportement de 53 élèves de niveau lycée lors de tâches ouvertes dans l'environnement de programmation par blocs *Alice* (Grover et al., 2017). Les auteurs montrent l'intérêt de cette approche hybride pour approfondir la compréhension de la construction des compétences en programmation. Ils identifient 13 compétences dont *Incorporate repetition of blocks of code using looping constructs*, qui restent à un niveau assez global. Dans cette

étude, les auteurs ne creusent pas la manière dont se construit chaque compétence, ce que nous nous sommes fixé comme objectif pour l'identification de motif lors de l'introduction de la notion de boucle bornée.

Deux autres études concernent plus spécifiquement la résolution de puzzles de programmation, dans l'environnement code.org. Çakiroğlu et Mumcu combinent enregistrements vidéo d'écran, observations en situation et entretiens à l'issue de l'activité pour étudier le comportement de 15 élèves lors de la résolution de puzzles (Çakiroğlu & Mumcu, 2020). La récurrence des comportements amènent les auteurs à définir trois étapes, *focus*, *fight and finalize*, lors de la résolution de ces puzzles. Xu et al. étudient le *pair programming* en combinant enregistrement vidéo des participants, enregistrements vidéo d'écrans et journal de logs, lors de la résolution de puzzles de programmation. L'expérimentation implique 19 binômes de niveau lycée (Xu et al., 2023).

Les études précédentes dans le domaine de l'initiation à la programmation avec des langages par blocs s'appuient sur des expérimentations à l'échelle de quelques dizaines de participants. Elles correspondent à ce que nous projetons de réaliser à l'échelle du sujet. Notre apport se situe sur l'articulation des analyses à cette échelle avec celles réalisées sur les mêmes puzzles, mais à une échelle beaucoup plus large. L'enjeu est de donner une robustesse statistique à des résultats trouvés à petite échelle.

9.4.3 Traitements et analyses mis en œuvre

À l'échelle du sujet, nous disposons des mêmes traces d'interaction qu'à l'échelle des classes, et des données supplémentaires que sont les enregistrements vidéo des sessions et des moments de *debrief*. Les enregistrements vidéo d'écran permettent d'analyser finement le comportement du sujet. Lorsque ce comportement est accompagné de verbalisation, nous accédons à une explicitation de celui-ci qui nous permet d'appréhender la conceptualisation-en-acte. Le lecteur trouvera un aperçu du travail d'analyse manuelle des enregistrements vidéo dans l'annexe 9. Lorsque le comportement du sujet est récurrent, ou qu'on le retrouve chez un nombre significatif de sujets, nous cherchons à identifier le ou les schèmes mis en œuvre.

Cependant, les traitements manuels des enregistrements vidéo étant très chronophages, nous mettons en place une stratégie pour n'analyser que les extraits dans lesquels il est le plus probable que nous trouvions des éléments intéressants pour nos analyses.

Dans un premier temps, nous générons les frises de la session des sujets dont nous avons enregistré l'écran lors de la session du concours Algorea. Grâce à ces frises, nous découpons les vidéos en extraits qui correspondent aux confronta-

tions aux puzzles que nous avons retenus pour notre étude sur l'identification de motifs, afin de pouvoir y accéder rapidement par la suite.

Nous nous appuyons ensuite sur les analyses à l'échelle des classes, réalisées à partir des traces d'interaction. Parmi les programmes erronés les plus fréquents, nous regardons si nous disposons de données qualitatives, c'est à dire si parmi les sujets qui ont commis l'erreur, certains font partie de l'échantillon que nous suivons à l'échelle du sujet.

Nous avons conçu une fonction qui permet de générer une visualisation de la suite des programmes soumis par un sujet à un puzzle particulier. Lorsque nous rencontrons une erreur fréquente qu'un sujet suivi individuellement a commise, nous générons et étudions la suite des programmes qu'il a exécuté lors de la confrontation à ce puzzle. Nous mettons en regard cette visualisation avec l'enregistrement vidéo d'écran correspondant. Pour cet enregistrement vidéo, nous transcrivons les verbatim et décrivons les actions du sujet sur l'interface (manipulation des blocs, mouvements de souris).

La mise en correspondance avec les traces d'interaction vise à identifier des séquences de programmes exécutés qui caractérisent le schème, et qui pourraient être retrouvées dans les traces d'interaction indépendamment des enregistrements vidéo, afin d'automatiser la recherche de schèmes mobilisés.

Pour la session complémentaire de bilan, nous disposons également de l'enregistrement des mouvements de souris suite à la collaboration avec Maxime Bouton, stagiaire de master 2 en sciences des données de l'université de Lille que nous avons encadré. Notre stratégie consiste à construire une fonction qui génère une visualisation des mouvements de souris entre deux *timestamp*, en mettant en évidence les temps d'arrêt. En effet, nous considérons ces temps d'arrêt comme indicateur de la focalisation de l'attention sur la zone de l'interface.

La visualisation des mouvements de souris permet d'avoir accès à des actions habituellement visibles seulement sur les enregistrements vidéo d'écran. Cette modalité se rapproche de celle de Krieter (2020), dans le sens où elle permet de passer à des traitements quantitatifs sans analyse qualitative préalable. Dans le cadre de cette étude, nous avons disposé de l'enregistrement des mouvements de souris seulement en fin de collecte, et il n'a pas été possible de la mettre en place à l'échelle des classes. Ce serait maintenant techniquement possible.

9.4.4 Synthèse

Cette étude des schèmes mobilisés pour chaque classe de situations définie a priori amène à reconsidérer la catégorisation des classes de situations, à l'affiner.

On recherche des indicateurs qu'il serait possible de retrouver systématiquement dans les traces, afin d'automatiser l'identification de schèmes, c'est à dire enrichir le niveau d'analyse précédent. Les difficultés constatées à cette échelle

et leur analyse pourront venir expliquer les taux de réussite constatés à large échelle.

La succession des programmes soumis, bien qu'ayant une granularité assez fine, reste une collecte de données discrètes. Avec l'enregistrement vidéo, nous passons à un flux continu d'informations du point de vue de la manipulation de l'interface (mouvements de souris, déplacement et suppression de blocs..), qui de plus est accompagné des verbalisations du sujet. Nous sommes ainsi mieux en mesure d'identifier les schèmes mobilisés lors de la résolution du puzzle et donc d'appréhender plus finement la conceptualisation-en-acte du sujet.

9.5 Échelle du sujet : catégorisation des situations et analyse quantitative des verbatim

Dans cette section, nous présentons le traitement des verbatim issus de la tâche de catégorisation des situations, proposée lors de l'expérimentation complémentaire de l'été 2022 à l'échelle du sujet.

9.5.1 Appui sur le cadre théorique

Vergnaud préconise de prendre en compte le point de vue subjectif du sujet, notamment à travers l'étude des invariants opératoires, concepts-en-acte et théorèmes-en-acte que le sujet mobilise dans une classe de situations. Les invariants opératoires sont appréhendés du point de vue du sujet, « ils pilotent la reconnaissance par le sujet des éléments pertinents de la situation, et la prise d'information sur la situation à traiter » (Vergnaud, 1991, p.159).

Selon Tijus et Cordier, les tâches de catégorisation informent sur le développement cognitif (Tijus & Cordier, 2003). Elles font appel au processus de généralisation. Dans notre contexte, la tâche de catégorisation vise à identifier les propriétés prises en compte pour l'analyse de la situation de programmation.

La tâche consiste à identifier des similarités entre les différentes grilles présentées. Afin de prendre des points de repère pour notre analyse, nous consultons des travaux sur le raisonnement par analogie, dans lesquels la détection de similarité est un aspect important.

Nogry passe en revue un ensemble de recherches sur le raisonnement par analogie (Nogry, 2020). Pour la catégorisation de problèmes, nous retenons la distinction entre des *traits de surface* et des *traits de structure*. Les traits de surface concernent l'habillage des problèmes, ils sont inutiles dans le traitement de la situation. Au contraire, les traits de structure sont les propriétés qu'un expert identifie pour résoudre la classe de problèmes.

Pour Hofstadter et Sander, le développement conceptuel est intimement lié à l'évolution de la catégorisation des situations auxquelles est confronté le sujet (Hofstadter & Sander, 2013). Ces catégorisations admettent des critères qui vont du concret vers l'abstrait, du spécifique vers le général en établissant des relations d'inclusion. La manière dont les sujets catégorisent des situations est donc un indicateur de leur niveau d'expertise par rapport aux situations données. Pour un novice les traits saillants, qui orientent le traitement de la situation, sont essentiellement les traits de surface, alors que ce sont les traits de structure chez un expert.

Nous nous appuyons sur cette distinction entre traits de surface et traits de structure pour analyser les critères donnés par les élèves lors de la tâche de catégorisation.

9.5.2 Traitements et analyses mis en oeuvre

Nous menons des investigations quant aux critères avancés par les sujets afin de les interpréter en termes de traits de surface et de traits de structure. Nous nous demandons ensuite s'il existe une corrélation entre les critères proposés par les sujets et les résultats sur le parcours, en tenant compte du niveau de classe.

Nous prenons comme unité d'analyse une planche contenant six grilles issues des parcours de programmation et nous organisons nos données en conséquence dans une table. Pour chaque planche et pour chaque sujet, nous relevons les critères avancés que nous regroupons en catégories interprétables en termes de traits de surface et de traits de structure.

Une perspective est de traiter les verbatim avec la bibliothèque Python NLTK, avant d'établir une catégorisation automatique par une méthode de *clustering*.

9.6 Aspects techniques de la mise en œuvre des traitements

Après les exports de la base de données réalisés avec des requêtes en langage SQL, l'ensemble du traitement et de l'analyse des données est réalisé en langage Python, en utilisant des bibliothèques spécifiques :

- Numpy (version 1.24.3) et Pandas (version 1.4.4) : traitement des données
- Matplotlib (version 3.6.2), Seaborn (version 0.12.1) et Pygame (2.1.2) : visualisation des données
- Scipy.stats (1.10.1) : statistiques

- Scikit learn (1.1.3) : *machine learning*

Nous avons développé une bibliothèque Python de traitement et de visualisation des données en lien avec ce travail de recherche, déjà initiée lors de notre travail de master et régulièrement maintenue et enrichie depuis.

Nous avons collaboré avec deux stagiaires de master 2 en science des données que nous avons co-encadrés, et qui ont travaillé à l'amélioration et l'extension de nos outils. Une architecture de traitement des données a été mise en place en collaboration avec Gabriel Loiseau. Maxime Bouton a amélioré le code des fonctions de nettoyage des données et de certains traitements. Il a ensuite travaillé sur l'adjonction d'un modèle de *machine learning* et sur la visualisation des mouvements de souris.

Nous collaborons par ailleurs avec les développeurs de l'association France-ioi, notamment avec Michel Blockelet qui a conçu un script permettant de générer automatiquement dans un navigateur l'image d'un programme Scratch à partir de son code au format XML. Cette génération d'image *a posteriori* est à la base de l'ensemble des visualisations de notre bibliothèque Python.

Nous effectuons l'ensemble de nos traitements de données dans des *notebook Jupyter*, dans lesquels nous importons toutes ces bibliothèques.

9.7 Synthèse

Dans ce chapitre, nous avons explicité la stratégie que nous mettons en place pour traiter le volume conséquent de données que nous avons collecté. Nous avons détaillé comment nous articulons les traitements et analyses entre les différentes échelles de collecte et entre les différentes natures de données.

Notre protocole de traitement et d'analyse s'appuie conjointement sur le cadre d'analyse de la théorie des champs conceptuels et sur des travaux plus récents en analyse de traces. Nous avons montré comment nous combinons les apports respectifs de ces travaux afin d'étudier la résolution de puzzles de programmation avec un langage par blocs. D'un côté, la théorie des champs conceptuels nous donne des clés pour structurer l'analyse de nos données et les interpréter. Réciproquement, la possibilité d'articuler des traitements statistiques sur de larges échantillons avec la précision d'observation d'une approche qualitative est de nature à apporter une robustesse aux résultats établis en se plaçant dans ce cadre d'analyse. Plus précisément, nous avons montré comment l'automatisation du traitement de données est à la fois le support d'analyses quantitatives et le moyen d'accès rapide à des extraits pertinents pour l'analyse qualitative.

Le tableau 9.1 récapitule l'ensemble des traitements et analyses mis en œuvre dans le cadre de cette étude. Des pictogrammes en indiquent la nature :

- traitement ou analyse manuelle
- traitement ou analyse automatisée

Étape du processus	Échelle nationale	Échelle des classes	Échelle du sujet
Collecte des données	• Entre 5000 et 92 400 participants au concours officiel Algorea sur concours.castor-informatique.fr Score et effectif – par niveau de classe – depuis 2018 : 5 éditions, 139 problèmes	• 248 élèves de collège pour l'édition 2022, 115 pour l'édition 2021 sur chti-code.algorea.org enregistrements datés de toutes les soumissions de programme dans la base de données bebras-chticode	• Entre 12 et 19 élèves du CP à la 3ème suivis pour les éditions 2021 et 2022 enregistrement vidéo des sessions + debrief • session complémentaire avec : – nombre de blocs non limité – set de blocs disponibles plus large enregistrement des mouvements de souris dans la base de données srl activité complémentaire de catégorisation de grilles
Preprocessing • extraction des données • nettoyage des données • pré-traitements • mise en forme	extraction des données dans le périmètre de la recherche (langage Scratch, classes du CM1 à la 3ème) avec une fonction python tag des puzzles : caractéristiques des grilles, notions algorithmiques mobilisées enregistrement des solutions de référence au format xml agrégation de l'ensemble des données dans une table	extraction des données via des requêtes SQL fonction de nettoyage des données brutes : suppression des enregistrements non utiles ou redondants appariement des sessions d'un même sujet	découpage des enregistrements par puzzles description de l'activité du sujet sous forme textuelle transcription des verbatim extraction des données via des requêtes SQL (mouvements de souris) mise en forme de ces extractions via des fonctions python (Maxime Bouton)
Construction d'indicateurs	% de réussite pour chaque puzzle et chaque niveau de classe	temps passé sur chaque puzzle et nombre de soumissions de programme indice de rapidité normalisée pour la résolution d'un puzzle (Léonard et al., 2023) fréquence des programmes exécutés	séquences d'évènements redondants temps d'arrêt du curseur de souris sur l'interface
Génération de visualisations (fonctions python)	Courbes d'évolution du % de réussite en fonction du niveau de classe	frise chronologique de la session d'un sujet visualisation des programmes exécutés par ordre décroissant de fréquence visualisation de la suite des programmes soumis à un puzzle	visualisation des mouvements du pointeur de souris sur l'interface reconstituée (Maxime Bouton)
Analyses qualitatives	travail théorique sur le concept de motif (Léonard, Secq et al., 2022)	identification de profils à partir des frises (Léonard et al., 2023) catégorisation des erreurs	identification et documentation des schèmes
Traitements de statistique descriptive	échelle dans la difficulté des puzzles (Léonard, Secq et al., 2022)	détermination automatique de profils qui caractérisent la confrontation sujet/puzzle analyse quantitative des programmes soumis	traitement des données textuelles avec la bibliothèque python NLTK (perspective)
Adjonction de modèle de machine learning (apprentissage statistique, visée prédictive)	Ébauche de traitement en utilisant les modèles classiques de machine learning présentés dans le MOOC FUN de l'INRIA	modèle de self attention pour la prédiction des profils identifiés (Bouton et al., 2022)	clustering à partir des données textuelles (perspective)

Tableau 9.1 – Synthèse des traitements et analyses mis en œuvre

Quatrième partie

Résultats et analyses

Notre quatrième partie regroupe quatre chapitres dans lesquels nous présentons nos résultats et les discutons. Le chapitre 10 contient une étude instrumentale de l'environnement de programmation Algoréa. Dans le chapitre 11, nous réalisons un travail théorique sur le concept de motif, qui sert d'appui à nos résultats à l'échelle nationale. Nous détaillons nos résultats à propos des premiers apprentissages de la boucle dans le chapitre 12. Enfin, le chapitre 13 traite de l'identification et du dénombrement de motifs lors de la programmation d'une boucle bornée.

Chapitre 10

Étude instrumentale de l'IEAH

Sommaire du présent chapitre

L'activité de programmation au sein des environnements de programmation par blocs relève de la classe des activités instrumentées au sens de Rabardel (1995). Mener à bien une tâche de programmation dans ce type d'environnement requiert d'en avoir pris en main au moins les fonctionnalités de base. L'objectif de ce chapitre est d'identifier la composante instrumentale de l'activité de programmation dans notre contexte, afin de pouvoir ensuite la mettre à distance ou la prendre en compte dans l'étude des genèses conceptuelles qui est le cœur de notre travail. Nous cherchons notamment à répondre à la question de recherche formulée dans le chapitre 6 :

> Quels sont les schèmes d'usage attachés à l'environnement de programmation par blocs qu'il est nécessaire de construire afin d'être en mesure de résoudre les puzzles proposés ?

Nous fondons notre analyse sur les éléments théoriques de l'approche instrumentale introduits dans le chapitre 5. Dans une première section, nous identifions et analysons les pôles de la situation d'activité instrumentée que constitue la confrontation avec un puzzle de programmation dans l'environnement Algoréa. Parmi les composantes artefactuelles de l'environnement de programmation, la souris est celle sur laquelle le sujet agit directement. Nous identifions les genèses instrumentales qui lui sont attachées, et montrons l'apport pour notre recherche de l'analyse des mouvements de souris. Nous détaillons ensuite la construction du langage de programmation comme instrument, ainsi que la construction de la représentation de quelques éléments du dispositif informatique par les sujets ayant participé à l'étude, en relevant quelques difficultés rencontrées. Nous illustrons notre propos par des extraits vidéos de confrontation avec des puzzles. L'ensemble des extraits vidéos utilisés dans ce document de thèse, ainsi que le travail d'analyse préalable associé est disponible dans l'annexe 9.

10.1 Identification des pôles de la situation instrumentale

Du point de vue de l'approche instrumentale, l'EIAH que constitue l'environnement de programmation Algoréa est caractérisé par une position interne de l'objet de l'activité dans l'artefact (5.2). La tâche du sujet, explicite dans l'énoncé de chaque puzzle, est de programmer le robot virtuel pour qu'il effectue une certaine mission sur la grille (rejoindre une case cible, ramasser, déposer, déplacer des objets). Pour réussir la tâche, le sujet doit donc agir sur des objets (le

robot virtuel, les éléments de la grille via le robot). Mais ces objets ne sont pas accessibles directement, car ils se trouvent sur l'interface, au sein de l'EIAH. Le fait que l'objet de l'activité ne soit pas manipulable directement est constitutif de l'activité de programmation en tant que *faire faire* à une machine. Kalas et al. parlent de *manipulation indirecte*, caractéristique de la programmation, par opposition à une *manipulation directe* lorsqu'il est possible de déplacer les objets présents sur l'écran par glisser-déposer (Kalas et al., 2018). Cette tâche de *programmer* implique donc de constituer l'EIAH comme instrument.

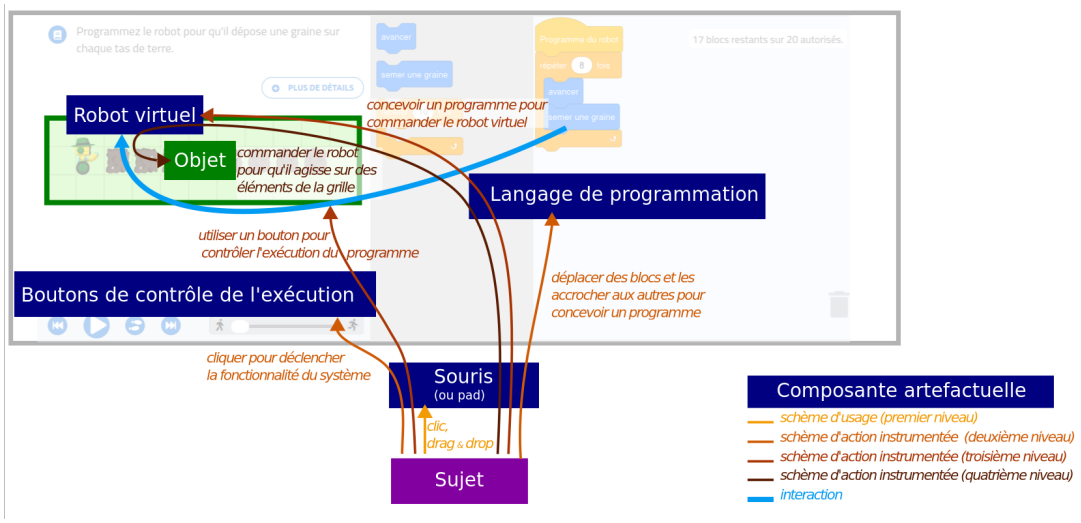


Figure 10.1 – Composantes artefactuelles de l'environnement de programmation Algoréa

L'environnement de programmation Algoréa est un artefact que nous pouvons qualifier de complexe. Nous identifions quatre composantes artefactuelles : la souris, le langage de programmation, le robot virtuel et les boutons de contrôle de l'exécution (Figure 10.1). Trois de ces composantes artefactuelles sont imbriquées les unes dans les autres d'une part, et deux le sont d'autre part. Dans le cas d'un écran tactile, nous avons une composante artefactuelle en moins, puisque le sujet agit sur les blocs du langage de programmation et sur les boutons de contrôle de l'exécution directement avec son corps (doigt).

Les trois composantes artefactuelles imbriquées sont impliquées dans la conception du programme : la souris permet d'agir sur les blocs de langage qui, accrochés les uns aux autres pour former un programme, permettent d'agir sur le robot virtuel, qui, aux yeux du sujet, agit sur les éléments de la grille.

Les composantes artefactuelles ont un statut d'objet lors de leur prise en main, puis un statut d'instrument. Nous sommes en présence d'un système

de schèmes, qui comporte jusqu'à quatre niveaux hiérarchiques de schèmes d'action instrumentée. Pour chaque niveau, ces schèmes d'action instrumentée incorporent à titre de schème d'usage ceux des niveaux précédents (Figure 10.2). Il est à noter que, sur la figure 10.1, la composante artefactuelle langage de programmation englobe deux niveaux de schèmes d'action instrumentée : déplacer les blocs pour concevoir un programme, et concevoir un programme pour commander le robot virtuel.

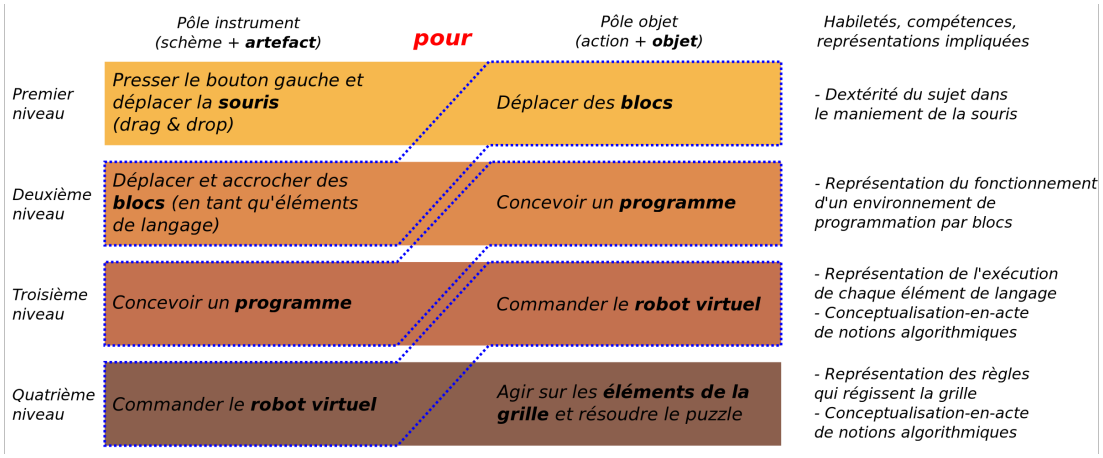


FIGURE 10.2 – Système hiérarchique de schèmes

En plus de la conception du programme, le sujet agit aussi avec sa souris pour contrôler l'exécution de celui-ci, via des boutons en bas à gauche de l'interface. Ces boutons de contrôle de l'exécution constituent la quatrième composante artefactuelle de l'EIAH. Si nous adoptons un regard externe au fonctionnement du système informatique (considéré dans ce cas comme une boîte noire), ces boutons permettent de gérer l'interaction entre la composante artefactuelle *langage de programmation (programme)* et l'objet *robot*.

Outre les schèmes d'utilisation, la prise en main des composantes artefactuelles de l'EIAH implique de construire des représentations pour l'action. L'activité représentative, que nous retrouvons à la fois lors de la construction de la signification des éléments de langage, mais aussi lors de la construction de la représentation de l'effet des boutons de contrôle de l'exécution, est essentielle dans la constitution de l'instrument (5.3.3).

Dans les sections qui suivent, nous détaillons ce qui est en jeu pour chaque composante artefactuelle, en termes de schèmes et de représentations pour l'action. La constitution des instruments pour les différents niveaux du système hiérarchique de schèmes se fait de manière concomitante à la résolution des puzzles du parcours. C'est pourquoi nous visons d'identifier ce qui, dans l'ac-

tivité du sujet, relève de la construction des instruments, par rapport à ce qui relève de leur mise en œuvre et des premières conceptualisations de notions de programmation. L'objectif est de mettre à distance l'activité en lien avec la construction de schèmes d'usage (trois premiers niveaux sur la figure 10.2) afin de pouvoir nous concentrer sur les schèmes d'action instrumentée du langage de programmation (troisième et quatrième niveau sur la figure 10.2).

10.2 Apport méthodologique de la souris constitué comme instrument

Dans notre dispositif expérimental, bien que nous n'ayons pas imposé de matériel particulier, la grande majorité des sujets suivis à l'échelle du sujet a utilisé un ordinateur avec une souris. Cette modalité représente 97 des 105 sessions de la catégorie blanche et de l'expérimentation complémentaire enregistrées en vidéo. Pour cette section, nous écartons de notre analyse les quelques sessions où le sujet a utilisé un pavé tactile (7 sessions) ou une tablette (1 session).

10.2.1 Pourquoi analyser les mouvements de souris ?

Dans le système de schèmes, ceux qui se rapportent à la souris ont un statut particulier dans le sens où ils se trouvent au niveau hiérarchique le plus bas. La souris est l'artefact matériel sur lequel le sujet agit directement. Nous montrons comment une part de notre méthode d'analyse est basée sur l'activité du sujet avec sa souris.

Lorsqu'il résout un puzzle dans l'environnement de programmation Algoréa, le sujet tient la souris en main. Celle-ci a le statut d'objet *saisi*, par opposition à un objet *lâché/déplacé* (Lenay et al., 2002). Elle constitue le prolongement de la main, c'est à dire qu'une fois constituée comme instrument, le sujet ne perçoit plus sa souris, mais perçoit l'interface à travers sa souris.

Dans la constitution de la souris comme instrument, les deux composantes de la genèse instrumentale sont présentes. En effet, nous identifions des processus qui sont de l'ordre de l'instrumentation et d'autres qui relèvent de l'instrumentalisation.

L'utilisation prévue de la souris pour déplacer les blocs ou activer les boutons de contrôle de l'exécution requiert des schèmes d'usage, le glisser-déplacer et le clic, qui sont généraux à l'utilisation d'un ordinateur et relèvent de l'instrumentation. La dextérité du sujet dans le maniement de la souris révèle l'efficacité de ces schèmes d'usage, dont nous n'étudions pas la genèse, trop périphérique à notre travail de recherche. Cependant, comme expliqué dans le chapitre 9, nous

avons été amenée à neutraliser l'effet de la dextérité du sujet avec la souris, afin de pouvoir comparer entre elles les confrontations sujet/puzzle.

Outre ces schèmes relevant du processus d'instrumentation, nous identifions, sur les enregistrements vidéo d'écran, une utilisation de l'artefact souris comme instrument de pointage. Le schème *Pointer un objet pour le désigner ou pour accompagner le raisonnement* s'applique à de nombreux artefacts, lorsqu'ils sont saisis et constituent un prolongement de la main. Dans notre contexte, utiliser la souris pour pointer une zone de l'interface, constitue une extension de ce même schème de pointage réalisé avec le doigt. La souris est utilisée pour adapter le schème de pointage à la situation d'activité sur une interface. Nous sommes dans le cas d'un schème déjà constitué associé à un nouvel artefact. Cette utilisation de la souris relève du processus d'instrumentalisation.

Nous relevons plusieurs schèmes d'action instrumentée qui incorporent ce schème de pointage dans notre contexte :

- Suivre le texte avec le curseur de souris lors de la lecture de l'énoncé ou du tutoriel
- Pointer les éléments à compter lors du schème de dénombrement (4.4)
- Pointer des cases de la grille lors de l'identification d'un motif
- Pointer les cases de la grille pour accompagner la simulation mentale de l'exécution d'un programme
- Pointer des cases de la grille et/ou des blocs lors de l'analyse d'une exécution
- Désigner des éléments de l'interface afin de soutenir une question à l'expérimentatrice

L'utilisation de la souris comme instrument de pointage met en évidence la zone de l'interface sur laquelle est focalisée l'attention du sujet. En ce sens, nous pouvons dire que le curseur de souris visible à l'écran suit le raisonnement du sujet, qu'il révèle une part de sa conceptualisation-en-acte. C'est pourquoi, l'analyse des déplacements du curseur de souris sur l'interface nous permet de soutenir notre propos. Dans la suite, pour alléger l'écriture, nous synthétisons *déplacements du curseur de souris sur l'interface* par *mouvements de souris*.

10.2.2 Quelques exemples représentatifs d'analyse de mouvements de souris

Les mouvements de souris sont directement observables sur les enregistrements vidéo d'écran. Suivre ces mouvements de souris sur l'interface nous renseigne sur la focalisation de l'attention du sujet, ce qui nous procure des

données précieuses sur sa conceptualisation-en-acte. Cependant, pour pouvoir être analysés, les mouvements de souris demandent une description minutieuse et chronophage, de l'ordre de 10 à 12 minutes de travail pour une minute d'enregistrement vidéo. Cette analyse manuelle n'est donc possible qu'à très petite échelle.

En conséquence, nous nous sommes demandé quelle part de ce travail il est possible d'automatiser, dans une perspective de passage à l'échelle. Nous avons évoqué les traitements réalisés dans le chapitre 9. En particulier, nous repérons les moments où le sujet pointe un élément de l'interface, qui sont marqués par un temps d'arrêt du curseur de souris. Nous avons conçu une visualisation des mouvements de souris sur laquelle ces temps d'arrêt sont mis en évidence par des disques de taille proportionnelle à la durée de l'arrêt.

Nous montrons quelques visualisations qui correspondent aux schèmes d'action instrumentée relevés précédemment. Le dégradé de couleur marque la chronologie du mouvement : le début du mouvement est en violet et la fin est en rouge. Ces visualisations ont été confrontées aux enregistrements vidéo d'écran correspondants, pour vérification.

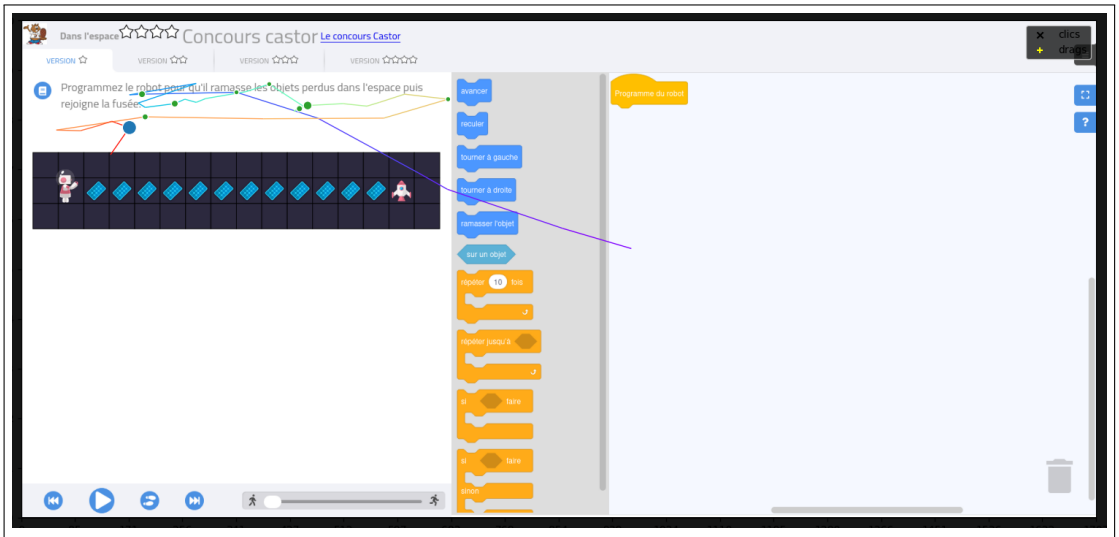


FIGURE 10.3 – Mouvements de souris pendant la lecture de l'énoncé

La figure 10.3 montre les huit premières secondes de la confrontation d'Alix avec le puzzle 2022t4p3v1, pendant lesquelles le mouvement du curseur de souris accompagne la lecture de l'énoncé. Dans l'enregistrement vidéo d'écran correspondant, Alix lit à voix haute et marque un temps d'arrêt plus long à la fin de la lecture (point bleu sur la figure 10.3), avant de commencer la résolution du puzzle (Alix, CM2, 2022t4p3v1).

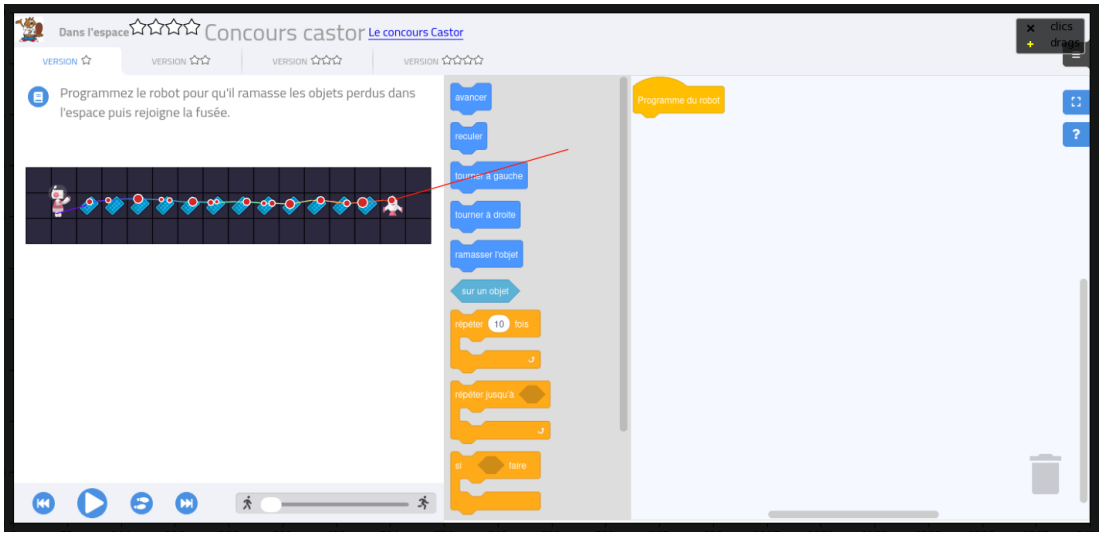


FIGURE 10.4 – Pointage des éléments avec la souris lors d'un schème de dénombrement

La figure 10.4 met en évidence le pointage des éléments avec la souris lors d'un schème de dénombrement, qui est aussi visible sur l'enregistrement vidéo d'écran (Louna, 6ème, 2022t4p3v1) : après avoir placé un bloc *répéter* dans l'éditeur, Louna survole la grille avec sa souris en marquant un temps d'arrêt sur chaque élément, puis va aussitôt renseigner le nombre de répétitions dans ce bloc *répéter*.

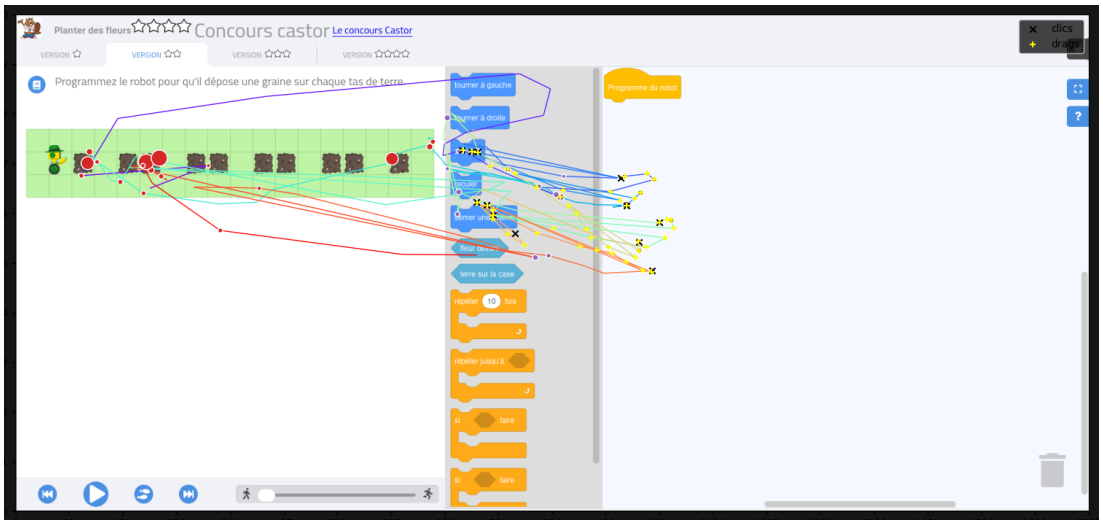


FIGURE 10.5 – Prise de repères sur la grille lors de l'édition du motif

La figure 10.5 montre la prise de repères sur la grille lors de l'identification du motif. Sur l'enregistrement vidéo de cette session, Alix fait des allers-retours avec sa souris entre la grille et la zone d'édition, où elle est en train d'éditer le corps de la boucle. Elle accompagne verbalement l'identification du motif sur la grille, « On est là » en même temps qu'elle pointe l'endroit avec sa souris (points rouges sur les première et troisième zones de terre) (Alix, CM2, 2022t4p1v2).

Ces quelques exemples donnent un aperçu de la manière dont il est possible, grâce à la visualisation des mouvements de souris, de mettre en évidence une part de l'activité cognitive au cours de la confrontation avec un puzzle de programmation. Ces visualisations nous renseignent en particulier sur les phases de prise d'information, phases qui sont indétectables avec une granularité de collecte au niveau des programmes exécutés.

L'enjeu est de générer les visualisations de mouvements de souris qui viennent en soutien de nos analyses de la conceptualisation-en-acte du sujet.

10.3 Construction du langage de programmation comme instrument

Le langage de programmation, artefact symbolique, est une composante artefactuelle constitutive de tous les environnements de programmation. Son apprentissage est d'ordre instrumental (5.3.4).

Dans le cas des environnements par blocs, ce langage de programmation est constitué de blocs colorés mis à disposition dans une zone dédiée de l'interface. Agir sur ces blocs de programmation consiste à les déplacer par glisser-déposer dans une autre zone, la zone d'édition, pour les accrocher les uns aux autres. Sauf dans le cas d'une interface tactile, cette activité requiert elle-même un instrument : souris ou pavé tactile. Ce mode d'action, avec ses schèmes d'usage associés, est une spécificité des environnements par blocs par rapport aux environnements de programmation textuelle. En effet, l'instrument qui permet d'agir est différent, souris, doigt ou pavé tactile pour le langage par blocs, clavier pour le langage textuel.

Concevoir un programme pour commander le robot virtuel suppose d'une part de construire la signification et la représentation de l'exécution de chaque bloc du langage de programmation, mais aussi d'appréhender la manière de les accrocher les uns aux autres (règle de syntaxe). Ces représentations sont constitutives du schème d'usage du langage de programmation, qui devient l'instrument de représentation de l'algorithme. La construction de ce schème d'usage implique de construire de manière concomitante une représentation de la machine, plus précisément du fonctionnement du système informatique.

L'importance de la représentation du dispositif informatique lors de l'activité de programmation a été mise en évidence par Rogalski pour les langages de programmation textuels (Rogalski, 1988b).

Dans le cas des puzzles dans un environnement de programmation par blocs, la mise en œuvre de l'instrument consiste à choisir des blocs et les agencer de manière adéquate afin de faire exécuter la mission demandée au robot virtuel. La conception d'un programme comprend une composante gestuelle et une composante symbolique qui sont inséparables : accrocher les blocs les uns aux autres (activité gestuelle) et exprimer un algorithme dans un langage de programmation (activité symbolique). L'activité gestuelle peut être réalisée sans activité symbolique sous-jacente, dans le cas d'une stratégie par essai-erreur. L'activité symbolique peut avoir lieu sans donner lieu à une activité gestuelle, c'est à dire que le sujet peut concevoir mentalement un algorithme sans l'exprimer dans le langage de programmation. Mais la résolution de la tâche implique d'associer activité symbolique et activité gestuelle. Pour cette raison, sauf dans le cas d'une pure stratégie par essai-erreur, la manipulation des blocs révèle une part de l'activité symbolique sous-jacente.

10.3.1 La genèse instrumentale de l'aspect syntaxique du langage de programmation

La prise en main de l'environnement de programmation pour concevoir un programme comprend le déplacement et l'accrochage de blocs dans l'éditeur, indépendamment de la pertinence du programme conçu. Pour étudier la genèse instrumentale de l'aspect syntaxique du langage de programmation, nous nous intéressons à la première confrontation avec l'environnement de programmation Algoréa pour les sujets que nous avons suivi individuellement. Sur ces 23 sujets, nous disposons d'un enregistrement vidéo de cette première confrontation pour 18 d'entre eux. Dans la suite, nous renvoyons à des extraits de ces enregistrements vidéos d'écran pour étayer notre propos.

Lors de la découverte de l'interface, la plupart des sujets commencent par l'explorer. Cette exploration est révélée par les mouvements du curseur de souris (Louane, CM2, 2022t1p1v1).

La tâche *concevoir un programme avec les blocs mis à disposition* implique plusieurs schèmes et représentations pour l'action.

- *Déplacer un bloc par glisser-déposer* : schème d'usage de l'artefact souris (ou pavé tactile, ou directement doigt pour un écran tactile). L'efficacité de ce schème est lié à la dextérité du sujet dans le maniement de la souris (ou du pavé tactile). Dans la suite, nous ne mentionnons plus que la souris (sauf

besoin spécifique), qui a été l'artefact utilisé dans la grande majorité des cas.

- *Accrocher les blocs les uns aux autres pour concevoir un programme* : schème d'action instrumentée de l'artefact souris qui incorpore à titre de schème d'usage le déplacement par glisser-déposer. La construction de ce schème implique plusieurs représentations pour l'action, qui sont des schèmes d'usage de la composante artefactuelle *langage de programmation par blocs* :
  - *Les blocs dans la zone sur fond gris sont des éléments de langage*. Pour quelques sujets, ce repérage a nécessité d'être fortement étayé (Ugo, CM1, 2022t1p1v1).
  - *Les blocs de langage sont des éléments déplaçables*. Pour des sujets qui n'ont jamais été confronté à un environnement de programmation par blocs, cela ne relève pas de l'évidence. Certains repèrent les commandes sans savoir comment s'en servir (Robin, CM1, 2022t1p1v1). 12 des 18 sujets dont nous possédons la vidéo de la première session Algoréa ont eu besoin de regarder le tutoriel disponible, qui montre comment déplacer et accrocher des blocs les uns aux autres. Regarder ce tutoriel suffit à provoquer la construction de la représentation nécessaire (Lou, CM2, 2021t1p1v1), sauf pour 3 sujets. Cependant, nous notons que seulement 4 sujets ont ouvert le tutoriel spontanément, les autres l'ayant fait sur incitation de l'expérimentatrice, ce qui pose question sur l'efficacité de la mise à disposition de ce tutoriel.
  - *Un programme est un agencement de blocs accrochés verticalement les uns aux autres*. Une fois les blocs repérés comme étant les commandes pour actionner le robot, la question de la manière de s'en servir se pose. Nous avons relevé plusieurs représentations spontanées inadaptées : amener le bloc sur le bloc *Programme du robot*, (Alix, CM1, 2021t1p1v1) amener le bloc au-dessus de la grille, sur la case du robot ou la case cible (Tom, CE1, 2021t1p1v1) , (Ali, CM1, 2021t1p1v1). La forme des blocs incite à les accrocher, mais l'axe vertical entre en conflit avec le sens habituel de la lecture, horizontal de gauche à droite, ce qui amène quelques sujets à disposer les blocs horizontalement (Sam, 6ème, 2022t1p1v1).

10.3.2 La construction de la signification pour quelques éléments de langage

Quasiment simultanément à la construction des schèmes qui permettent de concevoir un programme en accrochant des blocs les uns aux autres, la prise en main du langage de programmation implique de construire la signification

de chaque bloc, c'est à dire une représentation de l'exécution de ce bloc par la machine. Pour la plupart des blocs, cette représentation correspond à la représentation spontanée qu'en a le sujet, et la prise en main ne pose aucun problème. En revanche, quelques blocs d'action sont plus difficiles à prendre en main, parfois même pour des experts. Nous passons en revue quelques difficultés couramment rencontrées, en nous appuyant sur des extraits de confrontations à l'échelle du sujet, ainsi que sur des taux d'erreur à l'échelle des classes.

Les blocs *tourner à droite* et *tourner à gauche*

Les blocs *tourner à droite* et *tourner à gauche* font partie des blocs mis à disposition dans tous les puzzles en orientation relative. Ils sont donc introduits dès les premiers problèmes des parcours (p1 ou p2).

La construction de la représentation de l'exécution de ces blocs se heurte à plusieurs difficultés :

- Dans le langage courant *tourner* signifie *décrire une ligne courbe en avançant*. Cette acception associe déplacement vers l'avant et pivotement. Or l'action engendrée par l'exécution des blocs *tourner à droite* et *tourner à gauche* est un pivotement seul, ce qui remet en cause la représentation de ces blocs qu'ont spontanément les sujets, représentation issue du sens courant du mot *tourner*. L'erreur se manifeste dans le programme conçu par l'absence d'un bloc de déplacement avant ou après un bloc *tourner*.
- La confusion entre les deux types d'orientation amène à construire *se déplacer d'une case vers la droite* au lieu de *pivoter à droite* comme représentation de l'exécution du bloc *tourner à droite* (idem pour *tourner à gauche*). À nouveau, cette erreur se manifeste par l'absence d'un bloc de déplacement dans le programme conçu (Lou, CM2, 2021t1p1v2).
- La vue de côté du robot (distorsion de la perspective par rapport à la grille vue de dessus) en même temps que la position verticale de l'écran amène certains sujets à chercher un bloc avec le label *descendre* (Tom, CE1, 2021t1p1v2), (Alix, CM1, 2021t1p1v2).
- Une difficulté supplémentaire est présente lorsque le robot est vu de face. En effet, il convient de se placer du point de vue du robot. Cela provoque une erreur dans le sens de pivotement (Ugo, CM1, 2022t1p2v1), (Gina, CE1, 2022t1p2v1).

À partir des traces, il est impossible de différencier les trois premiers cas de représentation spontanée car l'erreur dans le programme est la même. De plus, l'erreur qui consiste à dénombrer les cases vides au lieu des déplacements du robot produit aussi le même type de programme erroné où il manque un déplacement. Nous pouvons seulement affiner pour l'erreur de sens de pivotement

du robot (figure 10.6). Cette erreur est persistante, surtout si un pivotement lorsque le robot est de face est à gérer (en bleu foncé sur la figure). La fréquence de cette erreur reste élevée pour certains puzzles, l'erreur concernant parfois plus de 10% des sujets, alors que l'expertise globale augmente au fur et à mesure de l'avancée dans le parcours, ainsi que d'un tour sur l'autre.

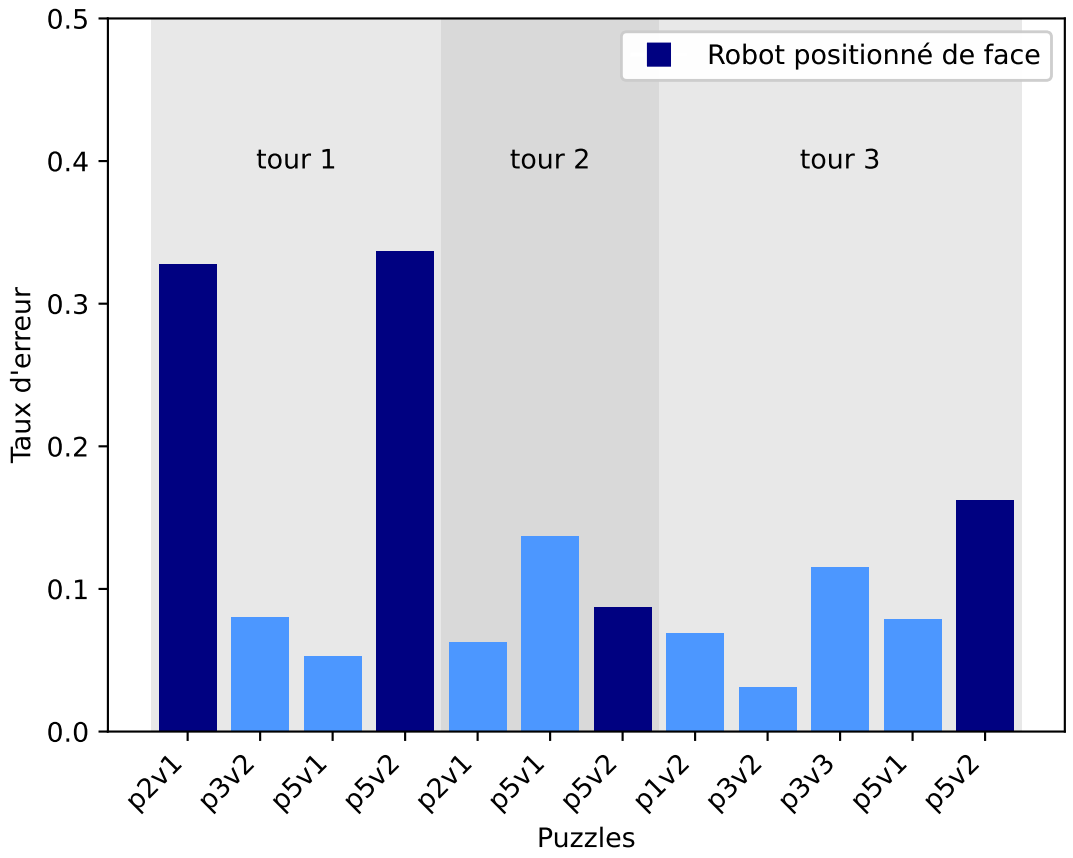


FIGURE 10.6 – Taux d'erreur dans le sens de pivotement du robot pour les puzzles du concours 2022

Le bloc *sauter* du contexte *Chauffer le château*

Le bloc *sauter* est introduit dans le puzzle 2022t1p1v2. 154 des 198 sujets qui ont exécuté un programme pour ce puzzle, soit environ 78%, ont commis une erreur qui concerne la prise en main du bloc *sauter*. Nous identifions plusieurs sources d'erreur dans la construction de la représentation de ce bloc. La figure 10.7 montre la grille, ainsi que quelques programmes erronés représentatifs qui ont été soumis.

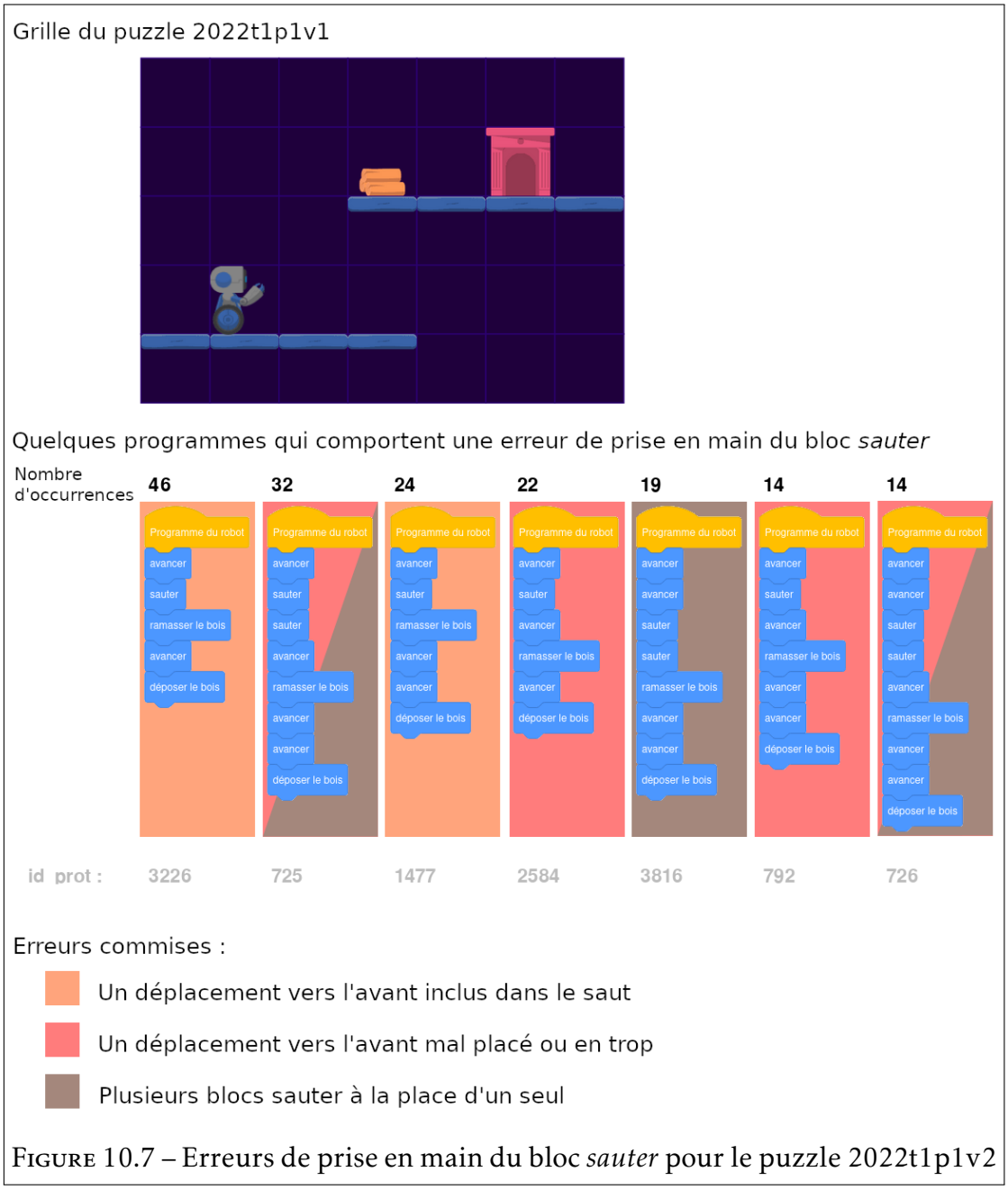


FIGURE 10.7 – Erreurs de prise en main du bloc *sauter* pour le puzzle 2022t1p1v2

- Dans les programmes sur fond orange, un déplacement vers l'avant est inclus dans le saut. Ce déplacement vers l'avant en sautant correspond à la motricité humaine, c'est donc une représentation intuitive, mais qui ne correspond pas à l'action provoquée par le bloc *sauter*. À l'échelle du sujet, nous repérons cette représentation dans le geste avec la souris qui accompagne parfois la simulation de l'exécution (Colin, 6ème, 2022t1p1v2)

- Dans les programmes sur fond marron, deux blocs sont utilisés à la place d'un seul. L'exécution du bloc *sauter* est en effet en contradiction avec la règle implicite en vigueur pour les autres blocs, qui est *un bloc provoque le déplacement du robot d'une seule case*. Or, dans ce contexte, les plateformes sont espacées de deux cases verticalement, et un seul bloc *sauter* fait passer le robot d'un étage à celui du dessus, c'est à dire de deux cases vers le haut. Par exemple, Colin qui a placé trois blocs *sauter* consécutifs dans son programme le justifie a posteriori par « Moi je croyais qu'il fallait sauter trois cases de suite ».
- Lorsqu'un bloc *avancer* est mal placé (programmes sur fond marron), il peut signifier que le sujet ne prend pas en compte la gravité, qui est une caractéristique spécifique à ce contexte.

Le bloc *pousser la caisse* du contexte *Ranger les caisses*

FIGURE 10.8 – Erreur la plus fréquemment commise pour le puzzle 2022t3p3v1

L'erreur la plus fréquemment commise pour le puzzle 2022t3p3t1 est de placer deux instructions dans le corps de la boucle, *pousser la caisse* et *avancer* au lieu du seul bloc *pousser la caisse* (Figure 10.8). En agrégeant les programmes qui comportent cette erreur mais diffèrent sur un autre aspect (nombre d'itérations, ordre des deux instructions), nous calculons que 110 des 132 sujets qui ont soumis au moins un programme sur le puzzle, soit 83%, ont commis au moins une erreur de ce type.

L'enregistrement vidéo de la session de Timéo offre un exemple représentatif de la prise en main de ce bloc *pousser la caisse* (Timéo, 5ème, 2022t3p3v1). Malgré une lecture de l'intégralité de l'énoncé à voix haute, énoncé dans lequel il est bien précisé que le robot avance d'une case en même temps qu'il pousse la caisse, Timéo place le bloc *avancer* dans le corps de la boucle en plus du bloc *pousser la caisse*. Après le premier essai raté, sa réaction est d'inverser l'ordre de ces deux instructions et de lancer une nouvelle exécution du programme. C'est après cette exécution qu'il a une démarche de construction de la représentation du bloc dont le début est marqué verbalement par « Alors..pourquoi ça marche.. » de manière concomitante à la suppression du bloc *répéter*. Timéo utilise dans cette phase le mode pas à pas afin de visualiser l'exécution de chaque bloc. C'est ce qui

lui permet de trouver que le bloc *pousser la caisse* provoque l'exécution de deux actions, avancer et pousser la caisse. Après avoir validé le puzzle, il exprime verbalement la modification de sa représentation de l'exécution par « Ah ok, donc il n'y a pas besoin d'avancer en plus de pousser la caisse ».

L'exécution du bloc *pousser la caisse* entre en conflit avec un théorème-en-acte construit lors de la confrontation avec les puzzles précédents, *un bloc code une seule action du robot*, théorème-en-acte qui est mis en défaut dans ce cas puisque le robot avance en poussant la caisse.

Le bloc *construire une plateforme devant* du contexte *Chauffer le château*

FIGURE 10.9 – Erreur la plus fréquemment commise pour le puzzle 2022t2p4v1

Le bloc *construire une plateforme devant* est introduit dans le puzzle 2022t2p4v1. L'erreur la plus commise pour ce puzzle est à nouveau liée à la prise en main du bloc nouvellement introduit : 102 des 186 sujets qui sont confrontés à ce puzzle, soit environ 55%, mettent seulement le bloc *construire une plateforme devant* à l'intérieur de la boucle.

La confrontation d'Alix avec ce puzzle nous renseigne sur le raisonnement sous-jacent (Alix, CM2, 2022t2p4v1). Avant même de lancer une exécution, Alix demande « Quand ils disent construire une plateforme devant est-ce que je dois mettre avancer ? ». Lors d'une première exécution, elle utilise le bloc *avancer* mais le place avant le bloc *construire une plateforme devant*. Elle déduit du feedback que le déplacement est pris en charge par le bloc *construire une plateforme devant* et teste donc avec ce seul bloc dans la boucle. Lors de la tentative suivante, elle replace un bloc *avancer* dans la boucle, cette fois de manière adéquate après le bloc *construire une plateforme devant*, et construit donc la bonne représentation de l'exécution de ce bloc.

Plus de la moitié des sujets font l'hypothèse que l'action d'avancer est prise en charge par le bloc *construire une plateforme devant*, comme pour le bloc *pousser la caisse*. Or la règle sous-jacente pour l'exécution est différente pour ces deux blocs. Dans un cas, le déplacement du robot est pris en charge par le bloc d'action, alors que dans l'autre cas, le déplacement n'est pas géré par le bloc d'action. Le fait que la règle diffère selon les blocs constitue une difficulté de prise en main pour les sujets car une règle dans la représentation de l'exécution construite pour un bloc ne peut pas être réinvestie pour construire la représentation de l'exécution d'un autre bloc.

Les trois blocs précédents sont spécifiques à un contexte. L'exécution de ces

blocs entre en concurrence avec la représentation spontanée qu'en a le sujet, qui provient soit de sa motricité propre, soit d'un théorème-en-acte construit auparavant. Les sujets raisonnent en termes d'actions du robot virtuel et non en termes de fonctions préprogrammées qui est une représentation d'experts.

Le bloc *sauter* est introduit dans le premier problème du parcours. Sa prise en main n'interfère donc pas avec notre étude sur le concept de motif. En revanche, les blocs *pousser la caisse* et *construire une plateforme devant* sont respectivement introduits dans la version 1 des problèmes 3 et 4, puzzles où sont aussi introduits des notions qui concernent la boucle. La prise en main de ces deux derniers blocs interfère donc avec notre progression pour l'apprentissage de la boucle, et nous devrons en tenir compte lors de l'analyse de ces puzzles dans les chapitres suivants. Nous pourrons aller jusqu'à prendre la décision d'écarter certains puzzles de nos analyses si ces difficultés masquent les aspects que nous souhaitons étudier.

Ces difficultés de prise en main sont perceptibles à l'échelle nationale à travers un taux de réussite de 3 à 5 points de pourcentage plus faible, et surtout à l'échelle des classes à travers une moyenne de l'indicateur de rapidité normalisée plus élevée que pour les puzzles des autres tours équivalents dans le parcours.

10.4 Construction de la représentation de quelques éléments du dispositif informatique, complémentaires au langage de programmation

Du point de vue des sujets de notre étude, le cœur du dispositif informatique (notamment l'exécution du programme par le processeur de l'ordinateur) est une boîte noire. La part du dispositif informatique qui est à la portée de ces jeunes sujets est constituée de l'exécution des blocs de langage qu'ils manipulent, de l'exécution du programme qu'ils peuvent contrôler et visualiser sur l'interface. Une large part du fonctionnement du système est dissimulée, encapsulée dans les blocs ou dans l'image générée de la grille. En termes de transparence opérative (5.2), le dispositif opère une simplification du système informatique de manière à apporter aux jeunes sujets seulement les informations dont ils ont besoin, sous une forme qu'ils sont capables d'interpréter.

Dans la section précédente, nous avons analysé la constitution comme instrument du langage de programmation, qui est un élément majeur du dispositif informatique. Nous complétons par une analyse de la prise en main des boutons de contrôle de l'exécution, puis par une analyse de la construction de quelques représentations de règles qui régissent les actions du robot virtuel sur la grille.

10.4.1 La prise en main des boutons de contrôle de l'exécution

Après la conception d'un programme pour faire faire la mission demandée au robot virtuel, une deuxième tâche, tout aussi nécessaire, est d'*exécuter le programme conçu*. Cette tâche nécessite de construire des schèmes et des représentations pour l'action associés aux boutons de contrôle de l'exécution, que nous avons identifiés :

- *Presser le bouton gauche de la souris, autrement dit cliquer* : schème d'usage de l'artefact souris, la difficulté principalement rencontrée étant le clic sur le bouton droit (qui fait apparaître un menu contextuel) au lieu du gauche.
- *Lancer l'exécution du programme complet en cliquant sur le bouton play* : le sujet doit associer le déclenchement de l'exécution du programme avec le geste de cliquer sur le bouton. Lorsque cette représentation n'est pas construite, le sujet peut s'attendre à ce que les blocs s'exécutent au fur et à mesure qu'il les accroche (Timéo, 5ème, 2022t1p1v1). Sur le plan conceptuel, la prise en main du bouton *play* est concomitante de la construction du caractère différé de l'exécution du programme.
- *Lancer l'exécution d'une seule instruction en cliquant sur le bouton du mode pas à pas* : un clic sur ce bouton provoque d'une part l'exécution de l'instruction mise en évidence par une couleur plus foncée, et la mise en évidence de la prochaine instruction à exécuter. Ce schème d'usage une fois construit est incorporé dans des schèmes d'actions instrumentés, par exemple le schème de traitement pas à pas (10.5), qui, chez les débutants, participe à construire les notions de séquence d'instructions et de programme (Louane, CM2, 2022t1p1v2).

10.4.2 La construction de la représentation des règles régissant le micromonde

Une simulation visuelle de l'exécution du programme de l'utilisateur est rendue sur la grille. En ce sens, du point de vue du sujet, la grille et les règles qui la régissent peuvent être considérées comme un élément du système informatique. Ces règles sont attachées au contexte du micromonde, elles sont de deux types, et peuvent générer quelques difficultés lors de leur introduction :

- *Le statut des éléments présents sur la grille (objet à ramasser, zone de dépôt, obstacle)*. Ce statut n'est parfois pas bien identifié par les sujets. Par exemple, une case cible peut être considérée comme un objet à ramasser, telle la fusée dans le contexte *Dans l'espace* (Lou, 6ème, 2022t4p3v1).

- *Les règles que le robot doit respecter dans la réalisation de sa mission*. Bien que précisées dans l'énoncé, ces règles ne sont pas forcément respectées d'emblée. Par exemple, lorsqu'une deuxième bille est introduite dans le contexte *Ranger les billes*, la règle qui stipule que le robot ne peut porter qu'une bille à la fois est souvent ignorée en première intention (Alix, CM2, 2022t2p1v3).

Pour résumer, la tâche du sujet peut être décomposée en deux sous-tâches, relativement indépendantes lors de la constitution de l'EIAH comme instrument : concevoir un programme, puis exécuter le programme conçu. D'une part, nous avons des cas où le sujet clique sur le bouton *Play* sans avoir conçu de programme (Louane, CM2, 2022t1p1v1). À l'inverse, certains sujets s'arrêtent après avoir accroché des blocs dans l'éditeur (Timéo, 5ème, 2022t1p1v1).

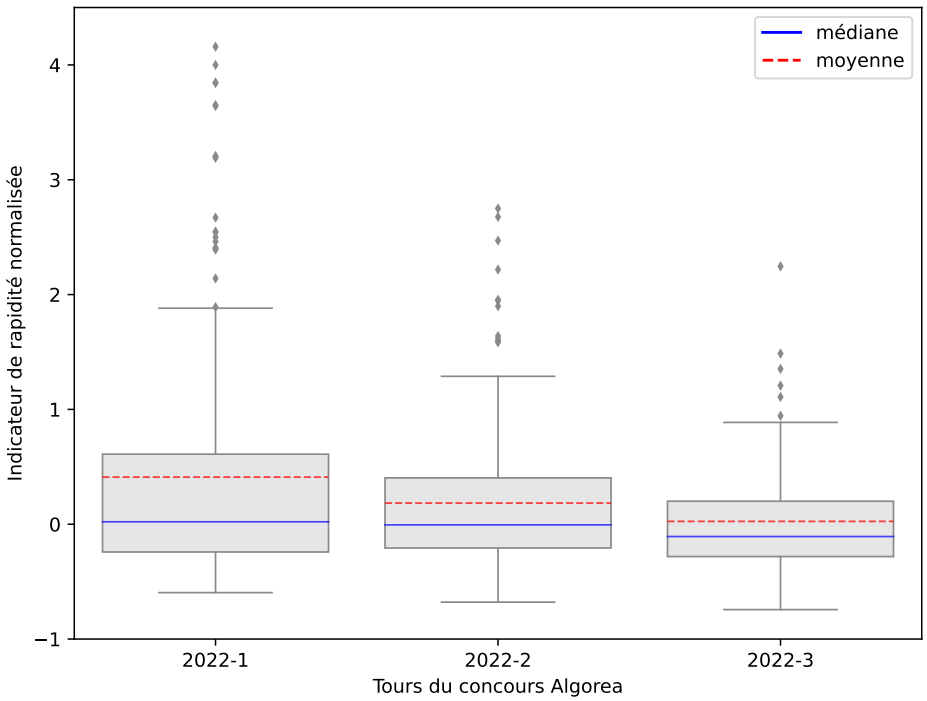


FIGURE 10.10 – Répartition de l'indicateur de rapidité normalisée pour le premier puzzle des parcours du concours Algoréa 2022

Nous ne savons pas quantifier de manière précise les difficultés associées à la prise en main des fonctionnalités de base de l'EIAH dans les traces d'interaction

à l'échelle des classes. D'une part, nous ignorons quels sujets sont confrontés pour la première fois à l'environnement de programmation. D'autre part, nous ne connaissons pas le degré d'étayage de l'enseignant. Nous montrons seulement que la moyenne de l'indicateur de rapidité normalisée décroît au fur et à mesure des trois tours de 2022 (Figure 10.10). La médiane devient négative pour le deuxième tour. C'est cohérent car une fois la prise en main réalisée, le premier puzzle ne comporte plus aucune difficulté. Il n'est donc pas étonnant que l'indicateur de rapidité normalisée soit négatif pour environ la moitié des élèves. Si nous regardons les durées passées sur ces puzzles pour avoir une représentation plus intuitive, la moyenne se situe à 37, 32 et 20 secondes, respectivement pour les premier, deuxième et troisième tours. Ces données montrent que l'environnement de programmation est constitué relativement rapidement comme instrument par la plupart des élèves.

10.5 Construction de premières notions de programmation de manière concomitante à la prise en main de l'EIAH

La confrontation aux premiers puzzles du parcours permettent la construction de la structure hiérarchique de schèmes qui amène à constituer l'EIAH comme instrument pour faire réaliser les missions demandées au robot virtuel. Nous montrons dans cette section que de manière concomitante et indissociable se construisent les premières conceptualisations-en-acte de notions de base en programmation.

Notre travail n'est pas centré sur l'acquisition de ces premières notions. Néanmoins, nous avons besoin que celles-ci soient acquises pour que les sujets puissent aborder le concept de répétition et l'identification de motifs, cœur de notre recherche. En conséquence nous les abordons succinctement.

- Le *passage du faire au faire faire* est porté par la constitution du langage de programmation comme instrument.
- La *décomposition en actions élémentaires* de la mission du robot est contrainte par l'utilisation des blocs d'action.
- La disposition spatiale des blocs dans l'éditeur, verticale de haut en bas, porte l'ordre d'exécution des actions du robot, et donc l'*aspect séquentiel du programme*.
- Le fait que le robot soit remis en position initiale dès qu'un bloc est manipulé dans l'éditeur, et donc que le programme soit exécuté dans sa

globalité lors de chaque lancement d'exécution, contraint le sujet à prendre en compte l'*exécution différée du programme*.

Schème de traitement pas à pas

- **but** : traiter une par une les actions à faire exécuter par la machine
- **règles de conduite de l'action** :
  - **prise d'information** :
    - Si toutes les actions n'ont pas encore été exécutées, alors repérer à partir de l'état courant la prochaine action à faire exécuter par la machine.
    - Repérer, parmi les blocs disponibles, celui à utiliser pour coder cette action.
  - **action proprement dite** : si c'est la première instruction, commencer le programme en suivant la syntaxe du langage, sinon placer cette instruction à la suite de celles déjà placées, pour concevoir le programme pas à pas.
  - **contrôle** : lancer une exécution pour vérifier que la machine adopte bien le comportement attendu.
- **invariants opératoires**
  - **concepts-en-acte** : correspondance terme à terme, mémoire de la machine
  - **théorèmes-en-acte** :
    - *Les actions de la machine sont déterminées par le programme* (relation de cause à effet). Ce n'est pas magique.
    - *Pour un même programme, la machine effectue les mêmes actions* (reproductibilité) : certains sujets éprouvent ce théorème-en-acte en lançant plusieurs fois de suite l'exécution du même programme.
    - *On peut lancer autant d'exécutions que souhaité, le programme n'est pas effacé après exécution* (permanence du programme en mémoire).
    - *L'accrochage d'un bloc dans le programme correspond à une action du robot virtuel* (correspondance terme à terme entre action et bloc)
    - *Le robot virtuel effectue les actions dans l'ordre du programme* (aspect séquentiel)

Par ailleurs, nous identifions un schème, que nous appelons schème de *traitement pas à pas* (encadré page précédente), dont la construction est concomitante à la conceptualisation-en-acte de ces premières notions de programmation et à la constitution de l'EIAH comme instrument. Ce schème de *traitement pas à pas* consiste à lancer une exécution après chaque ajout de bloc au programme. En plus d'illustrer la manière dont les premières notions de programmation se construisent en même temps que la prise en main de l'EIAH, le schème de *traitement pas à pas* constitue par la suite un schème *refuge* lorsque le sujet bute sur une difficulté.

Le schème de traitement pas à pas s'appuie sur le concept de *correspondance terme à terme* (bijection si nous utilisons le mot mathématique), qui consiste à associer les éléments de deux collections de manière à ce que chaque élément soit associé à un et un seul élément de l'autre collection. Le théorème-en-acte associé à la correspondance terme à terme peut être formulé dans ce contexte de programmation par *L'accrochage d'un bloc dans le programme correspond à une action du robot*. Cette correspondance entre bloc d'instruction et action de la machine mobilise la construction de la représentation de l'exécution pour chaque élément du langage de programmation (schème d'usage du langage de programmation). Dans l'encadré, le schème de traitement pas à pas est renseigné selon la définition analytique de Vergnaud (1991).

Nous donnons deux exemples représentatifs de la mise en oeuvre du schème de traitement en pas à pas lors de la découverte de l'interface, l'un où le sujet utilise le bouton play pour lancer les exécutions (Alix, CM1, 2021t1p1v3), et l'autre où le sujet utilise le mode pas à pas (Paola, CE1, 2021t3p2v1).

Figure 10.11 – Exemple de traces attachées au schème de traitement pas à pas

Le schème de traitement pas à pas est également repérable à partir des traces. Les enregistrements de programmes exécutés se suivent à intervalles de temps assez réguliers, de l'ordre de 15 à 30 secondes, avec chaque programme exécuté

qui est une solution partielle (fond bleu ou gris selon qu'elle amène vers une solution optimale ou pas) contenant une instruction de plus que le précédent. À titre d'exemple, la figure 10.11 montre la succession de programmes soumis par Paola (correspondant à la référence d'extrait vidéo).

Nous retrouvons ce schème de traitement pas à pas dans la littérature concernant l'initiation à la programmation avec des robots pédagogiques : Komis et Misirli, qui ont mené des expérimentations avec des élèves de 4 à 6 ans, parlent de *stratégie pas à pas* (Komis & Misirli, 2013). Cette stratégie est tout à fait comparable au schème de *traitement pas à pas* que nous avons identifié. Elle consiste à entrer chaque commande dans la mémoire du robot tangible et à l'exécuter aussitôt. Cependant, la mémoire du robot tangible est vidée après chaque exécution, car dans ce contexte le robot n'est pas remis en position initiale après l'exécution comme dans notre contexte de robot virtuel. Cette stratégie est aussi relevée par Spach, qui identifie un *programme mono-pas* lors d'une expérimentation avec des élèves de CE1 (Spach, 2017, p.198).

Dans notre contexte, la construction puis la mobilisation du théorème-en-acte *On peut lancer autant d'exécutions que souhaité, le programme n'est pas effacé après exécution*, attaché au concept-en-acte de mémoire de la machine, sont facilitées par la visibilité permanente du programme. Dans le cas contraire, comme par exemple avec le robot *Bee Bot*, la prise de conscience de la permanence du programme en mémoire est loin d'être évidente. Pour un exemple, voir Spach (2017, p.207).

10.6 Synthèse

L'étude instrumentale de l'environnement de programmation a permis de préciser l'activité instrumentée du sujet dans l'EIAH et ses enjeux pour notre recherche, tant du point de vue de la conceptualisation-en-acte que du point de vue méthodologique.

Nous avons établi la structure hiérarchique de schèmes attachée à la constitution de l'EIAH comme instrument. Au sein de cette structure qui comporte quatre niveaux, les schèmes d'action instrumentée d'un niveau sont incorporés comme schèmes d'usage au niveau supérieur. Lors de la première confrontation avec l'environnement de programmation, les schèmes d'usage de l'EIAH sont construits s'ils ne sont pas encore disponibles : déplacer et accrocher des blocs, lancer l'exécution d'un programme. Pour les niveaux hiérarchiques supérieurs, schèmes d'usage du langage de programmation et schèmes d'action instrumentée des boutons de contrôle de l'exécution sont intimement liés à la construction des premières conceptualisations-en-acte des notions de base en programmation : notions d'instruction et de programme, caractère séquentiel de celui-ci. Nous

avons au passage documenté le schème de *traitement pas-à-pas*, qui accompagne la construction des premières notions, avant de s'estomper, laissant place à une conception plus globale du programme.

Pour aborder la notion de boucle et l'identification de motifs, il est nécessaire d'une part que ces premières notions soient maîtrisées. C'est pourquoi, mis à part les puzzles p1v3 qui nous servent d'évaluation de la maîtrise de la boucle simple en entrée de parcours, les puzzles retenus pour notre étude se situent au milieu du parcours. D'autre part, dans le cadre de notre étude centrée sur la conceptualisation-en-acte, il est important de repérer les aspects de l'activité du sujet et les difficultés qui relèvent plus de genèses instrumentales que de genèses conceptuelles : difficultés propres à la construction de la signification de certains blocs d'action, à des règles de fonctionnement de la grille, surtout quand ces nouveaux blocs et règles sont introduits tardivement dans les parcours, de manière concomitante à des éléments algorithmiques centraux dans notre étude. Avoir repéré ces difficultés qui constituent des variables parasites pour notre étude nous permet d'en tenir compte et de les écarter lors de nos analyses relatives aux genèses conceptuelles.

Du point de vue méthodologique, nous avons montré l'apport de l'analyse de l'activité du sujet muni de la souris constituée en instrument. L'enregistrement de la position du curseur de souris et la génération de visualisation des mouvements de souris sur l'interface nous semble un apport du point de vue de la recherche en EIAH. En effet, cette source de données est encore peu mobilisée dans les travaux sur les processus d'apprentissage, à cause probablement de l'aspect chronophage des descriptions minutieuses à réaliser manuellement. Cet aspect méthodologique reste implicite dans les travaux sur l'apprentissage de la programmation avec un langage par blocs, même dans ceux qui mobilisent des enregistrements vidéo d'écran (Çakiroğlu & Mumcu, 2020 ; Grover et al., 2017 ; Xu et al., 2023). Par rapport à cette problématique, notre approche permet d'accéder à une granularité de collecte très fine, à même de rendre compte du mouvement de manière précise, et ce sur un volume important de données en automatisant la collecte et le traitement de ces données. Cette approche offre des perspectives d'une part pour alléger ce travail chronophage d'analyse manuelle, et d'autre part pour effectuer des analyses quantitatives à partir des traces enregistrées. Nous envisageons aussi d'appliquer des algorithmes d'intelligence artificielle pour repérer automatiquement certains schèmes comme celui du dénombrement, ou pour établir des catégories via un *clustering*.

Chapitre 11

Champ conceptuel et classes de situations attachées au concept de motif

Sommaire du présent chapitre

Après le détour nécessaire par la composante instrumentale de l'activité de programmation dans l'environnement Algoréa, nous recentrons notre propos sur la problématique au cœur de notre recherche. L'objectif de ce chapitre est d'approfondir ce que sous-tend le concept de motif dans le contexte de l'initiation à la programmation par blocs à travers la résolution de puzzles. À cette fin, nous mobilisons la théorie des champs conceptuels de Vergnaud introduite dans le chapitre 4, les concepts piliers de la science informatique abordés dans le chapitre 1 et nos investigations relatives au concept de motif présentées dans le chapitre 3. Nous commençons par mener un travail théorique d'approfondissement sur le concept de motif, qui nous amène à délimiter un champ conceptuel des bases de l'algorithmique, pour lequel nous montrons que le concept de motif est un concept structurant. Nous contextualisons ce champ conceptuel à la programmation en langage Scratch dans une première étape, puis nous resserrons nos investigations à la programmation d'un robot sur une grille dans l'environnement Algoréa présenté dans le chapitre 7. Pour ce contexte, nous menons une analyse a priori des motifs en jeu, puis nous vérifions nos hypothèses à partir des données à l'échelle nationale (8.5.1.1).

Finalement, en nous appuyant sur les concepts de situation, de classe de situations et de variable de situation au sens de Vergnaud, nous traitons la question de recherche suivante :

> Quelles classes de situations attachées au concept de motif peut-on définir ?

11.1 Approfondissement du concept de motif dans le champ de l'initiation à la programmation informatique

L'objet de cette section est de mener une analyse épistémologique du concept de motif dans le champ de l'initiation à la programmation informatique. Nous décomposons d'abord ce que recouvre le concept de motif dans ce contexte, en en identifiant plusieurs instances distinctes. Une fois ces éléments théoriques

posés, nous définissons le périmètre d'un champ conceptuel des bases de l'algorithmique dans lequel nous positionnons le concept de motif. Pour ce champ conceptuel contextualisé au langage Scratch, nous déterminons des classes de problèmes de manière systématique, comme le préconise Vergnaud (1991).

11.1.1 Motif visuel et motif algorithmique

Dans le chapitre 3, nous avons défini un *motif* comme une entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles. Nous questionnons les liens entre le concept de motif appréhendé de cette manière et les concepts piliers de la science informatique (1.1.2) au sens de Dowek (2011) et Berry (2017) : information/données, algorithme, langage et machine. Deux de ces concepts sont plus étroitement en lien avec celui de motif, ceux d'algorithme et de données.

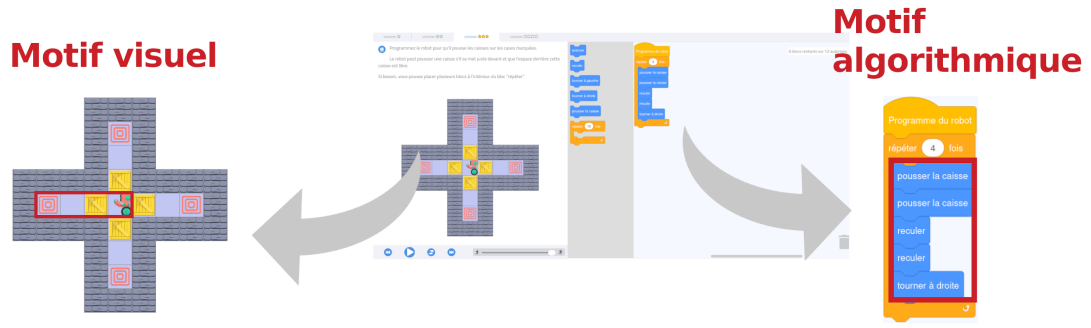


Figure 11.1 – Motif visuel et motif algorithmique sur l'interface d'un puzzle de programmation

Lorsque le traitement à faire exécuter par une machine, c'est-à-dire l'algorithme, comporte des régularités, nous pouvons identifier une occurrence de motif au sens où nous l'avons défini. Ce motif est une partie du traitement qui est repérable dans l'ensemble des traitements à faire exécuter par la machine car répété plusieurs fois, consécutivement ou non, à l'identique ou avec des variations prédictibles. Nous qualifions donc ce motif d'*algorithmique*. Le motif algorithmique, comme le concept d'algorithme, est de l'ordre du traitement logique. C'est la répétition d'un ensemble d'instructions dans le même ordre chronologique qui détermine le motif algorithmique, que ces actions soient exprimées dans un langage de programmation ou non. Lorsque l'algorithme est exprimé dans un langage de programmation, on retrouve une représentation du motif algorithmique exprimé avec des éléments de ce langage. En général, ce sont des représentations synthétiques, sous forme de boucle ou de fonction.

La représentation du motif algorithmique dans le langage de programmation dépend aussi des caractéristiques de la machine (système d'orientation d'un robot par exemple). Par ailleurs, nous associons le motif algorithmique avec l'aspect de *pattern recognition* qui concerne la conception des algorithmes (3.2.2).

Le motif algorithmique peut être intrinsèque au processus de traitement logique. Il peut aussi être lié à des régularités dans les données, ce à quoi réfère le deuxième aspect de *pattern recognition*. Dans ce cas, il est possible d'identifier un motif dans les données et toutes les instances de ce motif seront traitées de la même façon, déterminant un motif au sein de l'algorithme (motif algorithmique). Le concept de motif est alors associé à celui d'information, entendu comme l'ensemble des données à traiter automatiquement, ce qui nous amène à distinguer une deuxième occurrence de motif, qui dépend du format des données. Ce motif est présent indépendamment du traitement algorithmique effectué sur les données, c'est pourquoi nous le qualifions d'*inhérent*. Cependant, nous utiliserons plutôt un qualificatif relatif au format des données.

Le fait d'identifier des motifs dans les données à traiter est un point central directement en lien avec la définition de la science informatique. En effet, la présence de régularité dans les données est ce qui permet dans une certaine mesure d'automatiser le traitement de l'information en reproduisant un même traitement sur toutes les instances d'un même motif.

Dans les cas particuliers des suites de motifs identifiés dans la littérature (3.3), et des situations de programmation d'un robot virtuel sur une grille (7.1), le motif est dans une modalité visuelle, nous parlerons de *motif visuel* pour le motif inhérent. Mais il pourrait tout à fait être dans une autre modalité, sonore par exemple. La figure 11.1 montre une contextualisation des motifs visuel et algorithmique à un puzzle de programmation dans l'environnement Algoréa.

11.1.2 Modélisation d'un champ conceptuel des bases de l'algorithmique incluant le concept de motif

L'objet de cette section est de positionner le concept de motif au sein d'un champ conceptuel des bases de l'algorithmique. Nous concevons une modélisation de ce champ conceptuel, modélisation qui mène à décrire les filiations entre les différents concepts du champ (Figure 11.2). Nous détaillons en particulier les liens de ces concepts avec celui de motif. Pour cette modélisation, nous optons pour un périmètre qui comprend les concepts de la programmation impérative introduits dans le chapitre 1, concepts présents dans les programmes scolaires du cycle 4 (1.2.1) : séquence d'instructions, répétition, structure conditionnelle, variable et fonction.

Le concept de motif est impliqué lorsque plusieurs traitements répétés à

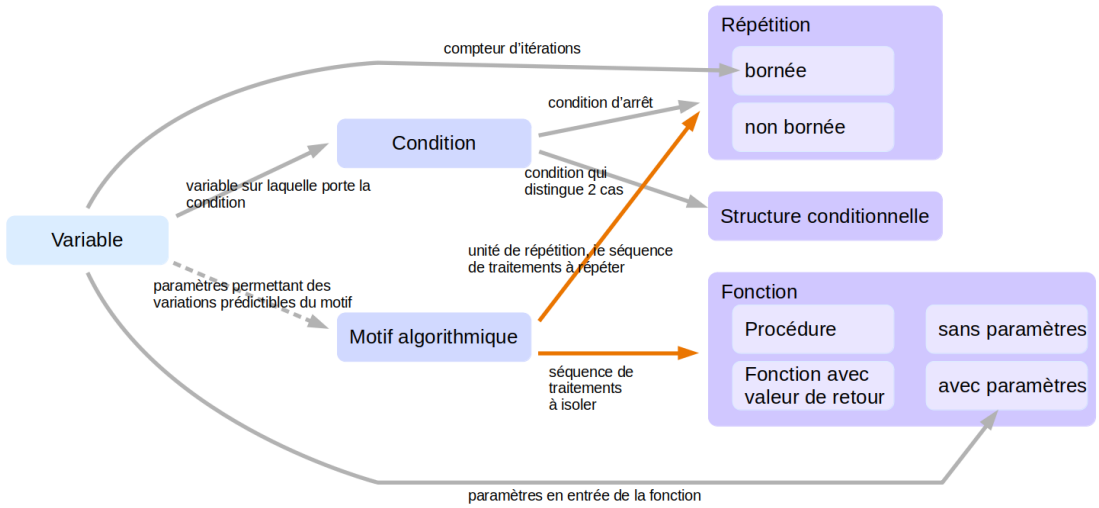


FIGURE 11.2 – Modélisation d'un champ conceptuel des bases de l'algorithmique

l'identique ou de manière prédictible sont à faire exécuter par la machine. Un tel traitement, considéré comme une entité, constitue un motif algorithmique, à traduire dans un langage de programmation. Comme nous ne nous intéressons pas aux données dans cette section mais seulement à l'enchaînement des traitements à faire exécuter par la machine, le motif algorithmique sera le seul pris en compte. En conséquence, la portée de ce champ conceptuel va au-delà de la seule situation de la programmation d'un robot virtuel sur une grille, mais concerne l'ensemble des situations de programmation impérative impliquant les concepts de base ciblés précédemment. Aussi, pour alléger l'écriture, nous employons parfois seulement le terme *motif* dans cette section, le qualificatif d'*algorithmique* étant alors sous-entendu. Le positionnement des concepts de base de l'algorithmique permet d'obtenir une modélisation où chaque flèche indique une filiation. Lorsque deux concepts sont reliés par une flèche, cela signifie que le concept qui est placé à l'origine de la flèche est impliqué dans la construction de celui que pointe la flèche. La spécification de cette filiation est précisée sur la flèche.

Sur cette modélisation, le concept de variable est la brique de base du champ conceptuel, c'est à dire celle à l'origine de toutes les flèches. C'est cohérent avec l'essence du traitement informatique qui consiste en une évolution de la valeur de variables au cours du processus (LAGRANGE & ROGALSKI, 2017). Le concept de motif algorithmique est, au même niveau que celui de condition, un concept structurant du champ. En effet, dans le système hiérarchisé de concepts obtenu, ces deux concepts sont des briques intermédiaires, nécessaires à la construction des concepts plus complexes de répétition, structure conditionnelle et fonction.

Plus particulièrement pour le concept de motif, il intervient directement dans la construction des concepts de répétition et de fonction (flèches orange sur la figure 11.2). Dans les paragraphes suivants, nous analysons de manière plus précise les filiations du concept de motif avec les autres concepts du champ.

Concept de motif et concept de répétition

Pour les deux types de répétition, bornée et non bornée, le concept de motif est en jeu de la même façon. Il correspond au corps de la boucle, c'est-à-dire au bloc de traitements à faire répéter par la machine, « l'expression de l'invariant dans les actions à répéter » (Rogalski & Samurçay, 1986). La différence entre boucle bornée et non bornée porte sur le « statut fonctionnel du contrôle d'arrêt », qui dans le premier cas porte sur le nombre d'itérations qui est connu à l'avance, et dans le second cas sur « une valeur calculée au cours de l'exécution du traitement répétitif » (Rogalski & Samurçay, 1986). Ce contrôle de l'arrêt se traduit par la combinaison avec d'autres concepts du champ : variable et condition.

Concept de motif et concept de fonction

Une fonction consiste à isoler une partie du traitement et à lui donner un nom pour pouvoir l'utiliser plusieurs fois (que cette fonction renvoie une valeur ou pas). Le motif correspond alors à cette partie du traitement qui est isolée et étiquetée. Dans un contexte de programmation, si on considère l'ensemble dont fait partie le motif comme allant au-delà du programme en cours de conception, on retrouve le processus de généralisation, de réemploi de morceaux de programmes précédemment construits, stockés dans des bibliothèques de fonctions.

Concept de motif et concept de structure conditionnelle

Le concept de motif n'intervient pas directement dans la construction du concept de structure conditionnelle. Cependant, le cas d'une structure conditionnelle imbriquée dans une boucle retient notre attention. En effet, pour cette combinaison de structures de contrôle, nous avons deux motifs dont l'un ou l'autre est présent suivant la satisfaction ou non d'une condition.

Concept de motif et concept de variable

Si les motifs sont répétés à l'identique, le concept de variable n'est pas mobilisé. Il l'est lorsque le motif est répété avec des variations prédictibles, une ou plusieurs variables constituant le ou les paramètres du motif.

Notre modélisation montre que les concepts de base de l'algorithmique peuvent être hiérarchiquement structurés sous la forme d'un graphe orienté acyclique (DAG). L'analyse des filiations au sein de ce DAG montre que le concept de motif est un élément structurant de l'ensemble des concepts de base de l'algorithmique.

11.1.3 Contextualisation de ce champ conceptuel à la programmation en langage Scratch

Après avoir décrit un champ conceptuel des bases de l'algorithmique au niveau du traitement logique, sans faire intervenir un langage de programmation particulier, nous le contextualisons à la programmation en langage Scratch dans l'environnement Algoréa (7.1). Comme le langage Scratch est celui préconisé par l'institution scolaire jusque la fin du collège, cette contextualisation constitue aussi un choix de niveau d'analyse au sens où le préconise Vergnaud. La figure 11.3 reprend la modélisation du champ conceptuel des bases de l'algorithmique introduite dans la section précédente en tenant compte de cette contextualisation : les blocs permettant d'exprimer les différents concepts en langage Scratch sont représentés, les concepts et filiations qui ne sont pas effectifs en Scratch sont grisés.

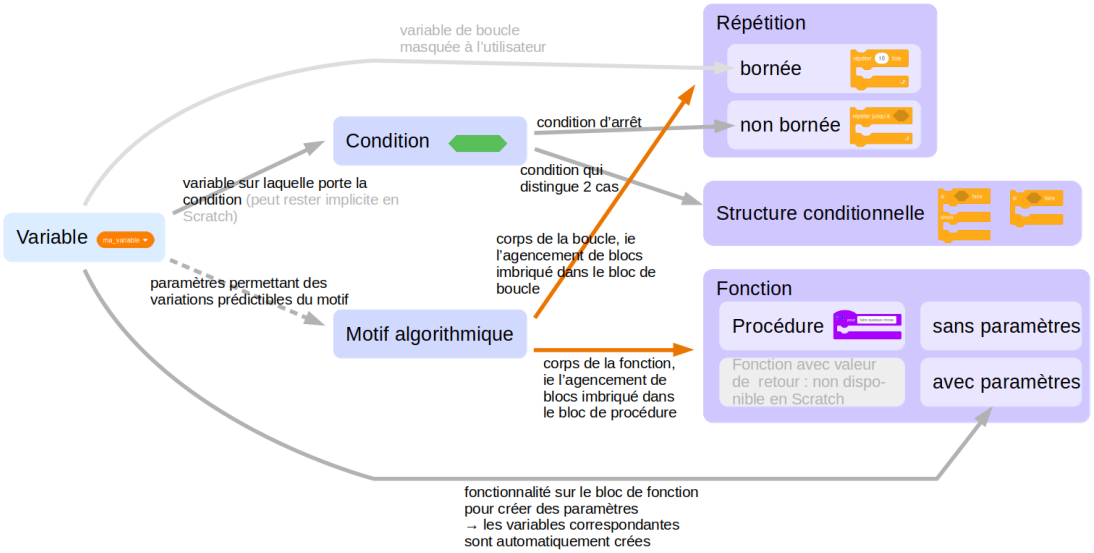


Figure 11.3 – Contextualisation du champ conceptuel des bases de l'algorithmique au langage Scratch

En langage Scratch, le motif algorithmique est représenté par un agencement de blocs. Dans l'environnement de programmation Algoréa, le robot virtuel constitue la machine aux yeux de l'utilisateur, bien qu'il n'en soit qu'une métaphore. Donc dans ce contexte, le motif algorithmique est représenté par l'agencement de blocs correspondant à une séquence d'actions à faire exécuter par le robot virtuel. Notamment pour les boucles, le motif algorithmique est représenté par l'agencement de blocs imbriqué dans le bloc *répéter 10 fois* ou le bloc *répéter jusqu'à*, c'est à dire par le corps de la boucle. Pour ce qui concerne les fonctions, seules les procédures, i.e. fonctions sans valeur de retour, sont disponibles en Scratch. Le motif algorithmique est alors représenté par l'agencement de blocs imbriqué dans le bloc de *procédure*, c'est-à-dire par le corps de la procédure. Dans les deux cas, boucles, et procédures, cet agencement de blocs est constitué de blocs d'actions, mais il peut comprendre d'autres types de blocs, amenant une variété de structures imbriquées les unes dans les autres.

Nous remarquons qu'en Scratch, le concept de variable reste implicite à plusieurs niveaux. Pour la répétition bornée, le bloc *répéter x fois* masque la variable de boucle à l'utilisateur. De même pour la structure conditionnelle, des capteurs permettent de générer des conditions, qui mobilisent la création et l'évaluation de variables booléennes ou numériques en arrière-plan, sans que le concept de proposition booléenne soit introduit de manière explicite. Nous interprétons cette prise en charge des variables en arrière-plan par le système comme une transposition didactique qui permet d'introduire les concepts de répétition et de structure conditionnelle avant celui de variable, concept identifié comme difficile lors de l'initiation à la programmation. En termes de transparence opérative, les concepteurs de Scratch ont considéré que l'utilisateur n'avait pas besoin de ces informations pour s'initier à la programmation. Néanmoins, la gestion explicite de variables est disponible en Scratch dans la mesure où l'utilisateur a la possibilité d'en créer lui-même. La création de variables permet notamment de paramétrer un motif pour rendre compte de variations prédictibles. Lorsque le paramétrage du motif est réalisé au sein d'une procédure, les variables sont automatiquement créées lors de la définition des paramètres de la procédure et mises à disposition dans le menu de blocs.

Le point clé de cette section pour notre focus sur le concept de motif est le fait qu'en langage Scratch, le motif algorithmique correspond à un agencement de blocs qui est une représentation de la séquence d'actions à faire répéter à la machine.

11.1.4 Détermination des classes de problèmes au sein de ce champ conceptuel

Notre démarche consiste alors à définir a priori des classes de situations correspondant à la nécessité de mobiliser des notions algorithmiques autres que celle de séquence d'instructions (traitement par défaut). C'est une étude analytique du contenu du champ conceptuel défini, en repérant, ce qui, dans les situations, constitue des ruptures. La modélisation est réalisée du point de vue de l'organisation du savoir, dans une visée de classification systématique que Vergnaud décrit comme indispensable : « Toute situation peut être ramenée à une combinaison de relations de base avec des données connues et des inconnues, lesquelles correspondent à autant de questions possibles. La classification de ces relations de base et des classes de problèmes qu'on peut générer à partir d'elles est un travail scientifique indispensable. » (Vergnaud, 1991). Dans cette section, une classe de situations définie a priori sera appelée *classe de problèmes* comme dans la citation de Vergnaud.

Pour ce travail, nous nous plaçons au niveau du traitement logique, et nous nous limitons aux concepts effectifs pour notre niveau d'analyse, c'est-à-dire que nous considérons le champ conceptuel des bases de l'algorithmique contextualisé au langage Scratch (Figure 11.3). Comme le concept de variable est implicite dans ce contexte, l'une des racines possibles de la hiérarchie de concepts est le concept de motif algorithmique. Nous envisageons donc le traitement algorithmique du motif comme fil conducteur pour catégoriser les situations de notre champ conceptuel. Nous aboutissons à l'arbre de décisions présenté sur la figure 11.4. Celui-ci comprend un ensemble de questions (sur fond jaune) à se poser lors de l'analyse d'une situation. La réponse à chaque question (OUI/NON) détermine la mobilisation ou non d'un concept algorithmique pour le traitement de cette situation, et éventuellement une nouvelle question (flèche verte).

Si on suit cet arbre de décision afin de classifier une situation par rapport aux concepts de base de l'algorithmique, la présence de régularités dans les actions à faire exécuter par la machine est la première question à se poser. Suivent d'autres questions relatives au motif (variations prédictibles), à la distribution des motifs au sein du traitement (consécutifs ou non), à la connaissance en amont du nombre de répétitions, qui déterminent le concept algorithmique avec lequel celui de motif va être combiné.

Dans cette classification, une classe de problèmes correspond à une combinaison de réponses aux questions de l'arbre de décision. Le cheminement dans cet arbre de décision montre l'importance du concept de motif algorithmique, dont l'identification est un préalable indispensable à la mobilisation de boucles ou de procédures.

Ce travail de modélisation apporte des éléments théoriques pour la compré-

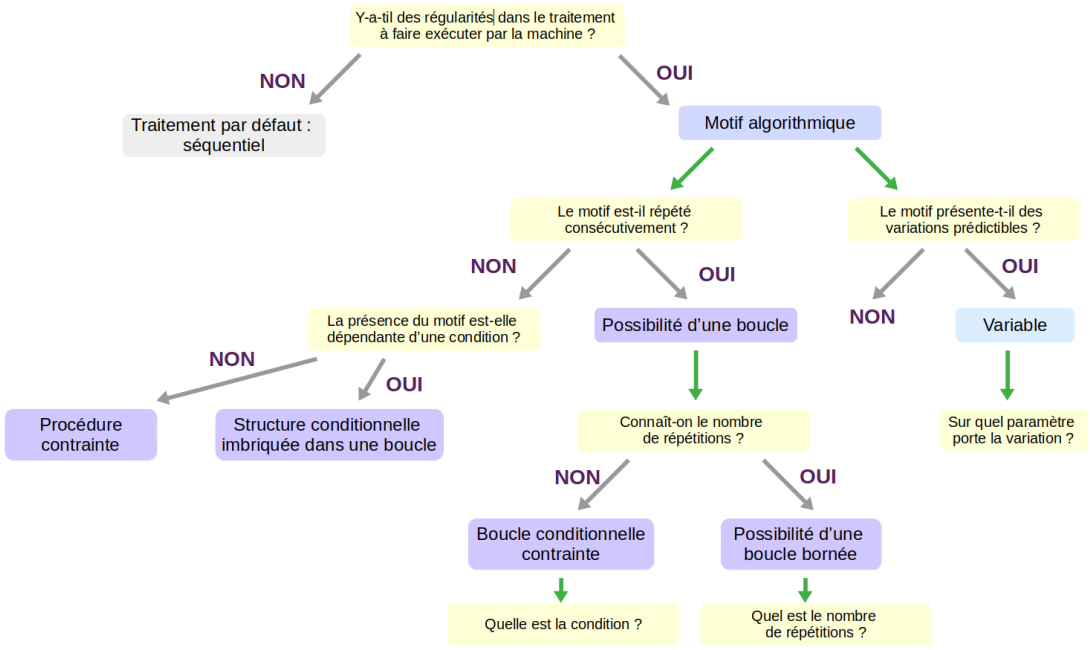


FIGURE 11.4 – Arbre de décision en vue du traitement d'une situation comportant des régularités

hension et la structuration du champ conceptuel des bases de l'algorithmique. Il peut aussi constituer une aide à destination des enseignants en charge de l'initiation à l'informatique.

11.2 Cas de la programmation d'un robot virtuel sur une grille comportant des régularités

Dans la suite, nous resserrons notre analyse au cas des puzzles de programmation d'un robot virtuel sur une grille. Dans le cas de ces puzzles, la définition de la grille et la position du robot virtuel sur cette grille constituent les données d'entrée sur lesquelles va opérer l'algorithme. La régularité dans les données prend alors la forme d'un *motif visuel* observable sur la grille. Celui-ci est constitué de cases adjacentes, contenant ou non un élément saillant visuellement (case marquée, ou contenant un objet). Le *motif algorithmique*, quant à lui, n'est observable sur la grille qu'à travers la simulation de l'exécution du programme par le robot virtuel. Il dépend du motif visuel et du système d'orientation du robot virtuel. En langage Scratch, la représentation d'un motif algorithmique est constituée d'un agencement de blocs accrochés les uns aux autres.

Pour la classe de problèmes relative à la boucle bornée, nous établissons le lien avec les tâches impliquant le concept de motif relevées dans les articles de didactique des mathématiques (section 3.3) dans une première section. Dans la section suivante, nous resserrons encore notre analyse, pour nous concentrer sur la programmation d'un robot virtuel sur une grille qui comporte des régularités. Nous menons, pour ce cas, des investigations sur le lien entre motif visuel et motif algorithmique.

11.2.1 Ensemble de situations impliquant le concept de motif dans le champ de l'initiation à l'informatique

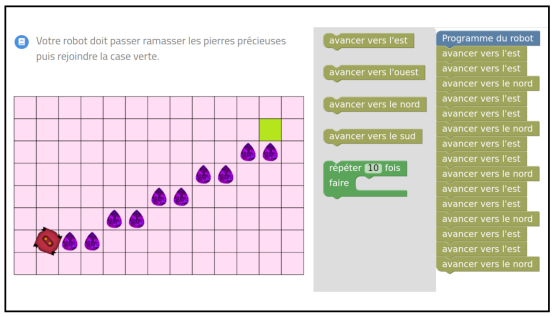
Dans cette section, notre objectif est de reprendre les tâches impliquant le concept de motif relevées dans les articles de didactique des mathématiques (3.3) et de questionner leur contextualisation à l'initiation à la programmation, en nous appuyant sur nos travaux antérieurs, et sur quelques travaux relevés dans la littérature. Nous situant désormais dans le cadre théorique de Vergnaud, les termes *situation* et *tâche* ont le même sens et nous employons un ou l'autre indifféremment.

Dans le champ de la science informatique, le concept de machine est convoqué en lien avec ceux d'algorithme, de langage et d'information, ce qui constitue une différence majeure par rapport aux mathématiques. La contextualisation des situations précédemment décrites consiste donc à analyser ce qui se passe lorsqu'on convoque le concept de machine, c'est-à-dire lorsqu'on passe du *faire* au *faire faire*. Nous retrouvons les situations élémentaires impliquant le concept de motif relevées en didactique des mathématiques dans des situations où un motif visuel est présent.

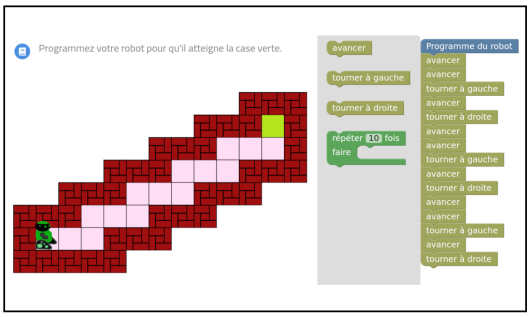
11.2.1.1 Transférer une suite de motifs d'une représentation vers une autre

Déterminer à partir des éléments présents sur la grille les actions que le robot virtuel doit exécuter et exprimer ces actions dans un langage de programmation est une tâche de transfert, de la représentation des actions du robot qui sont déduites des éléments de la grille (cases avec certaines caractéristiques) à la représentation de ces mêmes actions dans un langage de programmation. Comme la grille reste visible en permanence, cette tâche de transfert peut être réalisée par correspondance terme à terme, c'est à dire en considérant les actions du robot les unes après les autres de manière indépendante. Dans ce cas, l'activité de transfert d'une représentation vers une autre est indépendante de la présence de motifs visuels sur la grille.

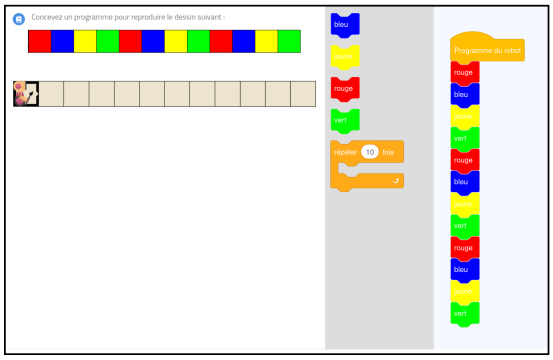
Sur la figure 11.5, ce qui varie entre les différents exemples est la proximité entre les deux représentations, celle des actions du robot à inférer à partir des



Exemple 1 : situation d'un parcours Chticode en orientation absolue



Exemple 2 : situation d'un parcours Chticode en orientation relative
(le programme traite le parcours des trois premières marches)



Exemple 3 : situation dans l'environnement Motif Art

Exemple 4 : situation dans l'environnement de programmation Emil
(Kalas &al., 2018)

FIGURE 11.5 – Exemples de robots virtuels sur une grille et représentations des actions à leur faire faire

cases de la grille, et celle des actions du robot exprimées dans un langage de programmation. Nous présentons ces exemples en allant des représentations les plus proches aux représentations les plus éloignées.

Dans l'environnement Motif Art (LÉONARD et al., 2021) destiné à de jeunes élèves à partir de 5 ans, les deux représentations, case de couleur et bloc de couleur, sont très similaires visuellement. Nous sommes dans le cas d'une suite de motifs visuels comme dans les situations de didactique des mathématiques (section 3.3). Lorsque le programme conçu pour reproduire la frise est une séquence d'instructions (pas d'utilisation du bloc *répéter*), cela revient à copier la suite de motifs en utilisant, pour chaque élément de la frise, un bloc de langage de même couleur (figure 11.5 exemple 3). À noter que dans cet environnement de programmation accessible à des non lecteurs, le transfert nécessite tout de même une rotation mentale qui est potentiellement source de difficulté, la frise étant présentée horizontalement alors qu'il faut la transcrire sous forme d'une séquence verticale de blocs. Cette situation est malgré tout très proche de la situation « Copier une suite de motifs » répertoriée dans la section 3.3.

Dans la plupart des cas, il n'y a pas similarité visuelle entre les deux représentations, qui ne sont pas de même nature. D'un côté, nous avons affaire à des éléments iconiques sur la grille. De l'autre côté, même dans le cas d'un langage de programmation par blocs, l'action est représentée sous forme textuelle sur le bloc (Exemples 1 et 2 de la figure 11.5 en langage Blockly).

De plus, il faut distinguer deux cas pour la correspondance entre les déplacements du robot et les cases de la grille. Ce qui distingue ces deux cas est le système d'orientation du robot. En orientation absolue (Figure 11.5, exemple 1), la correspondante est stricte, bijective. Chaque déplacement correspond au passage d'une case à une autre et le trait de séparation entre deux cases peut représenter visuellement le déplacement. En orientation relative (Figure 11.5, exemple 2), il n'y a pas de bijection entre élément visuel et action de déplacement du robot puisque le robot ne change pas de case lors d'un pivotement. La représentation sur la grille des déplacements du robot est donc seulement partielle.

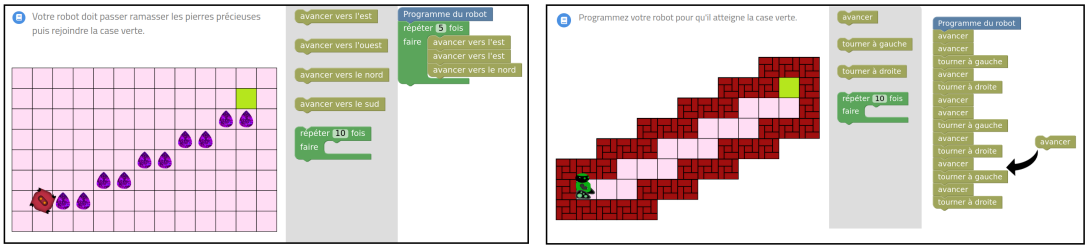
Un dernier exemple est relevé dans l'article de Kalas et al. qui présente un environnement de programmation nommé *Emil* (Kalas et al., 2018). Dans la situation de l'exemple 4 (figure 11.5), le langage de programmation est constitué de symboles iconiques (flèches pour contrôler les actions du robot, quelques autres symboles). Ce jeu d'instructions avec des flèches constitue une représentation intermédiaire, qui ne nécessite pas de savoir lire : de même nature (visuelle) mais avec deux représentations différentes (cases vs flèches).

11.2.1.2 Continuer une suite de motifs

Dans le contexte de l'initiation à la pensée informatique, continuer une suite de motifs revient à se mettre à la place de la machine, à anticiper l'exécution que l'on prévoit de lui faire faire. D'une part, nous retrouvons cette situation lors d'activités débranchées. En amont d'une séquence de robotique pédagogique, Saxena et al. proposent une tâche qui consiste à continuer une suite de motifs avec des briques de construction posées les unes à côté des autres (Saxena et al., 2020). Le même type de tâche mais avec des gommettes (Figure 11.6) a aussi été utilisé lors d'un essai de pré-test en amont de l'une de nos études de cas, antérieure à la présente recherche doctorale.

D'autre part, nous retrouvons aussi cette situation lors d'activités de programmation, lorsqu'une boucle est mobilisée et que le sujet simule mentalement l'exécution de son programme. Un seul motif algorithmique est représenté dans l'éditeur, placé à l'intérieur du bloc de boucle (Figure 11.7 exemple 5). Pour se représenter l'exécution du programme, il est nécessaire de dérouler mentalement la suite des motifs algorithmiques, et de la mettre en regard de la suite des motifs visuels de la grille.

FIGURE 11.6 – Situation sur papier où il s'agit de continuer une suite de motifs



Exemple 5 : situation d'un parcours chticode - motif algorithmique isolé (dans une boucle)

Exemple 6 - situation d'un parcours chticode - phase de débogage : retrouver un bloc manquant dans une suite de motifs algorithmiques

FIGURE 11.7 – Exemple de situation pour *identifier un motif* et pour *retrouver un élément manquant dans une suite de motifs*

11.2.1.3 Retrouver les éléments manquants dans une suite de motifs

Lors des situations de programmation d'une suite de motifs, nous pouvons identifier l'activité de retrouver des éléments manquants dans une suite de motifs lors de la phase de débogage. Cela correspond au cas où le sujet a oublié un bloc ou plusieurs blocs dans son programme (Figure 11.7 exemple 6). Il s'agit alors pour lui d'identifier quel bloc manque mais aussi à quel endroit de la séquence placer ce bloc, ce qui ajoute une difficulté. Cette activité est cependant sensiblement différente, et plus complexe, de celle décrite par les auteurs de didactique des mathématiques car l'élément manquant est identifié à la fois par rapport à d'autres représentations du motif algorithmique, identiques visuellement, mais aussi par rapport à l'adéquation de cette représentation du motif algorithmique dans le programme avec le motif visuel sur la grille. Le changement de représentation, décrit précédemment, interfère avec l'identification de l'élément manquant.

11.2.1.4 Identifier un motif

En algorithmique, identifier et isoler un motif qui se répète dans les données à traiter permet de synthétiser l'écriture de la séquence à faire faire par la machine sous forme de boucle (si les motifs sont consécutifs) ou sous forme de procédure (si d'autres actions sont intercalées). L'identification de motif est

donc centrale dans le traitement automatisé des données. Dans le scénario pédagogique *MOTIF..MOTIF..* mis en œuvre lors de l'une de nos expérimentations, la tâche d'identification de motif est introduite par une activité débranchée en amont de l'initiation à la programmation dans l'environnement *Motif Art* (Léonard et al., 2021). Il est demandé aux élèves de casser une tour constituée d'une suite de motifs dans le but d'obtenir le plus de morceaux identiques possibles (figure 11.8). En d'autres termes il s'agit de décomposer la tour en isolant les motifs les uns des autres. Cette manipulation de matériel tangible vise à faire prendre conscience de la structure d'une suite de motifs visuels, afin d'être plus à l'aise pour faire reproduire le même type de frises par le robot virtuel, en synthétisant l'écriture de la suite de motifs avec le bloc *répéter n fois* du langage Scratch.

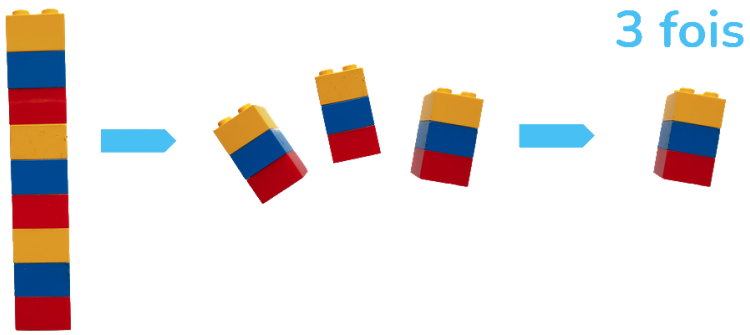


Figure 11.8 – Situation débranchée d'identification de motif

Cette situation d'identification de motif, centrale lors de l'initiation au concept de répétition, est l'objet de toute notre attention dans la suite de cette recherche. Nous y reviendrons plus en détails dans les parties suivantes, à travers l'analyse des situations de programmation dans l'environnement de programmation Algoréa.

11.2.1.5 Synthèse

Dans cette section, nous avons donc défini un ensemble de situations qui donnent du sens au concept de motif dans le champ de l'initiation à l'informatique. Nous retrouvons toutes les situations élémentaires impliquant le concept de motif relevées en didactique des mathématiques (3.3). Ces situations élémentaires interviennent à différentes étapes de la programmation d'un robot virtuel sur une grille comportant un motif visuel : analyse de la grille (qui dans ce cas constitue les données en entrée), construction du programme dans un langage

par blocs, représentation mentale de l'exécution du programme, débogage de ce programme.

Parmi ces situations élémentaires, l'identification de motif est centrale pour faire traiter les régularités par la machine, notamment dans le cas de la programmation d'un robot virtuel sur une grille. Cette situation de programmation est cependant plus complexe que les situations élémentaires décrites dans le chapitre 3 car deux types de motifs, visuel et algorithmique, sont à prendre en compte simultanément.

11.2.2 Positionnement de l'identification de motif dans le traitement de la situation

Dans le cas de la programmation d'un robot virtuel sur une grille qui comporte des motifs visuels (pas forcément consécutifs), nous sommes en présence de deux motifs qui sont en correspondance : ce motif visuel sur la grille, motif par rapport auquel vont être effectuées les actions du robot, et le motif algorithmique dont la représentation en langage Scratch est un agencement de blocs. Contrairement au motif visuel, le motif algorithmique n'est observable sur la grille que lors de l'exécution effective des actions par le robot. Comme nous sommes en présence de deux instances de motif, nous nous demandons à partir de laquelle la régularité est perçue par le sujet. Autrement dit, nous nous questionnons sur le moment du processus cognitif où se déroule l'identification du motif, identification nécessaire afin de traiter algorithmiquement cette régularité, c'est à dire d'utiliser des structures telles que boucles ou procédures pour synthétiser l'écriture du programme. Nous identifions deux possibilités a priori (Figure 11.9).

Un première possibilité correspond au cas où le sujet perçoit la régularité sur la grille et se trouve en mesure de délimiter précisément le motif visuel qui est répété. Il déduit de cette représentation visuelle du motif la séquence des actions à faire exécuter par le robot virtuel, en l'exprimant en langage Scratch. Il traite ensuite algorithmiquement l'agencement de blocs obtenu en le plaçant dans la structure qui convient à la situation (boucle *répéter n fois*, boucle *jusqu'à* ou procédure). Ce traitement algorithmique correspond à la modélisation de la section 11.1.4 (Figure 11.4).

Une deuxième possibilité correspond au cas où le sujet n'identifie pas le motif visuel sur la grille, soit parce qu'il ne perçoit pas la régularité, soit parce qu'il ne sait pas délimiter précisément ce motif. Il exprime alors l'ensemble des actions à faire faire au robot virtuel sous la forme d'une séquence d'instructions, i.e. d'un enchaînement linéaire de blocs d'actions. Le motif algorithmique devient alors observable à travers sa représentation dans le langage de programmation. Il est

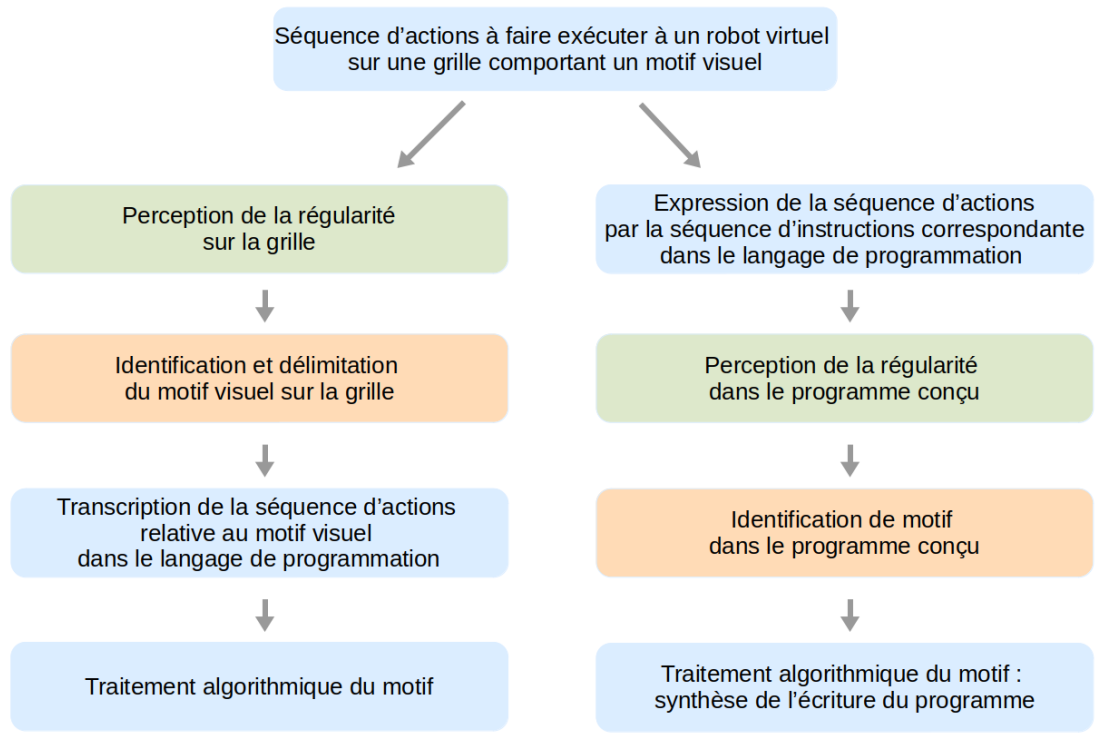


FIGURE 11.9 – Positionnements possibles de l'identification de motifs dans le traitement d'une situation qui comporte de la régularité dans les données

possible de repérer la régularité au sein de la séquence de blocs obtenue, et de synthétiser l'écriture du programme *a posteriori*, en remplaçant les représentations de motif algorithmique répétées par une structure adéquate. Finalement, il s'agit ici encore de l'identification d'un motif qui comporte des éléments visuels (forme et couleur des blocs) et textuels (libellé des blocs).

Dans l'environnement de programmation Algoréa, la contrainte en nombre de blocs favorise largement le premier processus. En effet, même si le sujet a toujours la possibilité de concevoir un programme sous forme de séquence d'instructions, un message d'alerte apparaît au moment où il atteint le nombre de blocs autorisé. De plus, afin de contraindre le passage à la boucle, il ne peut pas exécuter un programme qui comporte plus que le nombre de blocs autorisé. Cette contrainte, nécessaire à la validation des programmes dans un contexte de concours, est une limitation à l'observation des procédures spontanées dans le cadre de notre recherche. C'est l'une des raisons de la mise en œuvre d'une expérimentation complémentaire, pour laquelle nous avons supprimé cette contrainte en nombre de blocs (8.5.2).

11.2.3 Classes de situations relatives à l'identification de motif : analyse a priori

Nous étudions de manière approfondie l'identification de motifs pour la classe de problèmes repérée par la réponse oui à la suite des questions « Y a-t-il de la redondance ? », « Le motif est-il répété consécutivement ? », « Connaît-on le nombre de répétitions ? » de l'arbre de décisions de la figure 11.4. Pour cette classe de problèmes, les motifs visuels (et donc aussi algorithmiques) sont consécutifs et leur nombre est connu. On retrouve dans ce cas la définition de motifs répétitifs de Liljedahl en didactique des mathématiques (Liljedahl, 2004). Cette classe de problèmes est liée au concept de boucle bornée et amène l'utilisation du bloc *répéter n fois* dans un programme implémenté en Scratch.

Selon Vergnaud, « les variables de situation sont un moyen de générer de manière systématique l'ensemble des classes possibles » (Vergnaud, 1991, p.20). Afin de construire une telle catégorisation a priori, nous nous appuyons sur la distinction présentée dans la section 11.1.1 entre motif visuel et motif algorithmique. Pour chacun de ces motifs, visuel puis algorithmique, nous identifions plusieurs paramètres ou caractéristiques, qui correspondent à des variables de situation au sens de Vergnaud.

Pour le motif visuel, nous considérons le nombre de cases qu'il occupe sur la grille, la présence d'éléments saillants visuellement au sein des motifs visuels et la présence d'éléments de décor sur la grille. Pour le motif algorithmique, nous retenons le nombre d'actions constituant le motif et la présence d'actions ne faisant pas partie de la suite de motifs (correspondant aux instructions hors de la boucle). Comme variable de situation, nous étudions aussi le degré de correspondance entre le motif visuel et le motif algorithmique.

11.2.3.1 Variables de situation relatives au motif visuel

Taille du motif

Nous entendons par taille du motif visuel le nombre de cases qu'il occupe sur la grille, que les cases soient de la couleur de fond de la grille ou qu'elles contiennent un élément saillant. Par exemple, sur les figures 11.10 et 11.12, les motifs sont de taille 3.

Disposition des motifs sur la grille

Une autre caractéristique des motifs visuels est leur disposition sur la grille. Papic distingue les motifs répétitifs linéaires et les motifs répétitifs cycliques

(exemple : bordure d'une grille) (Papic, 2007). Pour notre part, nous distinguons trois cas de disposition des motifs visuels sur la grille (Figure 11.10) :

1. Linéaire 1D : les motifs sont disposés linéairement selon la seule dimension horizontale de la grille, impliquant des déplacements du robot virtuel dans une seule direction.
2. Linéaire 2D : les motifs sont disposés linéairement selon les deux dimensions de la grille, en diagonale.
3. Cyclique : les motifs sont disposés de manière cyclique, avec un angle de 90° entre chaque motif.

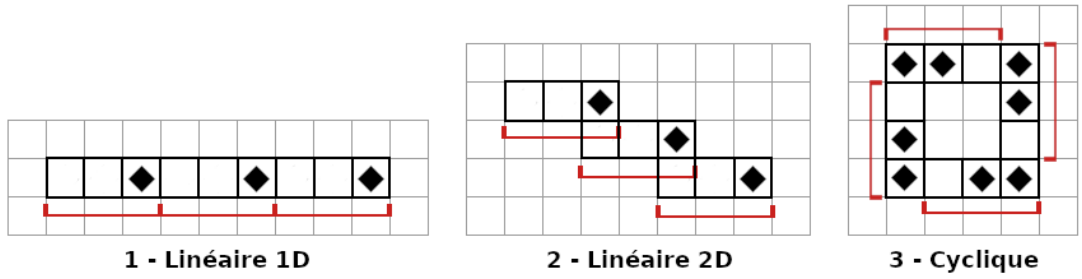


Figure 11.10 – Dispositions possibles des motifs visuels sur la grille

Composition du motif

Pour les motifs linéaires de petite taille, nous étudions la composition du motif. Pour faciliter l'exposition exhaustive, nous utilisons une représentation abstraite des motifs, avec des lettres, comme dans la note du CSEN (Ciccione & Dehaene, 2023) (3.4.2).

Il est à noter que des travaux sur la composition du motif visuel existent déjà en mathématiques. Warren et al. ont investigué si la place des éléments dans le motif a un effet sur la capacité d'élèves de 6 à 8 ans mener des tâches relatives à ces motifs (Warren et al., 2012). Ils étudient les motifs AAB, BAA, AAAB et BAAA. Ils concluent que la position de l'élément unique en première ou en dernière position n'a pas d'effet sur la réussite aux tâches de copier, continuer, ou compléter une suite de motifs. L'expérimentation ne comprenait pas la tâche d'identification du motif. Aussi, la configuration où l'élément unique est intercalé entre les éléments présents plusieurs fois (ABA, ABAA, AABA) est manquante. Il n'y a pas non plus comparaison avec la situation où tous les éléments du motif sont différents (ABC, ABCD).

Dans notre contexte, il est nécessaire d'envisager la composition du motif en deux dimensions, et non en une seule comme dans les travaux cités précédemment.

- Pour un motif de taille 1, il n'y a qu'une possibilité, toutes les cases de la suite de motifs sont identiques (symbole A).
- En dimension un (1D), un motif de taille 2 est forcément une alternance (AB : une case de type A suivie d'une case de type B), il n'y a aussi qu'une seule possibilité. En dimension 2, nous considérons également un motif AA comme de taille 2, si les motifs ne sont pas sur la même ligne ou la même colonne (exemple figure 11.11). Dans ce cas, nous pouvons aussi considérer que nous avons une imbrication de deux motifs de taille 1 (A) dans ce motif de taille 2.
- Motifs de taille 3 : AAA, AAB, BAA, ABA, ABC ; nous avons 5 possibilités de motif différents, suivant la position de l'élément unique, en plus des cas où les trois cases du motif sont soit identiques, soit différentes.
- Motifs de taille 4 : AAAA, AAAB, AABA, ABAA, BAAA, ABAB, ABBA, ABCA, ABAC, AABC, BAAC, BACA, BCAA, ABCD ; nous avons déjà une explosion du nombre de possibilités, c'est pourquoi nous ne poussons pas l'investigation à des motifs de taille supérieure à 4.

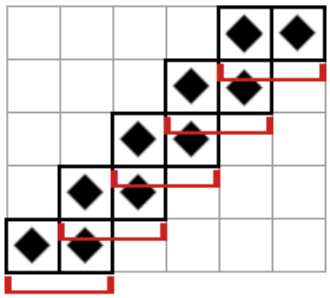


FIGURE 11.11 – Exemple de motif AA considéré comme de taille 2 sur une grille en deux dimensions

Fonctions des éléments saillants

Sur la grille, nous distinguons les cases qui contiennent un élément saillant visuellement (objet, marque) par rapport aux cases de la couleur du fond de la grille.

Ces éléments saillants peuvent soit constituer des éléments du motif, souvent associés à une action que le robot doit exécuter, soit constituer des éléments de décor. Sur l'exemple de la figure 11.12, les carrés (sur la pointe) appartiennent aux motifs alors que les ronds sont des éléments de décor. Il est à noter que dans certains cas, les éléments de décor peuvent avoir une fonction d'obstacle, interdisant l'accès de la case au robot.

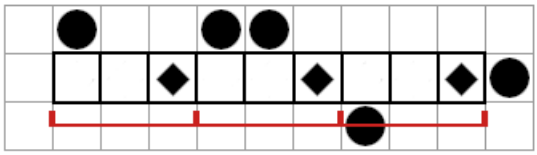


FIGURE 11.12 – Fonctions des éléments saillants : élément du motif (◆) vs élément de décor (●)

Cases adjacentes à un motif

Nous nous posons la question de la prise de repères pour distinguer visuellement deux motifs consécutifs. Plusieurs cas se présentent pour les cases adjacentes à un motif, d'une part pour les cases ne faisant pas partie de la suite de motifs, d'autre part pour les cases adjacentes appartenant à des motifs différents.

Pour les cases adjacentes à la suite de motifs, nous distinguons deux cas (Figure 11.13) :

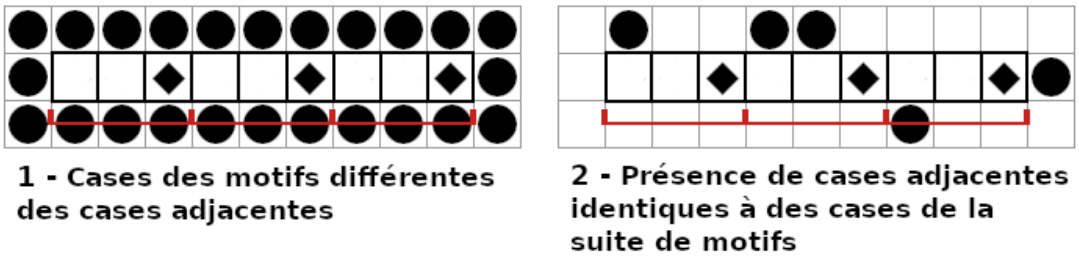


FIGURE 11.13 – Deux cas pour les cases adjacentes à la suite de motifs

1. Les cases de la suite de motifs sont toutes différentes des cases adjacentes à cette suite de motifs, rendant la suite de motifs facilement repérable.
2. Le motif est constitué de cases contenant un élément saillant et de cases de la couleur de fond de la grille. D'autres cases adjacentes au motif sont aussi de la couleur du fond de la grille.

Pour les cases adjacentes appartenant à des motifs différents, nous distinguons trois cas. Sur la figure 11.14, nous avons jauni les cases adjacentes aux deux premiers motifs :

1. Ces cases sont différentes visuellement, rendant facilement repérable la limite entre deux motifs.
2. Ces cases sont identiques, de la couleur de fond de la grille.
3. Ces cases sont identiques et contiennent le même élément saillant.

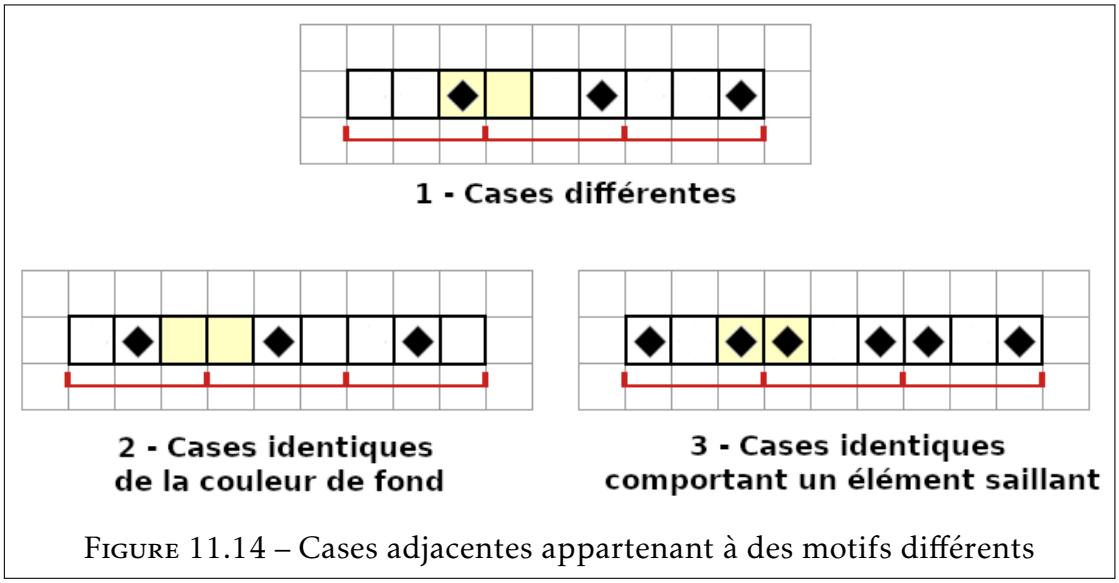


FIGURE 11.14 – Cases adjacentes appartenant à des motifs différents

Nous avons considéré le motif optimal, c'est à dire celui qui permet d'en trouver le plus dans une suite de motifs donnée. Il est à noter qu'en décalant, on peut toujours se ramener au premier cas. La suite des motifs comprend alors un motif de moins, et il reste des éléments avant et après la suite.

11.2.3.2 Variables de situation relatives au motif algorithmique

Nous considérons la représentation du motif algorithmique en langage Scratch, qui prend la forme d'un agencement de blocs. Nous retrouvons pour le motif algorithmique des caractéristiques similaires au motif visuel et d'autres qui lui sont propres, car intimement liées aux actions du robot virtuel.

Comme pour le motif visuel, nous nous questionnons sur la taille du motif algorithmique. Deux manières différentes sont possibles pour définir cette métrique : le nombre d'actions et déplacements que doit exécuter le robot, ou le nombre de blocs Scratch utilisé pour représenter ce motif. Le nombre obtenu

est différent si l'écriture du motif algorithmique a fait l'objet d'une synthèse (boucles imbriquées par exemple). Comme les situations que nous étudions ne contraignent pas une boucle imbriquée, le motif algorithmique peut toujours être exprimé avec une séquence de blocs Scratch (sans structure de contrôle). Nous choisissons donc le nombre de blocs de cette séquence comme définition de la taille du motif algorithmique, ce qui revient au nombre d'actions et déplacements du robot.

Pour le motif algorithmique, nous retrouvons la caractéristique de composition du motif. Lorsque l'écriture de la suite de motifs algorithmiques n'est pas synthétisée sous forme de boucle, nous retrouvons aussi la problématique des blocs adjacents n'appartenant pas au même motif.

Par ailleurs, la présence d'actions ne faisant pas partie de la suite de motifs s'envisage spécifiquement pour le motif algorithmique. Ces actions correspondent aux instructions hors de la boucle. En particulier, des instructions avant la boucle peuvent être nécessaires pour placer le robot virtuel de manière adéquate pour aborder la suite de motifs visuels.

11.2.3.3 Degré de correspondance entre motif visuel et motif algorithmique

Comme variable de situation, nous étudions aussi le degré de correspondance entre le motif visuel et le motif algorithmique. En effet, suivant le système de contrôle du robot virtuel et les caractéristiques de la grille, plusieurs cas se présentent :

- La correspondance entre motif visuel et motif algorithmique est stricte, bijective. Chaque action de déplacement du robot est repérable par la limite entre deux cases et les autres actions sont identifiables par un élément saillant visuellement. Ce sont les situations où le déplacement du robot n'est possible que dans une seule direction et les situations de déplacement en orientation absolue (nord, sud, est, ouest).
- La correspondance entre motif visuel et motif algorithmique est partielle. Pour ces situations, les motifs visuels sont identiques, disposés linéairement sur la grille, mais plusieurs états du robot virtuel sur une même case sont visuellement identiques. Ce sont les situations en orientation relative pour lesquelles les actions de pivotement du robot ne sont pas observables avant l'exécution du programme.
- Les motifs visuels ne sont identiques qu'à une rotation d'un quart de tour près. Ce sont des situations en orientation relative pour lesquelles la disposition des motifs est cyclique.
- Les motifs visuels ne sont pas identiques. On ne peut alors pas strictement parler de motifs visuels, au sens où il est nécessaire de faire abstraction de

certains éléments visuels pour repérer ce motif. Soit des éléments saillants ou des éléments de décor sont équivalents mais visuellement différents, soit plusieurs motifs visuels sont partiellement superposés, perturbant la lisibilité de chacun d'eux. Dans ce cas, la correspondance entre motif visuel et motif algorithmique est entravée.

Dans toute cette section, nous nous sommes placés du point de vue de l'expert afin de réaliser une analyse a priori des situations que nous mettons en place dans notre dispositif expérimental. Nous remarquons que la correspondance entre motif visuel et motif algorithmique est fortement en lien avec la disposition des motifs sur la grille. En conséquence, nous regroupons les deux analyses par la suite.

11.2.4 Synthèse du travail théorique

Nous avons montré à travers ce travail théorique ancré dans le cadre d'analyse de Vergnaud que le concept de motif est à la fois précis et opérationnel, et à la fois très général et indépendant du langage et de la technologie utilisés. Premièrement, nous avons distingué deux instances de motif dans le champ de la programmation, motifs qui sont essentiels dans l'automatisation des traitements.

Le **motif algorithmique** est une partie du traitement qui est repérable dans l'ensemble des traitements à faire exécuter par la machine car répété plusieurs fois, consécutivement ou non, à l'identique ou avec des variations prédictibles.

Le **motif inhérent** est une partie des données qui est repérable dans l'ensemble des données sur lesquelles va opérer un algorithme, car répétée plusieurs fois, consécutivement ou non, à l'identique ou avec des variations prédictibles. La modalité du motif inhérent dépend du format des données. Exemples : **motif visuel, motif sonore**

Nous avons ensuite construit un champ conceptuel des bases de l'algorithmique dont le concept de motif est un concept structurant. Au sein de ce champ conceptuel, nous avons ensuite établi une classification a priori des situations pour le niveau d'analyse sur lequel nous travaillons, à savoir la programmation avec des blocs Scratch par des sujets de 7 à 15 ans. Nous avons approfondi ce travail théorique pour le cas particulier de la programmation d'un personnage virtuel sur une grille, situation type répertoriée dans les programmes scolaires français des cycles 2 à 4. Pour ce cas, nous avons analysé le motif visuel sur la

grille et le motif algorithmique. Nous avons alors catégorisé les caractéristiques et la correspondance entre ces deux instances de motif.

Appréhender comment les sujets traitent les situations de programmation associées à des motifs visuels afin de déterminer quelles variations dans les caractéristiques entraînent un changement dans le traitement et donc une rupture distinguant deux classes de situations est l'objectif de la partie expérimentale de notre recherche.

11.3 Résultats expérimentaux à l'échelle nationale

Pour cette section, nous nous appuyons sur les données collectées à l'échelle nationale. Une partie des résultats ont fait l'objet de communications en 2022 lors du colloque Didapro9 (Léonard, Secq et al., 2022) et de la conférence EC-TEL (Léonard, Peter, Secq & Fluckiger, 2022). Ces résultats ont été mis à jour en y incorporant les données issues de l'édition 2022 du concours Algoréa. Nous avons aussi renforcé et ajusté notre analyse en procédant à des tests de statistique inférentielle et à l'adjonction d'un modèle de régression linéaire. L'ensemble de l'étude est disponible dans l'annexe 4. Nous en présentons les principaux résultats dans cette section.

Pour la présente analyse, nous retenons 139 puzzles issus des éditions 2018 à 2022 du concours Algoréa, qui constituent les individus statistiques de l'étude (Figure 11.15).

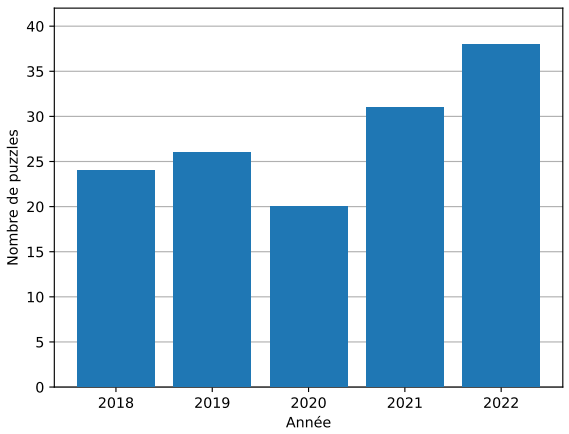


Figure 11.15 – Répartition des puzzles impliquant une boucle selon les tours

Leur point commun est que la séquence d'actions à faire exécuter au robot virtuel comporte une régularité qu'il est nécessaire d'identifier pour résoudre le

problème. En effet, le nombre de blocs disponibles contraint d'utiliser le bloc *répéter n fois*. La solution de référence implique donc une boucle ou plusieurs boucles en séquence (mais elle ne nécessite pas de boucles imbriquées).

À l'échelle nationale, nous disposons de la fréquence de réussite pour chaque puzzle et chaque niveau de classe (8.5.1.1). Ces fréquences de réussite nous permettent notamment d'étudier l'influence des variables de situation présentées dans la section 11.2.3 en construisant pour chaque puzzle la courbe de la fréquence de réussite en fonction du niveau scolaire, de manière similaire à l'étude de Vergnaud et Durand sur les problèmes additifs (Vergnaud & Durand, 1976) (9.1).

11.3.1 Étude de la robustesse des données

De manière préliminaire à l'étude sur l'identification de motifs, nous procédons à quelques analyses d'ordre plus général.

Nous vérifions la robustesse de nos données concernant les fréquences de réussite. Lorsque l'on considère tous les niveaux de classe ensemble, un test d'indépendance de *chi2* nous permet de vérifier que tous les écarts de 0,05 de fréquence de réussite entre deux situations sont statistiquement significatifs avec une *p-value* inférieure à 0,01. Pour un niveau de classe particulier, un écart de fréquence de réussite de 0,05 entre deux situations est significatif pour le collège (*p-value*<0,05). Seules quelques situations pour le niveau élémentaire dont l'effectif est plus réduit amènent à des écarts de fréquences de réussite de moins robustes statistiquement (voir l'annexe 5 pour plus de détails).

La figure 11.16 confirme, de manière attendue, que la fréquence de réussite baisse lorsque le nombre d'instructions dans la solution de référence augmente. La corrélation est avérée statistiquement (taux de corrélation linéaire -0,77 ; *p-value*<0,01 au test de Bravais-Pearson). La fréquence de réussite baisse aussi selon la version du problème, de une à quatre étoiles, ce qui confirme, à quelques exceptions près, le bon calibrage de la difficulté a priori. Néanmoins, nous remarquons une dispersion des valeurs importante sur l'axe vertical, parfois de plus de 0,5, ce qui nous indique que d'autres variables de situation ont un effet sur la fréquence de réussite. L'identification et l'étude de ces variables sont l'objet des sections suivantes.

Dans une première étape, pour chaque caractéristique relevée, nous calculons la moyenne et la médiane des fréquences de réussite ainsi que l'écart interquartile comme indicateurs de la distribution des données. Nous appliquons les tests statistiques de comparaison de moyennes adéquats pour la distribution des échantillons ainsi constitués. Ces analyses visent à relever les variables qui semblent avoir le plus d'effet sur la fréquence de réussite. Cependant, au terme de cette étape, il reste à neutraliser l'effet de la longueur de la solution de

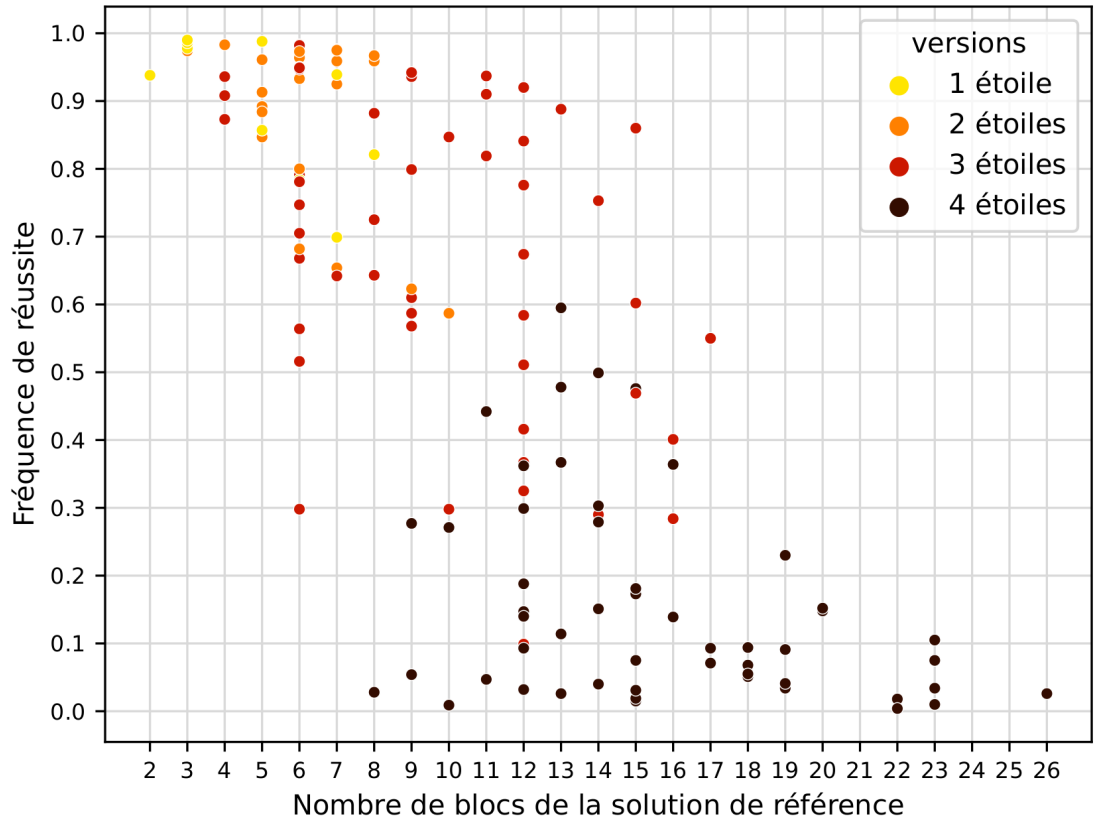


FIGURE 11.16 – Diagramme de dispersion de la fréquence de réussite en fonction du nombre d'instructions de la solution de référence

référence, et à déterminer le poids respectif de chaque variable. À cette fin, dans un second temps, nous construisons un modèle de régression linéaire en nous appuyant sur les analyses de la première étape, puis nous en étudions les coefficients.

11.3.2 Résultats concernant le motif visuel

Dans cette section, c'est l'aspect visuel du motif qui importe, indépendamment des actions que doit exécuter le robot virtuel. Nous avons étudié la taille du motif, sa composition, la fonction des éléments saillants (décor vs utilité pour la mission du robot) et la disposition des motifs les uns par rapport aux autres. Nous présentons les résultats significatifs.

Taille du motif

Concernant le nombre de cases sur lesquelles s'étend le motif visuel, nous distinguons deux classes de situations de manière très marquée, selon que le motif visuel s'étend sur une ou plusieurs cases de la grille.

Un test de Shapiro-Wilk indique que les distributions des groupes ainsi constitués ne sont pas normales. Nous appliquons donc un test de Mann-Whitney U qui permet de comparer les moyennes des deux distributions dans ce cas de figure. Le résultat de ce test nous indique que les moyennes des fréquences de réussite des deux échantillons sont très significativement différentes (p-*value* = $1\text{x}10^{-17}$). Pour chaque échantillon, la figure 11.17 montre l'évolution de la moyenne, de la médiane et de l'écart interquartile selon les niveaux scolaires.

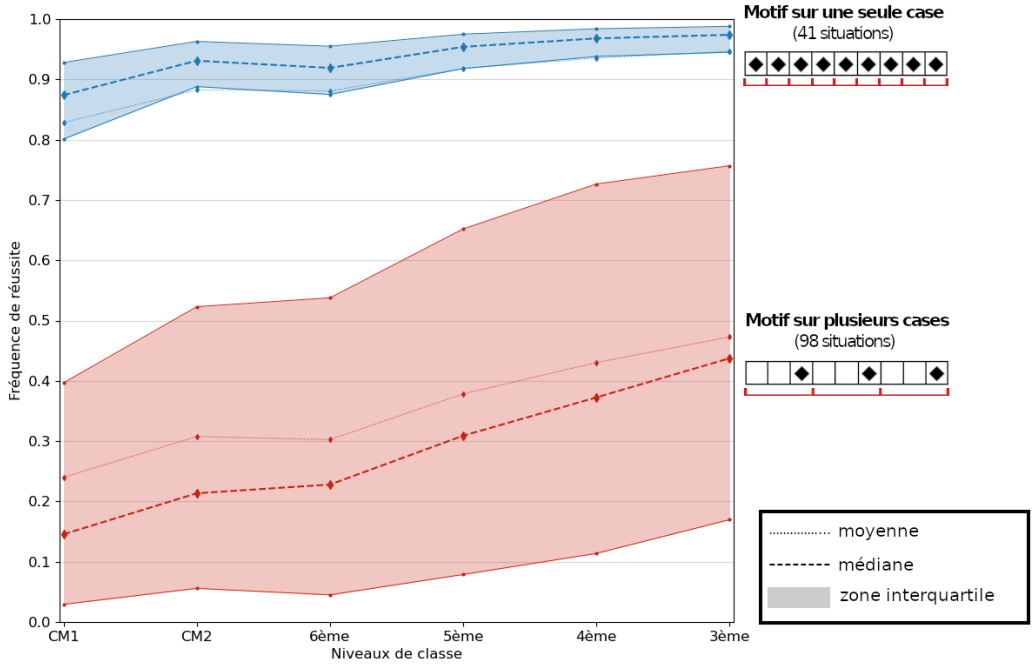


FIGURE 11.17 – Fréquence de réussite selon que le motif visuel s'étend sur une ou plusieurs cases

Lorsque le motif visuel est constitué d'une seule case de la grille, la fréquence de réussite des puzzles est élevée dès l'école élémentaire. L'écart interquartile est faible, ce qui signifie que cette caractéristique est prégnante dans l'explication de la fréquence de réussite. En conséquence, nous considérons la difficulté de ces

puzzles comme principalement déterminée par cette caractéristique, et écartons ceux-ci de la suite de notre étude.

En revanche, l'écart interquartile est beaucoup plus élevé si le motif visuel s'étend sur plusieurs cases. Dans ce cas, d'autres variables contribuent significativement à la valeur de la fréquence de réussite. Nous poussons plus loin nos investigations pour les 98 situations avec cette caractéristique, afin d'appréhender quelles variables ont un effet significatif sur la difficulté des problèmes.

Fonctions des éléments saillants

Nous montrons l'effet de la présence d'éléments de décor sur la grille. Suivant la manière dont ils sont disposés, les éléments de décor peuvent constituer une aide ou une source de difficulté. Nous divisons les 98 situations selon quatre modalités en rapport avec la fonction des éléments saillants sur la grille : « pas d'élément de décor sur la grille », « les éléments de décor contraignent le parcours du robot », « des éléments de décor sont présents sans contraindre le parcours du robot », « des éléments de décor ou des éléments saillants impliquant une superposition de motifs, rendent les motifs visuellement différents ».

Nous étudions la médiane (Q_2) et l'écart interquartile (EI) pour chacune des modalités, et nous représentons visuellement les distributions avec l'évolution selon les niveaux de classe (Figure 11.18). De manière complémentaire, nous appliquons les tests de comparaison de moyennes afin de nous assurer de la robustesse statistique de nos résultats. Les distributions des échantillons n'étant pas toutes normales (test de Shapiro-Wilk), nous appliquons le test de Kruskal-Wallis, non paramétrique, pour déterminer si les moyennes des fréquences de réussite des différents groupes sont significativement différentes (équivalent de l'ANOVA pour des distributions non normales). Le résultat de ce test nous indique de manière robuste qu'il y a une différence significative entre deux groupes au moins (statistique de Kruskal-Wallis : 36,5 ; *p-value* : $6\text{x}10^{-8}$).

Lorsque les éléments de décor contraignent complètement le parcours du robot (Q_2 : 0.58, EI : 0.27), ils pourraient constituer une aide par rapport à des situations sans éléments de décor (Q_2 : 0.45, EI : 0.57). La différence de moyenne de fréquence de réussite est cependant très faible entre les deux groupes, non significative statistiquement (p-valeur ajustée de l'ordre de 0,5 au test de Tukey).

Si des éléments saillants n'ont pas d'autre fonction que de décorer, la fréquence de réussite chute (Q_2 : 0.15, EI : 0.30). La différence de moyenne est significative avec les deux groupes précédents (p-valeur ajustée de l'ordre de 0,01 au test de Tukey). Dans ce cas, les éléments de décor constituent des distracteurs qui augmentent la difficulté de la situation. Pour ces trois premières modalités, les courbes de la médiane sont sensiblement parallèles, ce qui indique

une progression similaire à travers les niveaux scolaires. Cependant, l'écart interquartile est relativement important, ce qui signifie que la caractéristique n'est probablement pas la plus prégnante pour l'appréciation de la difficulté de la situation.

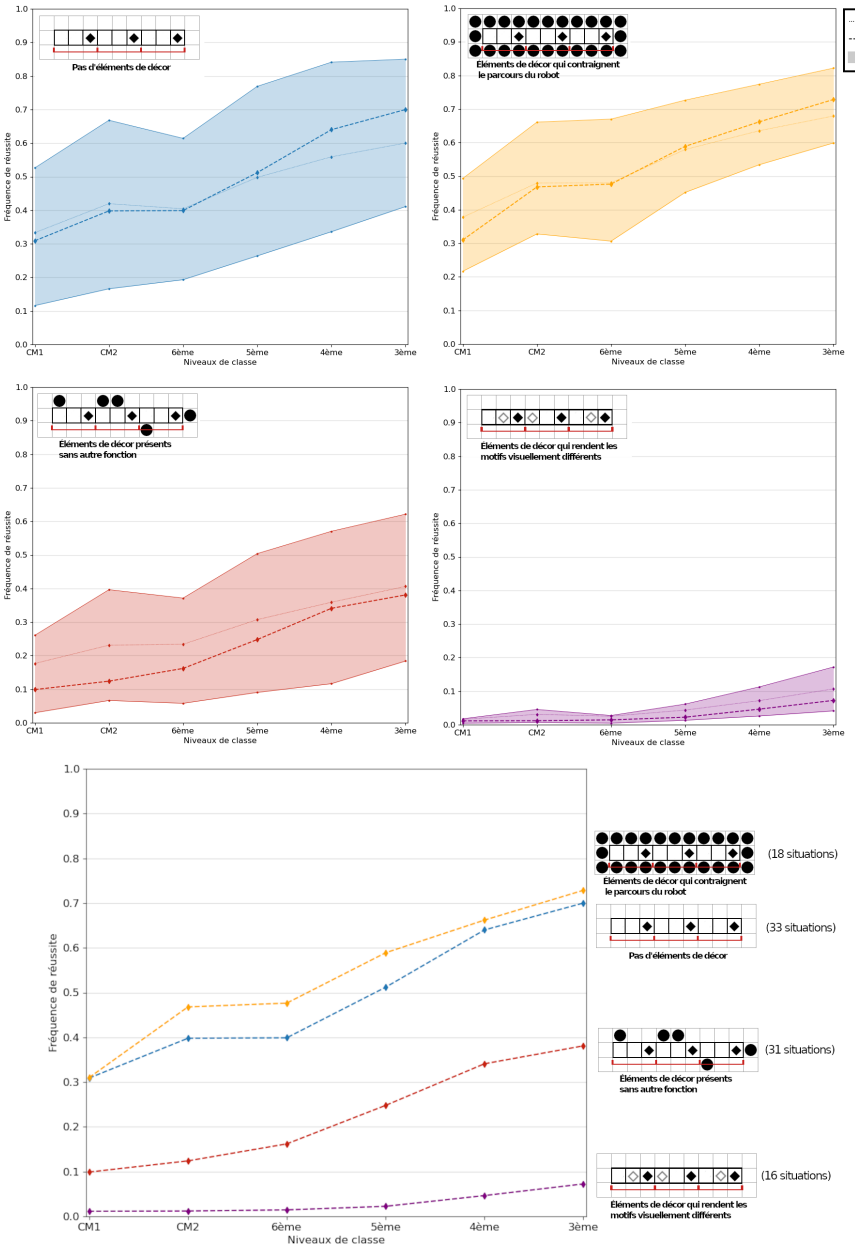


FIGURE 11.18 – Fréquence de réussite pour les différentes fonctions des éléments saillants

En revanche, la difficulté devient massive lorsque les éléments de décor rendent certains motifs visuellement différents (Q_2 : 0.03, EI : 0.03). La différence de moyenne avec les trois autres groupes est très significative (p-valeurs ajustée au test de Conover-Iman inférieures à 10^{-3}). L'écart interquartile est extrêmement faible, ce qui nous amène à penser que cette caractéristique est déterminante dans la difficulté de ces situations. La courbe de la médiane plus aplatie indique que la difficulté résiste plus dans le temps. Néanmoins, les programmes solutions de ces situations comportent aussi un nombre plus important de blocs, ce qui demande une étude plus approfondie pour distinguer l'effet respectif de ces variables. Pour l'étude des caractéristiques suivantes, nous ne gardons que les situations qui comportent un motif visuel au sens strict, qui sont aussi celles pour lesquelles l'écart interquartile est important. Et nous écartons donc celles pour lesquelles la difficulté principale s'avère être de faire abstraction d'éléments de la grille pour percevoir le motif, en plus de la longueur de la solution.

Cas particulier des adjacentes identiques n'appartenant pas au même motif

Nos investigations concernant la composition des motifs n'a pas abouti à des résultats significatifs, excepté pour le cas particulier des cases adjacentes n'appartenant pas au même motif. Nous en présentons donc les résultats, en prenant en compte les 82 situations pour lesquelles les motifs sur la grille sont visuellement identiques et s'étendent sur plusieurs cases.

Nous analysons les trois modalités suivantes : « pas de cases adjacentes identiques qui appartiennent à des motifs différents », « des cases identiques, de la couleur de fond de la grille, appartiennent à des motifs différents », et « des cases identiques comportant un élément saillant appartiennent à des motifs différents » (Figure 11.14).

Lorsque des cases identiques adjacentes comportant un élément saillant visuellement n'appartiennent pas au même motif, la fréquence de réussite est basse (Q_2 : 0.19, EI : 0.23), et ce de manière plus marquée chez les plus jeunes élèves.

En revanche, lorsque ce sont deux cases de la couleur du fond de la grille qui sont adjacentes et appartiennent à des motifs différents, la fréquence de réussite (Q_2 : 0.46, EI : 0.47) est proche de celle des situations sans cases adjacentes identiques appartenant à des motifs différents (Q_2 : 0.58, EI : 0.47). Un test de Kruskal-Wallis (Statistique : 9.7 ; *p-value* : 0,008), suivi d'un test de Tukey, confirme le caractère significatif de ces résultats.

Ce constat nous amène à penser que les éléments saillants sont pris comme points de repère privilégiés lors de l'identification du motif visuel. Des éléments

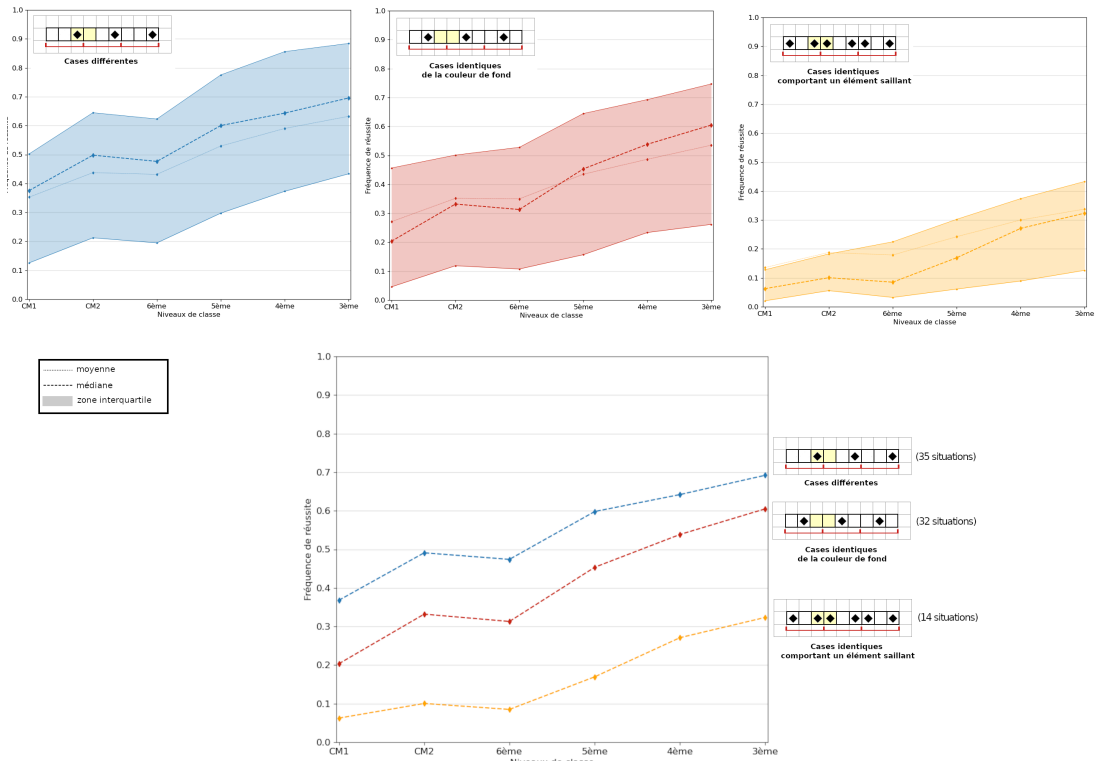


FIGURE 11.19 – Étude des cases adjacentes n'appartenant pas au même motif visuel

saillants identiques sur des cases adjacentes sont perçus comme faisant partie d'une même entité visuelle. Lorsque ceux-ci n'appartiennent pas au même motif, cela rend le motif moins lisible et donc son identification plus difficile.

Disposition des motifs sur la grille

Nous considérons à nouveau les 82 situations pour lesquelles les motifs sur la grille sont visuellement identiques et s'étendent sur plusieurs cases. Nous étudions la disposition des motifs sur la grille, en distinguant les modalités « motifs disposés linéairement sur une seule ligne de la grille (1D) », « motifs disposés linéairement en deux dimensions », et « motifs disposés de manière cyclique » (Figure 11.20).

Lorsque les motifs sont disposés sur une seule ligne, impliquant des déplacements du robot dans une seule direction, la fréquence de réussite est assez élevée (Q_2 : 0.79, EI : 0.26), et ce dès l'école élémentaire. Il faut cependant noter que les solutions de référence pour cette modalité sont aussi en moyenne beaucoup

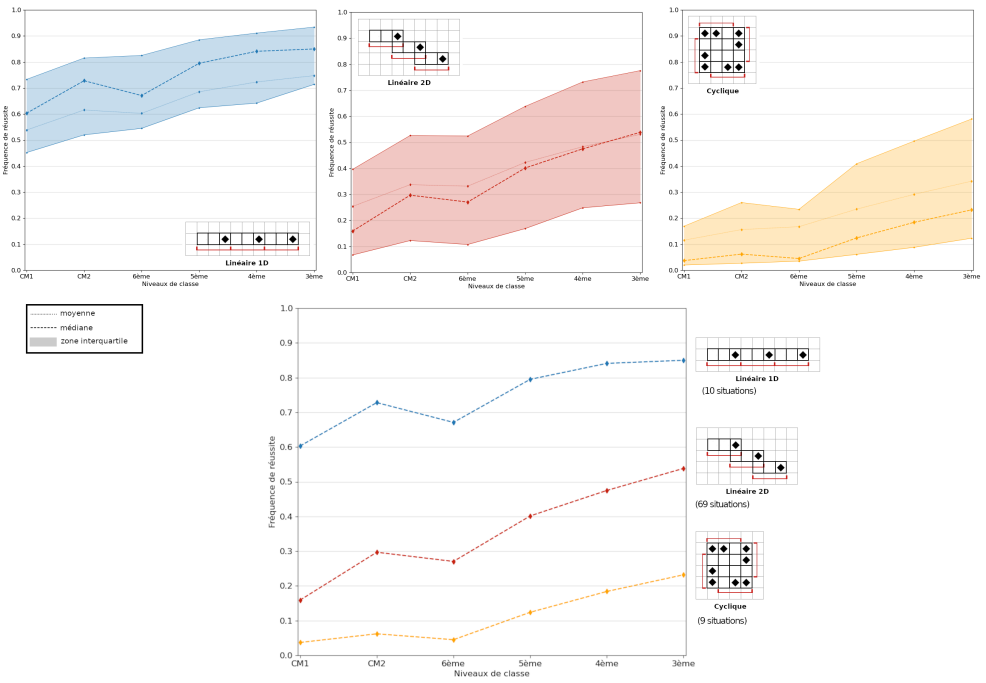


FIGURE 11.20 – Fréquence de réussite pour les différents types de disposition des motifs

plus courtes, ce qui atténue la portée de ce résultat. La disposition des motifs sur les deux dimensions de la grille rend la situation plus difficile. C'est le cas lorsque la disposition est linéaire (Q_2 : 0.38, EI : 0.47), et encore plus lorsqu'elle est cyclique (Q_2 : 0.14, EI : 0.35).

Pour la disposition cyclique, nous remarquons que l'écart interquartile tend à s'étendre vers le haut en fin de collège, ce qui signifie que certaines situations de ce type commencent à être mieux appréhendées. Après consultation de ces situations, nous constatons que ce sont celles où le parcours du robot est contraint par les éléments de décor, qui constituent dans ce cas très probablement une aide pour identifier le caractère cyclique de la disposition des motifs.

Un test de Kruskal-Wallis (statistique : 13,1 ; *p-value* : 0,001), suivi d'un test de Tukey, indique que la différence de fréquence de réussite est significative entre une disposition 1D et une disposition linéaire en 2D (*p-value* : 0,02), entre une disposition 1D et une disposition cyclique (*p-value* : 0,001). Pour la différence entre disposition linéaire et disposition cyclique, la *p-value* (0,07) est légèrement supérieur au seuil de 0,05 admis pour considérer la différence comme significative.

Synthèse

L'étude des caractéristiques des motifs visuels montre que la nature des éléments présents sur la grille a un effet sur la difficulté de la situation. Plus le motif est facilement isolé visuellement, et plus la situation est bien résolue. Les cases de la grille sont les unités les plus facilement discernables sur celle-ci. Lorsque la case correspond aussi à la délimitation du motif visuel, les situations de programmation sont très bien réussies. Lorsque les éléments de décor délimitent le parcours du robot, et partiellement le motif visuel de manière concomitante, ces éléments constituent une aide dans la résolution de la situation. À l'inverse, les facteurs qui perturbent la lisibilité du motif visuel impactent négativement la fréquence de réussite de la situation : éléments de décor qui rendent les motifs visuellement différents, éléments saillants identiques sur des cases adjacentes n'appartenant pas au même motif, disposition cyclique des motifs qui ne sont alors identiques qu'à une rotation d'un quart de tour près.

11.3.3 Résultats concernant le motif algorithmique

La représentation du motif algorithmique est exprimée en langage Scratch. Elle dépend à la fois du nombre de cases du motif visuel, des actions à faire réaliser par le robot (ramasser et/ou déposer des objets) et de son système d'orientation. Nous montrons que le taux de réussite est corrélé à la taille dde cette représentation du motif algorithmique, c'est à dire au nombre d'instructions dans la boucle (taux de corrélation linéaire de -0,78).

Système d'orientation du robot virtuel

Trois systèmes d'orientation sont possibles pour le robot virtuel : absolue, relative et hybride (7.1.2). Pour les grilles où le robot ne peut se déplacer que dans une seule direction, nous les plaçons dans la même catégorie que celles en orientation absolue. En effet, ce qui différencie principalement les déplacements du robot dans un système d'orientation relative par rapport à un système d'orientation absolue sont les actions de pivotement. Or, même si le robot est orienté, ces actions de pivotement sont absentes lorsque le robot ne se déplace que dans une seule direction. Nous distinguons donc deux classes de situations selon la nécessité ou non de programmer des actions de pivotement du robot.

Un test de Mann-Whitney U, qui permet de comparer les distributions de deux échantillons indépendants lorsque les données ne sont pas distribuées normalement, indique que la différence des moyennes de fréquence de réussite entre les deux groupes, de 0,30 pour l'ensemble de les niveaux scolaires, est significative (*p-value* : 0,007). Cependant, l'écart interquartile est très important,

que ce soit pour le groupe où aucun pivotement du robot n'est nécessaire (Q_2 : 0,66), ou pour le groupe où programmer au moins un pivotement du robot est requis (Q_2 : 0,70). Nous en déduisons que cette caractéristique influe sur la difficulté de la situation, mais sans être déterminante.

Présence d'instructions hors de la boucle

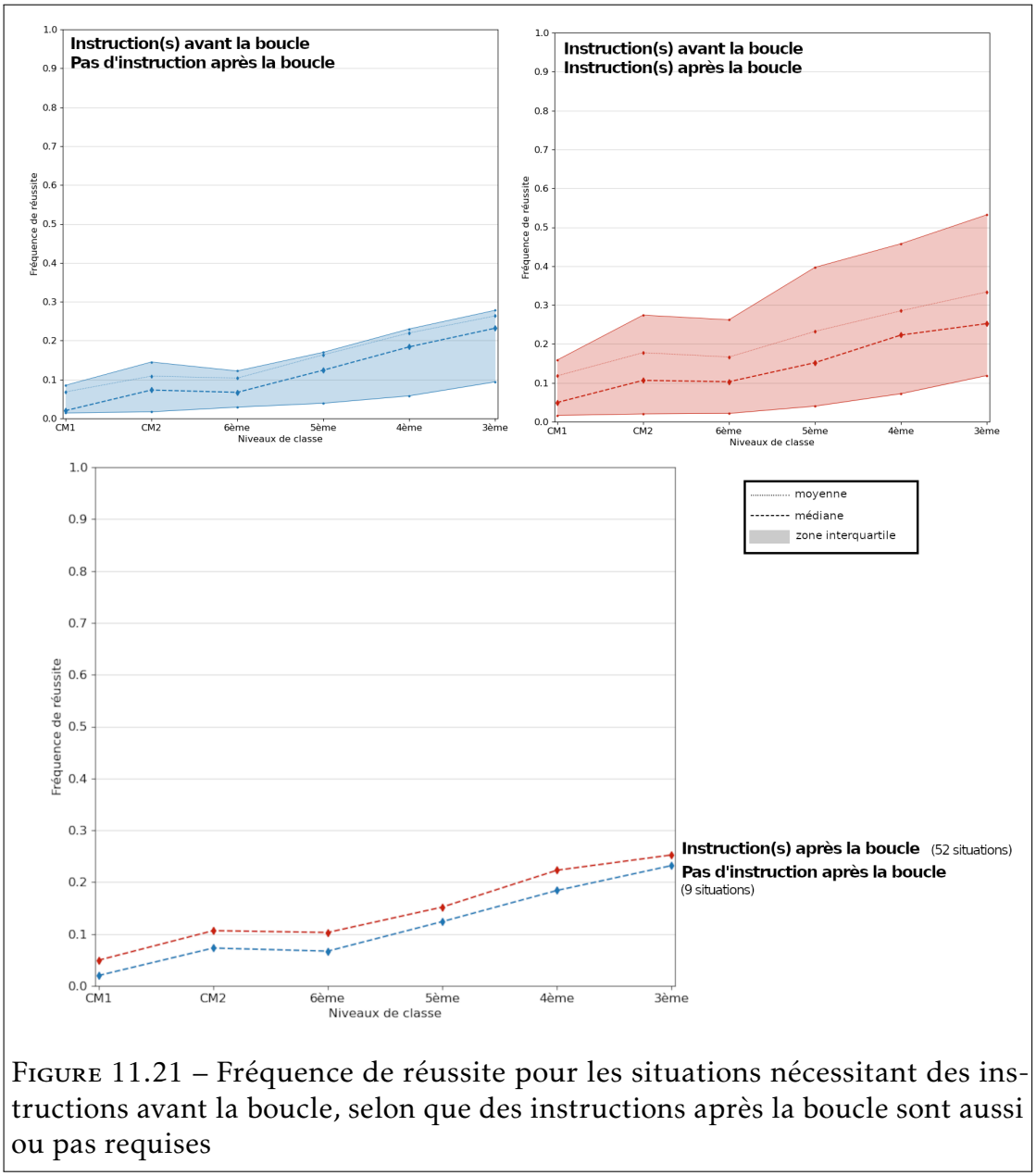


FIGURE 11.21 – Fréquence de réussite pour les situations nécessitant des instructions avant la boucle, selon que des instructions après la boucle sont aussi ou pas requises

Pour les 98 situations pour lesquelles le motif visuel s'étend sur plusieurs cases, nous analysons l'impact d'instructions en-dehors de la boucle.

La moyenne du nombre d'instructions de la solution de référence, très différent selon que des instructions sont requises en-dehors de la boucle (environ 15) ou non (environ 6), a trop d'impact pour pouvoir analyser l'effet des autres variables.

Nous parvenons seulement à montrer que lorsque des instructions sont requises avant la boucle, la nécessité d'instructions également après la boucle ne semble pas ajouter de difficulté supplémentaire (Figure 11.21).

Synthèse

Les caractéristiques du motif algorithmique semblent avoir une moindre influence sur la fréquence de réussite que les caractéristiques du motif visuel. En l'occurrence, nous ne parvenons pas à isoler l'effet de variables relatives au motif algorithmique de la même manière que pour le motif visuel.

11.3.4 Modèle de régression linéaire

Au terme des analyses précédentes, il apparaît que beaucoup de paramètres influent sensiblement sur la fréquence de réussite, sans que nous soyons en mesure de déterminer leur poids respectif. Afin d'approfondir cet aspect, nous reprenons les caractéristiques les plus saillantes pour les motifs visuels. Nous y adjoignons les deux variables étudiées pour le motif algorithmique, afin de tenter de positionner leur effet par rapport à celui des caractéristiques du motif visuel. Dans ce but, nous construisons un modèle de régression linéaire avec prétraitement. Pour ne pas surcharger le modèle, nous regroupons certaines caractéristiques, ce qui nous donne quatre variables qualitatives nominales et deux variables quantitatives discrètes.

- Variables qualitatives
  - disposition des motifs : sur une case / sur plusieurs cases avec disposition 1D / sur plusieurs cases avec disposition linéaire en 2D / sur plusieurs cases avec disposition cyclique
  - fonction des éléments saillants : pas d'éléments de décor et pas de perturbation visuelle du motif / éléments de décor qui contraignent le parcours du robot / éléments de décor présents sans contraindre le parcours du robot / éléments saillants qui perturbent la lisibilité du motif (motifs visuellement différents, cases adjacentes identiques comportant un élément saillant qui ne font pas partie du même motif)

  - déplacements du robot : nécessité de pivotement(s) du robot / pas de pivotement nécessaire
  - instructions avant la boucle : pas d'instruction avant la boucle / nécessité d'instructions avant la boucle

- Variables quantitatives :

  - nombre d'instructions de la solution de référence
  - nombre d'instructions dans la représentation du motif algorithmique

Nous évaluons ce modèle en utilisant une validation croisée. Le coefficient de détermination (R^2) calculé sur les données d'entraînement indique que le modèle explique 82% de la variation de fréquence de réussite sur celles-ci. Comme le R^2 tombe à 58% sur les données de test, la capacité de généralisation du modèle est moyenne. Une hypothèse est que le modèle est encore trop complexe pour le volume de données dont nous disposons (139 situations) (Figure 11.22). C'est pourquoi, nous adjoignons un modèle un peu moins précis, dans le sens où toutes les variables n'admettent que deux valeurs, mais dont la capacité de généralisation de 70% sur les données de test est cette fois correcte (Figure 11.23). Une perspective à court terme est d'augmenter le volume de données en entrée des modèles en y ajoutant celles issues des éditions de 2023 et 2024 du concours Algoréa.

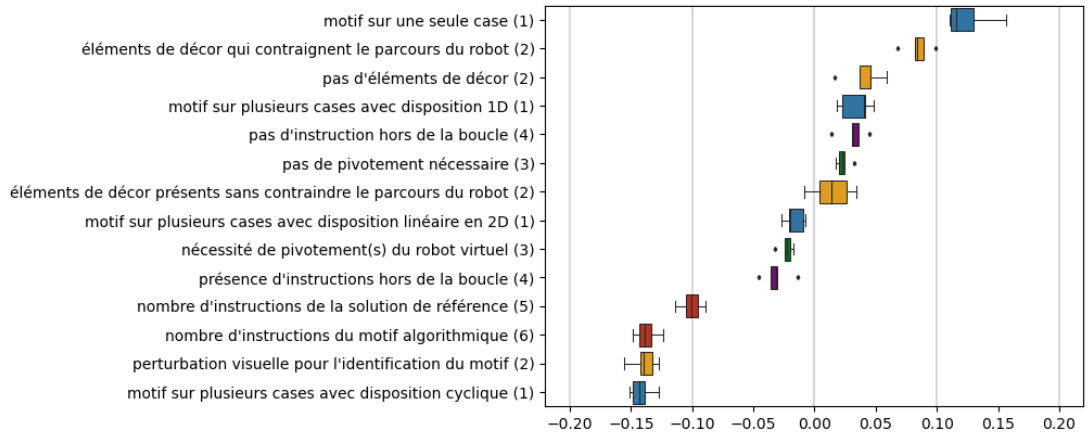


FIGURE 11.22 – Coefficients du modèle de régression linéaire implémenté en considérant les caractéristiques les plus saillantes des situations

Nous analysons les coefficients de régression pour disposer de l'importance relative des caractéristiques identifiées dans la prédiction de la fréquence de réussite. Sur les figures 11.22 et 11.23, les couleurs des boîtes sont relatives aux

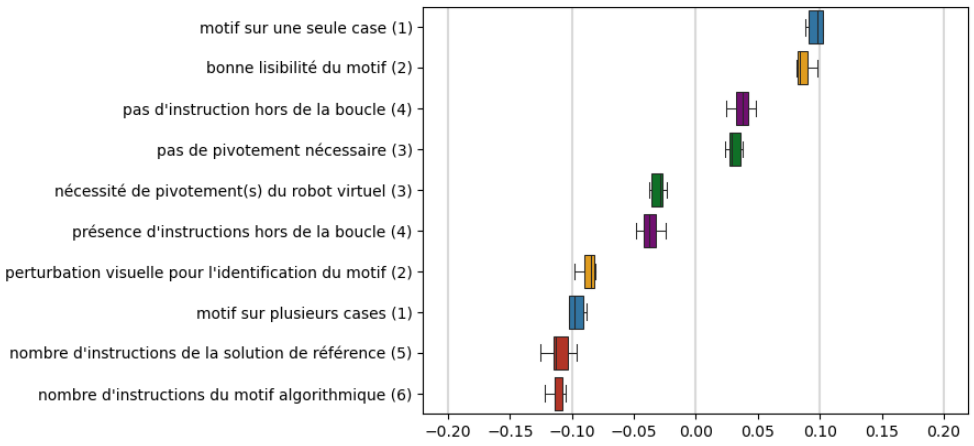


FIGURE 11.23 – Coefficients du modèle de régression linéaire simplifié

variables entrées dans le modèle. Les variables quantitatives sont représentées en rouge (5, 6). Pour les variables qualitatives, nous représentons la disposition des motifs (1), la fonction des éléments saillants (2), les déplacements du robot (3), les instructions hors de la boucle (4) respectivement en bleu, orange, vert et violet sur la figure 11.22. Sur la figure 11.23 du modèle simplifié, la disposition des motifs est synthétisée en motif sur une/plusieurs cases (1), la fonction des éléments saillants est remplacée par la lisibilité du motif (2).

Comme le montraient déjà les taux de corrélation linéaire, le nombre d'instructions de la solution de référence (5) et le nombre d'instructions de la représentation du motif algorithmique en Scratch (6) impactent significativement et négativement la fréquence de réussite. Parmi ces deux variables quantitatives, c'est la taille du motif algorithmique qui a le plus d'effet : plus la représentation du motif algorithmique comporte d'instructions et plus la situation est difficile.

Parmi les variables qualitatives, celle dont les modalités sont les plus discriminantes est la disposition des motifs visuels : un motif sur une seule case est la caractéristique qui a l'effet le plus positif sur la fréquence de réussite, alors qu'un motif sur plusieurs cases avec disposition cyclique est celle qui a l'effet le plus négatif, au-delà du nombre d'instructions dans la boucle ou dans le programme entier. Pour les deux modalités intermédiaires, motif sur plusieurs cases avec disposition linéaire en 2D et motif sur plusieurs cases avec disposition 1D, nous constatons qu'elles ont respectivement le même impact que la nécessité ou pas de pivotement du robot. Ce résultat nous semble cohérent dans la mesure où la nécessité de pivotement du robot n'intervient que lorsque les motifs visuels sont disposés en 2D (linéairement ou de manière cyclique). Ce résultat montre l'importance de l'identification du motif visuel lors de la résolution d'une situation

de programmation d'un robot sur une grille.

Nous retrouvons aussi le résultat déjà ébauché à propos des éléments saillants, qui, selon leur fonction, impactent différemment la fréquence de réussite. Ce qu'apporte l'analyse des coefficients du modèle, c'est le positionnement de cet impact par rapport à celui des autres variables. Il est très important. La lisibilité du motif visuel a par exemple un poids plus lourd que la variable plus au cœur de l'algorithmique de la présence ou non d'instructions hors de la boucle.

11.3.5 Classes de situations

En nous appuyant sur les analyses précédentes, notamment sur l'analyse des coefficients du modèle de régression linéaire de la section précédente, nous établissons une gradation dans la difficulté des situations de programmation, en distinguant 4 classes de situations. Elles reprennent et combinent les caractéristiques qui impactent le plus la fréquence de réussite.

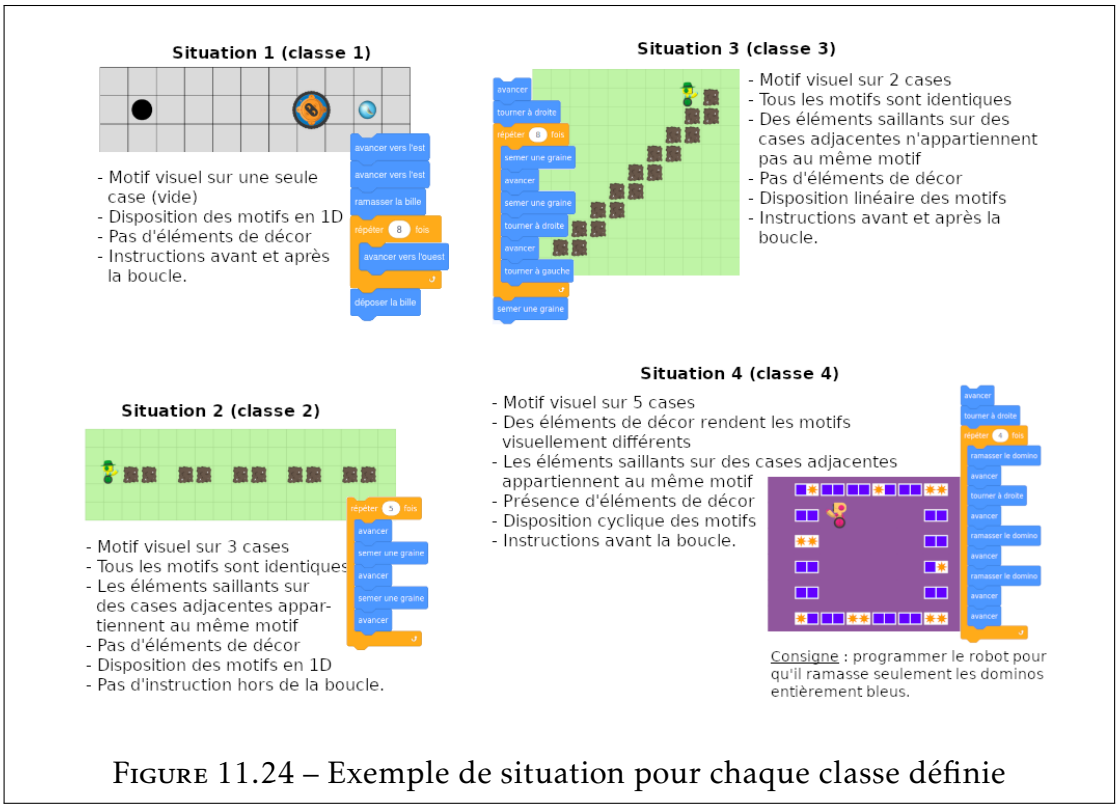


FIGURE 11.24 – Exemple de situation pour chaque classe définie

La figure 11.24 montre un exemple de situation pour chaque classe tandis que la figure 11.25 montre l'évolution de la moyenne, de la médiane et de l'écart interquartile selon les niveaux de classe.

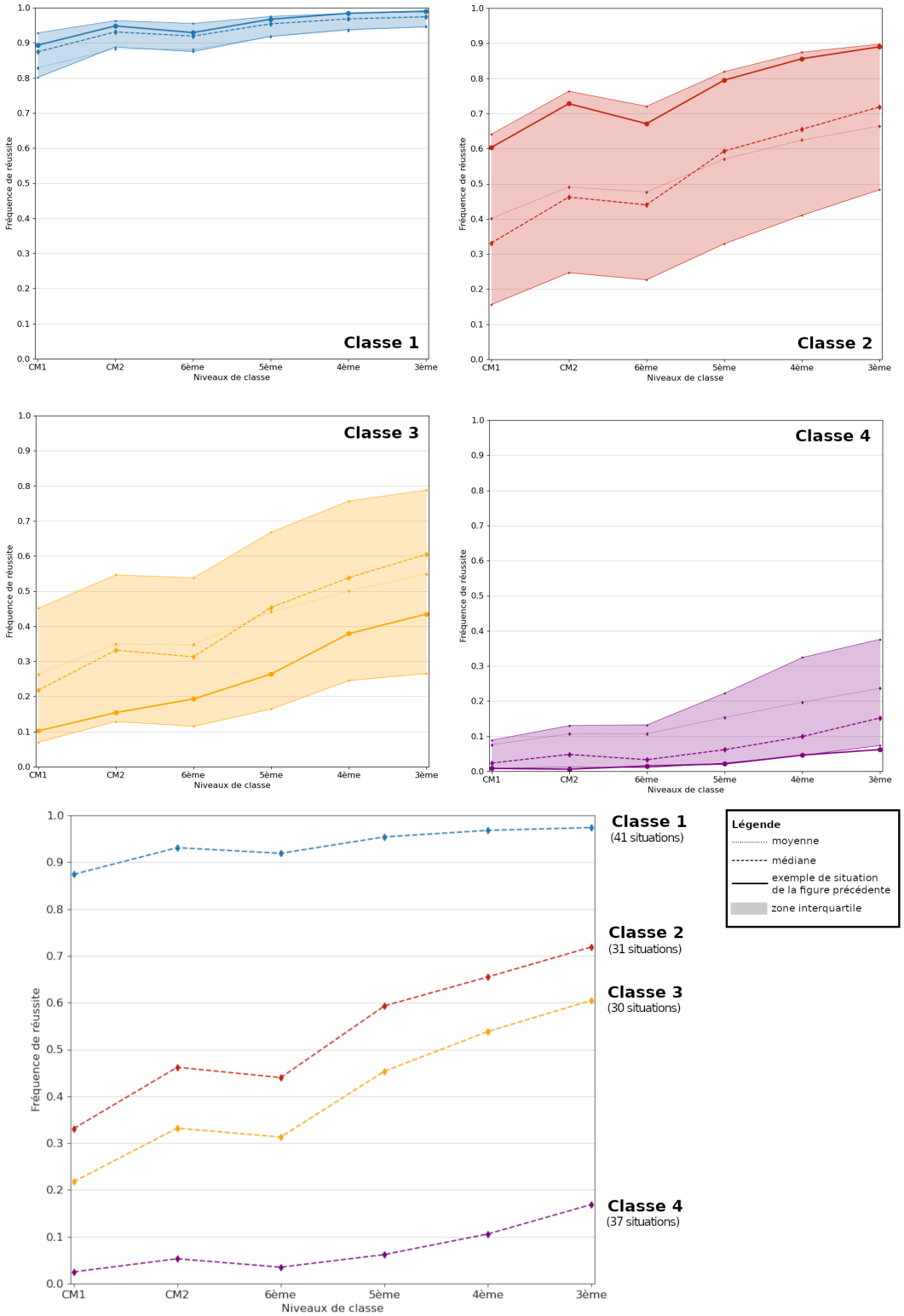


FIGURE 11.25 – Classes de situations déduites de la régression linéaire

Une première classe de situation, très distincte, concerne les situations où la case délimite le motif.

Les classes 2 et 3 correspondent à une combinaison entre disposition des motifs visuels sur la grille et système d'orientation du robot.

Nous regroupons dans la classe 2 les situations pour lesquelles nous avons une correspondance stricte entre motif visuel et motif algorithmique. Chaque action de déplacement est repérable par la limite entre deux cases et les autres actions sont identifiables par un élément saillant visuellement. Ce sont les situations où le déplacement du robot n'est possible que dans une seule direction, les situations de déplacement en orientation absolue (nord, sud, est, ouest) et les quelques situations en orientation hybride.

La classe 3 correspond aux situations où plusieurs états du robot virtuel sur une même case sont visuellement identiques, rendant partielle la correspondance entre motif visuel et motif algorithmique. Ce sont les situations en orientation relative, avec disposition linéaire des motifs en 2D. Pour ces situations, les actions de pivotement du robot ne sont pas observables avant l'exécution du programme, il est nécessaire de les simuler mentalement, en se les représentant sur les cases adéquates et en maintenant l'orientation du robot en mémoire.

Enfin, nous regroupons les situations pour lesquelles la lisibilité du motif visuel est perturbée dans la classe 4. Cette classe comprend d'une part les situations pour lesquelles il est nécessaire de faire abstraction de certains éléments visuels. Soit des éléments saillants ou des éléments de décor sont équivalents mais visuellement différents, soit plusieurs motifs visuels sont partiellement superposés, perturbant la lisibilité de chacun d'eux, soit des éléments identiques sur des cases adjacentes ne font pas partie du même motif. La classe 4 comprend aussi les situations en orientation relative pour lesquelles la disposition des motifs est cyclique. En effet, les motifs visuels ne sont identiques qu'à une rotation d'un quart de tour près, et non strictement, cette rotation ayant un impact très négatif sur la fréquence de réussite.

Un test de Tukey, appliqué après un test de Kruskal-Wallis, indique que les différences de fréquence de réussite entre les groupes pris deux à deux sont toutes très significatives (p-*value* $< 0{,}01$) à une exception près. La différence de fréquence de réussite entre les classes 2 et 3, avec une p-*value* de 0,12, n'atteint pas le seuil de significativité, ce qui se traduit visuellement par un recouvrement partiel des zones interquartiles.

Les situations de la classe 1, pour lesquelles la correspondance entre les deux motifs est attachée à la case, sont bien réussies par la majorité des élèves dès l'école élémentaire. En revanche, les situations de la classe 4, qui nécessitent beaucoup plus de capacités d'abstraction, sont encore difficiles pour la plupart des élèves de fin de collège, même si on constate un étirement de l'écart inter-

quartile vers le haut, indiquant qu'un certain nombre de situations sont mieux réussies. Les zones interquartiles des classes 1 et 4 ne chevauchent pas celle des autres classes de situations.

Une analyse plus approfondie est nécessaire afin de déterminer s'il y a bien rupture dans le traitement de la situation par le sujet, et donc si nous sommes bien en présence de classes de situations distinctes, en particulier pour les classes 2 et 3. Une autre hypothèse est que des facteurs annexes (comme la présence d'éléments de décor) joue un plus grand rôle pour ces classes intermédiaires. Cette difficulté de caractérisation constitue une limite de notre étude à cette échelle.

11.4 Synthèse

La partie épistémologique de ce chapitre a permis de définir les concepts de motif visuel et de motif algorithmique, et de construire un champ conceptuel autour du concept de motif algorithmique. Ce champ conceptuel comprend les notions de base de la programmation impérative (séquence d'instructions, boucle, condition, variable, fonction), que nous avons structurées hiérarchiquement. Dans la seconde partie, nous avons mobilisé les concepts de motif visuel et de motif algorithmique sur la fraction du champ conceptuel qui concerne la boucle bornée, pour analyser les situations de programmation d'un robot virtuel sur une grille en langage Scratch.

Dans la partie expérimentale du chapitre, nous avons montré qu'un problème de programmation de type itératif, même si sa complexité du point de vue algorithmique est proche (la solution de référence comporte seulement une ou plusieurs boucles bornées, sans imbrication), peut s'avérer plus ou moins difficile. Nos analyses mettent en évidence que lors de la résolution de ce type de problème dans le contexte de la programmation d'un robot virtuel sur une grille, l'identification d'un motif visuel et celle du motif algorithmique correspondant sont essentiels.

Nous avons caractérisé des facteurs qui rendent difficiles l'identification du motif visuel et nous avons établi une gradation dans les difficultés rencontrées. Nous interprétons chaque palier de difficulté comme une classe de situations au sens de Vergnaud. Les variables qui distinguent les différentes classes sont essentiellement la disposition et la lisibilité des motifs visuels.

Nous avons également mis en évidence que la fonction des éléments présents sur la grille est loin d'être neutre, que la difficulté des situations est corrélée au degré d'aide ou de perturbation pour la lisibilité du motif visuel. Nous déduisons de cela que les situations épurées sont à privilégier pour favoriser la réussite des sujets, ce qui va à l'encontre de choix qui consistent à surcharger la

grille d'éléments visuels, éventuellement animés, afin de rendre le design visuel attrayant pour l'utilisateur. Il est probable, au vu de ces résultats, que cette volonté d'accrocher l'utilisateur par un design attractif, se fasse au prix d'une lisibilité moindre de la grille, et rende la situation plus difficile à appréhender.

Sur le plan des enjeux pour les praticiens, au vu de nos résultats sur le rôle essentiel de l'identification du motif visuel lors de la résolution de puzzles de programmation d'un robot virtuel sur une grille, tâche très répandue lors de l'initiation à la programmation en langage Scratch, nous pensons que les préconisations du travail sur les motifs émises dans la note du CSEN (Ciccione & Dehaene, 2023) adressent aussi pleinement l'initiation à l'algorithmique.

Dans les situations que nous avons analysées, nous avons montré que l'identification du motif visuel est essentielle lors de la programmation d'une boucle bornée. Cette analyse reste cependant incomplète. Nous savons d'une part que l'identification de motif n'est pas seule en jeu dans le traitement de ces situations. Une fois le motif identifié, il est nécessaire d'en dénombrer les occurrences, ce qui peut induire d'autres difficultés qui sont aussi à analyser.

D'autre part, la présence de motifs visuels est contextuelle aux situations de programmation de robot sur une grille. Nous nous posons donc la question de la généralisation de ces résultats. Est-ce que la place du motif inhérent serait aussi centrale dans une autre modalité, ou pour d'autres situations pour lesquelles ne subsisteraient que le motif algorithmique ? Est-ce que réussir des puzzles de programmation d'un robot sur une grille conduit à acquérir des compétences transférables à d'autres situations de programmation ? En d'autres termes, quelle est la partie des compétences qui est généralisable ? Et par suite, comment se passe la transition vers d'autres situations de programmation qui requièrent l'utilisation d'une boucle ?

Concernant la portée de l'analyse, son point fort est l'échelle très large qui confère une robustesse statistique à la plupart des résultats établis. En revanche, la seule fréquence de réussite ne porte pas d'information relative au processus qui a conduit à la réussite. Lors du concours Algoréa, le nombre de tentatives pour chaque puzzle n'est pas limité. De ce fait, peut-on considérer qu'un sujet maîtrise la notion de boucle lorsqu'il a résolu un puzzle en parfois plus de 10 minutes et d'une dizaine d'essais ?

Pour affiner notre compréhension, et appréhender la conceptualisation-en-acte du sujet, nous avons besoin d'analyser des données plus fines. Les traces d'interaction dont nous disposons à l'échelle de classes et les enregistrements vidéo à l'échelle du sujet devraient nous permettre d'approfondir notre analyse, ce qui est l'objet des chapitres suivants.

Chapitre 12

Les premiers apprentissages de la boucle

Sommaire du présent chapitre

Notre objectif général est d'étudier la conceptualisation des notions algorithmiques de base par les élèves de 7 à 15 ans. Nous détaillons l'analyse pour la notion de boucle bornée, notamment en étudiant la conceptualisation progressive du concept de motif, notre objet de recherche.

À cette fin, nous analysons la confrontation de sujets avec des puzzles de programmation dans un environnement de programmation par blocs. Nous avons collecté des données sur les mêmes puzzles à trois échelles différentes et nous avons défini une méthodologie combinant analyses quantitatives de traces d'interaction et analyses qualitatives d'enregistrements vidéo d'écran.

Dans ce chapitre, nous mobilisons les données collectées à l'échelle des classes et à l'échelle du sujet. Les 23 sujets que nous avons suivis, pour certains pendant deux années, et pour lesquels nous disposons à la fois des traces d'interaction collectées sur la plateforme et des enregistrements vidéo de sessions, tiennent une place prépondérante dans les analyses de ce chapitre. Un extrait des analyses préalables à la rédaction de ce chapitre est disponible en annexe 9 (extraits vidéo) et en annexe 10 (traitements quantitatifs). Les observations réalisées à l'échelle du sujet éclairent et affinent les analyses quantitatives menées à l'échelle des classes, qui elles-mêmes apportent des éléments d'explication aux fréquences de réussite relevées à l'échelle nationale.

Grâce aux outils que nous avons construits et qui ont été présentés dans le chapitre 9 (indicateur de rapidité normalisée et profils induits, différentes visualisations de nos données), nous analysons les premières confrontations avec la notion de boucle, c'est à dire avec les situations de la classe 1 de la gradation établie dans le chapitre 11, pour lesquelles le motif visuel correspond à une case de la grille.

Du point de vue théorique, nous nous appuyons essentiellement sur ce que Vergnaud nomme la conceptualisation-en-acte, c'est-à-dire que nous faisons l'hypothèse que l'activité du sujet révèle une part de sa conceptualisation. Dans

notre contexte, l'activité du sujet dans l'environnement de programmation est instrumentée par la souris. Elle consiste en des déplacements de blocs par glisser-déposer, des clics sur les endroits précis, et des survols de zones de l'interface. Cette activité gestuelle est parfois accompagnée de verbalisations, que nous avons encouragées, et qui permettent dans certains cas de confirmer le sens des gestes du sujet. Nous cherchons à caractériser des schèmes, organisation invariante de l'activité du sujet, en en renseignant les composantes définies dans la définition analytique qu'en donne Vergnaud.

Nous avons adossé le protocole expérimental de notre recherche au concours Algoréa (chapitre 7). En conséquence, l'ordre de présentation des puzzles, qui est celui du concours en catégorie blanche, est imposé. Il nous conduit à structurer le présent chapitre de la manière suivante. Dans un premier temps, nous évaluons en début de session si les sujets mobilisent spontanément le bloc *répéter* en analysant leur activité sur le puzzle p1v3. Dans un deuxième temps, nous analysons le passage de la séquence à la boucle au cours de la session pour les sujets qui ne maîtrisent pas la boucle en début de session. À cette fin, nous investiguons l'activité de ces sujets sur le puzzle p3v2. Comment s'effectue ce passage ? La mobilisation de la boucle est-elle acquise lors des sessions suivantes ? Dans un troisième temps, nous étudions le passage d'une à deux instructions dans le corps de la boucle en analysant l'activité des sujets sur les puzzles p4v1 et p4v2. Nous complétons l'étude de ces deux paliers par une analyse des procédures expertes et des erreurs les plus fréquemment commises lors de la programmation d'une boucle pour les situations de la classe 1, avant de synthétiser nos résultats en les interprétant en termes de système hiérarchique de schèmes.

12.1 Évaluation de la mobilisation spontanée de la boucle en début de session

Les puzzles p1v3 des éditions 2021 et 2022 nous permettent d'évaluer si, lors de chaque début de session, le sujet mobilise spontanément le bloc *répéter* dans un cas simple. Contrairement aux puzzles de la suite du parcours, le bloc *répéter* est mis à disposition pour les puzzles p1v3, mais son utilisation n'est pas contrainte par le nombre de blocs disponibles. Le puzzle peut être résolu sans recourir à une boucle, en programmant une séquence d'instructions. La figure 12.1 montre un exemple de ce type de puzzle, ainsi que les deux solutions les plus fréquemment soumises, l'une comportant une boucle et l'autre non.

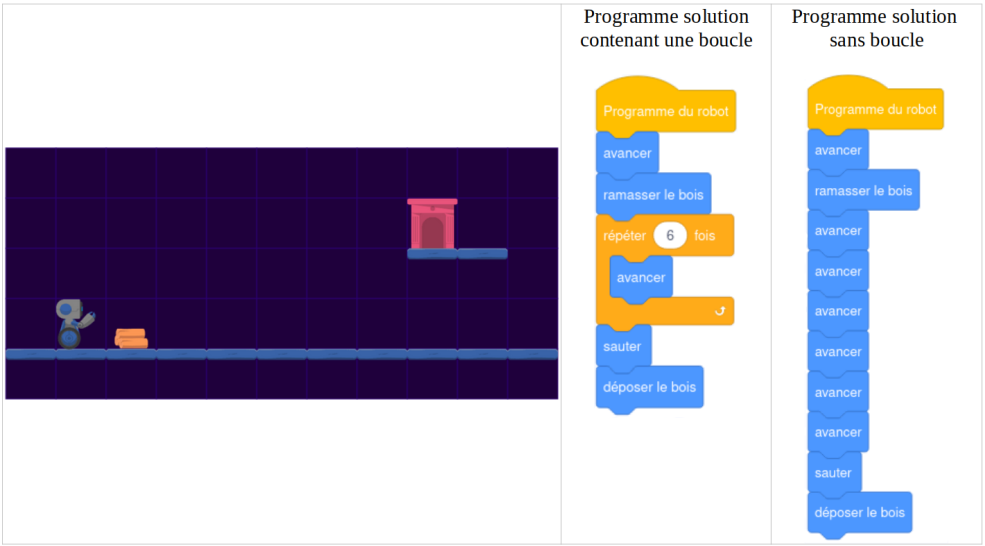


FIGURE 12.1 – Problème 1 version 3 étoiles du tour 1 de l'édition 2022 du concours Algoréa (2022t1p1v3)

12.1.1 Au collège

Niveau de classe	Type de résolution	2022t1p1v3		2022t2p1v3		2022t3p1v3	
		Effectif	Fréquence	Effectif	Fréquence	Effectif	Fréquence
6ème	boucle	44	0,94	41	0,95	21	1,00
	séquence	3	0,06	2	0,05	0	0
	échec	0	0	0	0	0	0
	non abordé	1		0		0	
5ème	boucle	61	0,87	53	0,88	66	1,00
	séquence	9	0,13	4	0,07	0	0
	échec	0	0	3	0,05	0	0
	non abordé	1		0		0	
4ème	boucle	23	0,96	18	0,95	20	1,00
	séquence	1	0,04	1	0,05	0	0
	échec	0	0	0		0	0
	non abordé	0		0		0	
3ème	boucle	43	0,92	53	0,93	16	1,00
	séquence	3	0,06	1	0,02	0	0
	échec	1	0,02	3	0,05	0	0
	non abordé	3		0		0	
Total participants		193		179		123	

TABLEAU 12.1 – Fréquence de résolution des puzzles p1v3 avec une boucle et avec une séquence pour le niveau collège en 2022

Nous relevons le nombre de sujets qui valident les puzzles p1v3 respectivement avec une boucle et avec une séquence dans les données de 2022 à l'échelle des classes (Tableau 12.1). Dans ce tableau, les sujets sont répartis selon leur niveau de classe.

Sur les 193 collégiens de notre échantillon qui ont participé au premier tour de l'édition 2022 en catégorie blanche, de l'ordre de 90% conçoivent spontanément un programme avec une boucle en début de session. Lors du troisième tour, c'est la totalité des sujets de notre échantillon qui utilise spontanément la boucle dans un cas simple. Nous n'observons pas de différence significative en fonction du niveau de classe.

Pour les sujets qui valident le puzzle avec une boucle, nous affinons notre analyse en considérant la proportion de résolutions expertes en utilisant le calcul de profils présenté dans le chapitre 9.

Nous avons respectivement, selon les tours, 53%, 27% et 30% des sujets qui résolvent le problème avec un profil expert si nous prenons en compte les seuls résultats en catégorie blanche. Si nous incluons aussi comme experts les élèves qui sont passés dans la catégorie supérieure pour les tours 2 et 3, les proportions s'élèvent à 53%, 44% et 55%. Comme il n'y a aucune raison que le taux chute d'environ 10 points pour le tour 2, nous en déduisons que probablement une erreur d'une autre nature intervient, ce qui est effectivement constaté lors de l'étude instrumentale (10.4.2 : erreur de représentation de la règle régissant le micromonde).

Nous pouvons donc affirmer que pour une grande majorité des sujets scolarisés au collège, une première maîtrise de la boucle est déjà acquise lorsqu'ils abordent le parcours du concours Algoréa, quelque soit leur niveau de classe.

12.1.2 À l'école élémentaire

Les données du premier tour de 2021 dont nous disposons pour des élèves plus jeunes (69 élèves de CE, 49 élèves de CM) nous permettent d'étendre notre analyse au niveau de la scolarité élémentaire. Nous joignons les quelques sessions de 6ème que nous avons pour le même tour. Bien que le nombre en soit très limité (9 sessions), ces sessions permettent d'entrevoir que nous retrouvons une présence massive de la validation avec une boucle en début de collège (Tableau 12.2).

Les résultats consignés dans le tableau 12.2 montrent que 57% des sujets scolarisés en CE de notre étude qui ont abordé le puzzle p1v3, soit plus de la moitié, utilisent déjà spontanément une boucle dans un cas simple. La proportion monte à 63% en CM.

Si nous considérons le profil de la résolution, respectivement 11% (4 sujets sur 38) et 32% (10 sujets sur 31) des sujets de CE et de CM qui résolvent le

Niveau de classe	Type de résolution	2021t1p1v3	
		Effectif	Fréquence
CE	boucle	38	0,57
	séquence	21	0,32
	échec	7	0,11
	non abordé	3	
CM	boucle	31	0,63
	séquence	15	0,31
	échec	3	0,06
	non abordé	0	
6ème	boucle	9	1,00
	séquence	0	0
	échec	0	0
	non abordé	0	

Tableau 12.2 – Fréquence de résolution des problèmes 1 version 3 avec une boucle et avec une séquence pour le niveau élémentaire en 2021

puzzle avec une boucle ont un profil expert.

Nous en déduisons que la maîtrise de la boucle dans un cas simple est en phase de construction et progresse rapidement au cours de la scolarité à l'école élémentaire. Ce constat nous amène à affirmer que la notion de boucle est largement abordable dès ce niveau de la scolarité.

Nous formulons deux hypothèses quant à cette compétence déjà-là. Soit le bloc *répéter* a été abordé lors d'une séquence pédagogique précédente (entraînement préalable au concours Algoréa, projet dans l'environnement Scratch ou Scratch Jr...), soit la prise en main du bloc *répéter* est assez aisée pour être réalisée en autonomie lors de la session.

Dans l'optique d'investiguer la seconde hypothèse, nous nous demandons dans la section suivante comment évoluent les élèves qui n'abordent pas le parcours en utilisant spontanément une boucle.

12.2 Le passage de la séquence à la boucle simple

Dans cette section nous étudions le parcours des élèves qui valident l'un des puzzles p1v3 avec une séquence, ou qui sont en échec sur l'un de ces puzzles. Nous analysons notamment leurs résultats sur le puzzle p3v2, qui est le premier puzzle du parcours où la boucle est contrainte sans que de l'aide ne soit disponible (ce qui est le cas pour le puzzle p3v1 qui le précède immédiatement).

12.2.1 Analyse quantitative

Nous débutons par une analyse quantitative de la structure de la solution soumise (séquence vs boucle) à laquelle nous ajoutons le profil de confrontation sujet/puzzle calculé (9.2.5).

Sujets scolarisés au collège en 2022

Dans les tableaux 12.3 et 12.4 , le profil de résolution des sujets qui ont validé le puzzle p1v3 avec une séquence est sur fond bleu, celui des sujets qui ont validé ce puzzle en utilisant une boucle est sur fond vert.

Identifiant du sujet / Classe	2022 tour 1		2022 tour 2		2022 tour 3	
	p1v3 Chauffer le château	p3v2 Planter des fleurs	p1v3 Ranger les billes	p3v2 Ranger les billes	p1v3 Dans l'espace	p3v2 Pousser les caisses
145 : 6ème	expert	expert	catégorie jaune		catégorie jaune	
18 : 6ème	expert	expert	expert	expert	ajustement	ajustement
40 : 6ème	accommodation	ajustement	accommodation	expert	accommodation	accommodation
45 : 5ème	expert	expert	ajustement	accommodation	expert	expert
139 : 5ème	expert	expert			accommodation	expert
108 : 5ème	expert	ajustement	échec	ajustement	accommodation	accommodation
192 : 5ème	ajustement	expert	catégorie jaune			
104 : 5ème	ajustement	expert	accommodation	expert	expert	expert
225 : 5ème	ajustement	ajustement	catégorie jaune		catégorie jaune	
64 : 5ème	accommodation	expert	accommodation	ajustement	accommodation	ajustement
105 : 5ème	accommodation	ajustement	accommodation	accommodation	accommodation	expert
126 : 5ème	accommodation	ajustement	accommodation	expert	expert	expert
100 : 4ème	expert	ajustement	accommodation	expert	accommodation	ajustement
52 : 3ème	expert	ajustement	accommodation	expert		
206 : 3ème	ajustement	expert	catégorie jaune			
116 : 3ème	ajustement	ajustement	accommodation	ajustement		

Tableau 12.3 – Parcours des sujets qui ont validé le puzzle 2022t1p1v3 avec une séquence

Dans un premier temps, nous analysons le parcours des 16 sujets qui ont validé le puzzle p1v3 avec une séquence lors du premier tour de 2022 (Tableau 12.3). Ces sujets valident tous le puzzle p3v2 dans la suite de la session. Cela signifie soit que ces sujets savaient utiliser la boucle mais qu'ils ne mobilisent pas cette compétence spontanément, soit qu'une première maîtrise du bloc *répéter* est construite au cours de la session. Pour 8 d'entre eux, soit la moitié, la résolution est même experte pour le puzzle p3v2.

Le puzzle p1v3 du tour 2 de 2022 est équivalent à celui du tour 1 du point de vue de l'utilisation d'une boucle et il est placé au même endroit dans le parcours. Si nous regardons les résultats de ces 16 sujets sur ce puzzle, nous constatons que :

- 8 utilisent spontanément une boucle

- 2 seulement utilisent à nouveau une séquence
- 5 n'ont pas participé au deuxième tour en catégorie blanche mais 4 d'entre eux ont participé en catégorie jaune, le changement de catégorie impliquant qu'ils ont acquis une maîtrise experte de la boucle.
- 1 est en échec sur le puzzle p1v3 mais réussit le puzzle p3v2 avec un profil ajustement

Nous en déduisons que pour 12 élèves sur les 15 qui ont participé au deuxième tour, soit 80%, le recours à la boucle dans un cas simple est devenu spontané. L'apprentissage est ancré au moins si l'on reste dans le même environnement de programmation.

Identifiant	2022 tour 1		2022 tour 2		2022 tour 3	
du sujet / Classe	p1v3 Chauffer le château	p3v2 Planter des fleurs	p1v3 Ranger les billes	p3v2 Ranger les billes	p1v3 Dans l'espace	p3v2 Pousser les caisses
40 : 6ème	accommodation	ajustement	accommodation	expert	accommodation	accommodation
186 : 6ème			ajustement	ajustement	ajustement	accommodation
48 : 5ème	expert	expert	accommodation	expert	accommodation	
64 : 5ème	accommodation	expert	accommodation	ajustement	accommodation	ajustement
214 : 5ème			ajustement	indéfini	ajustement	ajustement
239 : 5ème			accommodation	ajustement	accommodation	ajustement
41 : 4ème	accommodation	accommodation	ajustement	expert	ajustement	ajustement
30 : 3ème		expert	accommodation	ajustement		
101 : 5ème	accommodation	accommodation	échec	ajustement		
222 : 5ème			passé	accommodation	accommodation	accommodation
248 : 3ème	accommodation	expert	échec	ajustement		
89 : 3ème		accommodation	échec	accommodation		
188 : 3ème			échec	accommodation	accommodation	ajustement

Tableau 12.4 – Parcours des sujets qui ont validé le puzzle 2022t2p1v3 avec une séquence ou qui sont en échec sur ce puzzle

Nous considérons désormais les 8 sujets qui ont validé le puzzle p1v3 avec une séquence lors du deuxième tour (Tableau 12.4) :

- 4, c'est à dire la moitié d'entre eux, n'ont pas participé au premier tour.
- 2 avaient déjà validé avec une séquence lors du premier tour. Ces deux sujets n'utilisent la boucle que lorsqu'ils y sont contraints (ils valident à nouveau le puzzle p3v2, l'un de manière experte).
- 2 avaient validé le puzzle avec une boucle au premier tour, l'un de manière experte. Ces deux sujets maîtrisent la boucle, mais ne l'ont pas mobilisé sur ce puzzle particulier.

De ces analyses concernant des sujets scolarisés au collège, nous retenons que la prise en main du bloc *répéter* afin de coder une boucle dans un cas simple est réalisé au cours de la première confrontation avec l'interface du concours Algoréa, et aboutit dans les sessions suivantes à une utilisation spontanée de la boucle dans un cas simple, même lorsque cette utilisation n'est pas contrainte.

Sujets scolarisés à l'école élémentaire en 2021

Pour les sujets scolarisés à l'école élémentaire, dont nous ne disposons de données que pour un seul tour, nous mettons en regard le profil de confrontation sujet/puzzle sur les puzzles p1v3 (introduction de la boucle sans qu'elle soit contrainte) et p3v2 (utilisation de la boucle requise) (Tableaux 12.5 pour le niveau CM et 12.6 pour le niveau CE). La partie des tableaux sur fond vert correspond à la réussite sur le puzzle p3v2, donc à une première maîtrise de la boucle au cours de la session.

Profil p3v2 Profil p1v3	expert	ajustement	accommodat.	indéfini	échec	non abordé	Total
séquence	5	6	2			2	15
échec				1	1	1	3

Tableau 12.5 – Profil de confrontation avec le puzzle p3v2 des sujets de CM qui n'ont pas validé le puzzle p1v3 avec une boucle

Profil p3v2 Profil p1v3	expert	ajustement	accommodat.	indéfini	échec	non abordé	Total
séquence	8	7	1		2	3	21
échec			1			6	7

Tableau 12.6 – Profil de confrontation avec le puzzle p3v2 des sujets de CE qui n'ont pas validé le puzzle p1v3 avec une boucle

87% (13 sur 15) des sujets scolarisés en CM qui ont validé le puzzle p1v3 avec une séquence réussissent le puzzle p3v2 qui requiert l'utilisation d'une boucle, dont un tiers (5 sur 15) avec un profil expert (Tableau 12.5). En CE, ce sont 76% (16 sur 21) des sujets ayant validé le puzzle p1v3 avec une séquence qui réussissent le puzzle p3v2, dont à nouveau plus d'un tiers avec un profil expert (Tableau 12.6). Le passage de la séquence à la boucle est donc également réalisé pour une majorité de sujets de l'école élémentaire.

Parmi les sujets scolarisés en CE en échec sur le puzzle p1v3, un seul passe à la boucle au cours de la session. Tous les autres se sont arrêtés dans le parcours avant d'atteindre le puzzle p3v2. Probablement que ces sujets n'ont pas encore acquis des notions plus basiques attachées à la séquence d'instructions.

12.2.2 Analyse qualitative de quelques parcours

Pour compléter ces investigations et appréhender le passage de la séquence à la boucle plus finement, nous consultons des enregistrements vidéo de session. Neuf sujets scolarisés au niveau de l'école élémentaire ont participé au tour 1 de 2021 à l'échelle du sujet. Parmi eux, six sujets (deux de CE et quatre de CM), dont

c'était la première confrontation avec l'interface Algoréa, ont validé le puzzle p1v3 avec une séquence.

Nous détaillons le parcours de trois d'entre eux, à titre d'exemples représentatifs.

Alexis (CM2) : compétence déjà présente mais non mobilisée

Lors du tour 1 de 2021, c'est la première confrontation d'Alexis avec l'interface de programmation Algoréa, mais il a déjà programmé un peu dans l'environnement Scratch. Lors de ce tour, Alexis valide le puzzle p1v3 avec une séquence (Alexis, CM2, 2021t1p1v3). Alexis valide ensuite les puzzles p3v1 (Alexis, CM2, 2021t1p3v1) et p3v2 très rapidement, sans hésitation, ce qui signifie qu'il maîtrise l'utilisation du bloc *répéter* de manière experte. Nous retrouvons cette expertise lors de toutes les sessions suivantes, pendant les deux années qu'a duré l'expérimentation.

Il est très probable qu'Alexis savait déjà utiliser le bloc *répéter* en amont de l'expérimentation, mais qu'il n'a pas mobilisé cette compétence lors de la découverte de ce nouvel environnement.

Ali (CM1) : compétence acquise pendant la session grâce à un étayage de l'expérimentatrice

Ali découvre l'interface Algoréa et la programmation par blocs à l'occasion de l'expérimentation. Lors du tour 1 de 2021, il valide le puzzle p1v3 avec une séquence d'instructions (Ali, CM1, 2021t1p1v3).

Lorsqu'Ali aborde le puzzle p3v1 lors de cette même session, la prise en main du bloc *répéter* est laborieuse. La résolution dure plus de 7 minutes et nécessite une intervention conséquente de l'expérimentatrice (Ali, CM1, 2021t1p3v1). Ali est en difficulté pour comprendre que le nombre 7 placé dans le champ du bloc *répéter* synthétise l'écriture des sept occurrences du bloc *avancer* malgré plusieurs consultations du tutoriel. Une fois cet obstacle surmonté, Ali utilise correctement le bloc *répéter* pour résoudre le puzzle p3v2 (Ali, CM1, 2021t1p3v2).

Lors du tour suivant deux mois plus tard, Ali réagit verbalement à la mise à disposition du bloc *répéter* : « Ah je m'en souviens de celui-là » et le place immédiatement dans l'éditeur pour l'utiliser alors qu'il n'est pas requis en début de programme (Ali, CM1, 2021t2p1v3). La première ébauche de programme, supprimée par la suite, fait basculer le type de profil de la résolution d'expert vers indéfini, mais le visionnage de l'enregistrement vidéo montre qu'après ce temps nécessaire pour réactiver la compétence, Ali maîtrise l'utilisation du bloc

répéter. Dans la suite de la session, les puzzles p3v1 et p3v2 sont validés avec un profil de résolution expert.

Tom (CE1) : compétence acquise pendant la session grâce à un étayage de l'expérimentatrice

Lors du tour 1 de 2021, Tom, dont c'est la première confrontation avec un environnement de programmation par blocs, valide le puzzle p1v3 avec une séquence d'instructions. La succession de programmes exécutés (Figure 12.2) montre que Tom a besoin de contrôler l'exécution très régulièrement, parfois après l'ajout d'un seul bloc au programme. Cependant, l'ensemble des blocs *avancer* consécutifs est édité en une seule fois. Bien que nous n'observons pas de dénombrement explicite sur la vidéo, nous faisons l'hypothèse que Tom regroupe déjà ces actions redondantes dans son raisonnement.

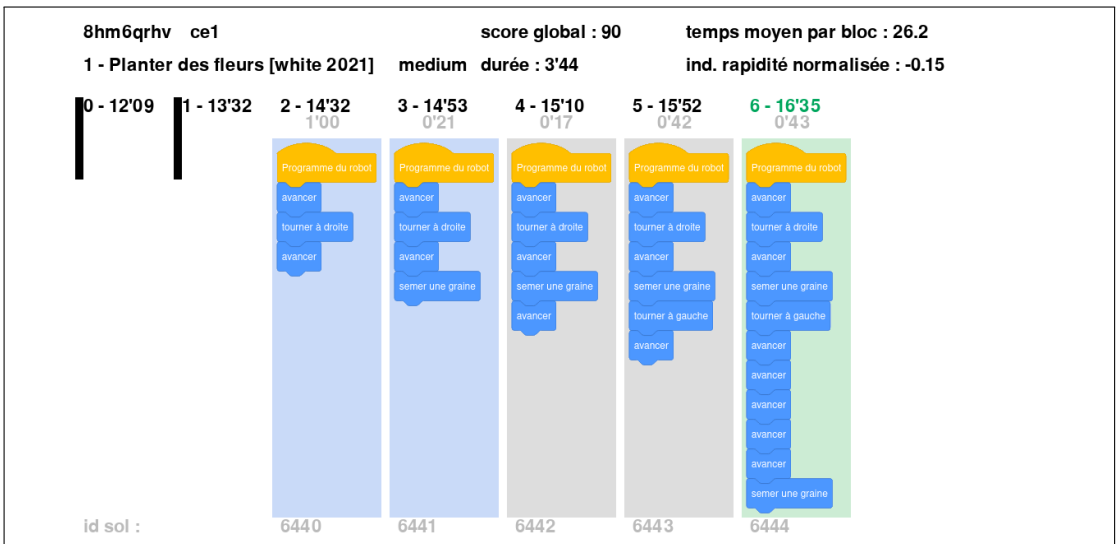


FIGURE 12.2 – Succession des programmes exécutés par Tom pour le puzzle 2021t1p1v3

Lors de la même session, plus de 7 minutes et un étayage massif de la part de l'expérimentatrice sont nécessaires pour que Tom réussisse le puzzle p3v1 (Tom, CE1, 2021t1p3v1). Tom édite spontanément une séquence de blocs *avancer vers l'est*, et n'a plus assez de blocs pour terminer son programme. Le tutoriel disponible ne faisant pas sens pour Tom, l'expérimentatrice prend le relai pour lui expliquer comment fonctionne le bloc *répéter*.

Les frises des sessions montrent que l'aide apportée est efficace (Figure 12.3).

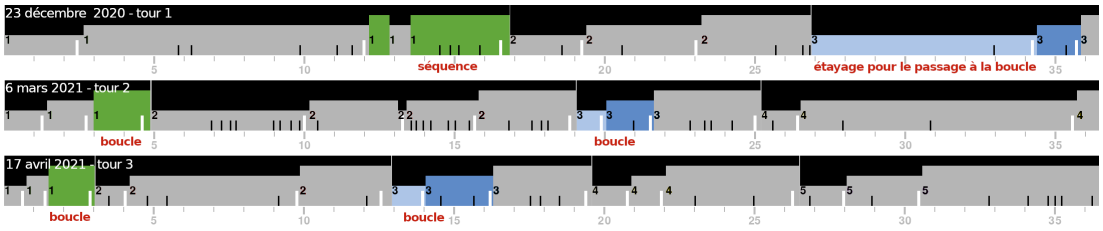


FIGURE 12.3 – Frises des sessions de Tom (CE1) en 2021 : passage de la séquence à la boucle

En effet, le puzzle suivant (p3v2) est validé beaucoup plus rapidement par Tom, qui utilise cette fois le bloc *répéter* en première intention (Tom, CE1, 2021t1p3v2).

Environ deux mois plus tard, lors du tour 2, Tom utilise le bloc *répéter* spontanément et de manière experte dès le début de la session (Tom, CE1, 2021t2p1v3). Il remarque d'ailleurs la mise à disponibilité de ce bloc pour le puzzle p1v3, premier puzzle du parcours pour lequel il est disponible : « Ah maintenant y'a le truc comme ça ». La remarque de Tom révèle une conceptualisation-en-acte qui n'est pas encore une connaissance, dans la mesure où Tom sait utiliser le bloc *répéter* sans savoir le nommer.

Le puzzle p3v1 du tour 2 est lui aussi résolu de manière experte. Pour le puzzle p3v2, l'exécution non valide correspond à un programme partiel et ne remet pas en question le caractère expert de l'utilisation du bloc *répéter*. Ainsi, pour Tom, l'acquisition réalisée lors du tour 1 est stable dans le temps. Nous la retrouvons aussi lors du tour suivant.

12.2.3 Difficultés rencontrées lors du passage de la séquence à la boucle

Nous complétons les analyses précédentes de parcours par des extraits d'autres sessions qui montrent les difficultés auxquelles sont confrontés les sujets lorsqu'ils utilisent le bloc *répéter* pour la première fois. Ces extraits sont issus du tour 1 des éditions 2021 et 2022.

- Le sujet essaie de placer des blocs dans le bloc *répéter* (Paola, CE2, 2022t1p3v1) ou de placer le bloc *répéter* autour du bloc *avancer* (Gina, CE1, 2022t1p3v1) alors que les blocs sont dans la réserve. Ces sujets, dont c'est la première confrontation avec l'interface de programmation, ne différencient pas bien les deux zones de l'interface, la zone où sont mis à disposition les blocs comme éléments du langage et la zone d'édition où ces blocs peuvent être manipulés.

- Le sujet utilise le bloc *répéter* mais l'instruction que le sujet veut répéter est placée à l'extérieur de celui-ci (Gina, CE1, 2022t1p3v1). Cette erreur admet deux interprétations possibles, non exclusives : un théorème-en-acte erroné selon lequel on ne peut pas placer un bloc dans un autre (idée fausse relevée par Vaníček et al. (2023)), ou/et l'ambiguïté avec le langage courant relevée par Rouchier (1990) selon laquelle la première action est détachée des répétitions suivantes.
- Le sujet n'a pas perçu qu'il peut changer le nombre d'itérations dans le champ du bloc *répéter*, il utilise ce bloc avec la valeur par défaut (Gina, CE1, 2022t1p3v1). Ce théorème-en-acte erroné selon lequel le nombre dans le bloc *répéter* ne peut pas être changé est également relevé en tant qu'idée fausse par Vaníček et al. (2023). Plus tard dans la même session, la représentation de Gina évolue, mais elle applique un théorème-en-acte, de nouveau erroné : « On ne peut placer que des nombres jusqu'à 10 dans le compteur du bloc *répéter* », ce qui l'amène à une solution non optimale en décomposant 12 itérations en 1,1, puis 10 (Gina, CE1, 2022t1p3v3).

Ces difficultés concernent la prise en main du bloc *répéter*. Ce sont des difficultés d'ordre instrumental, qui ont trait à la construction du bloc *répéter* comme instrument pour exprimer une répétition de manière synthétique. Une fois le schème d'usage du bloc *répéter* construit, ces difficultés disparaissent et la résolution des puzzles qui requièrent ce bloc dans un cas simple ne posent plus aucun souci.

En revanche, une fois le bloc *répéter* pris en main, celui-ci est parfois utilisé systématiquement, y compris de manière abusive (Inès, CE1, 2022t1p4v1). Le sujet emploie le bloc *répéter* alors que celui-ci n'est pas utile dans le cas d'un seul déplacement. Le sujet n'a pas encore bien délimité la classe de situations pour laquelle le recours à une boucle est pertinent.

12.2.4 Synthèse

Au regard de ces résultats, nous pouvons affirmer que pour une part significative des sujets qui arrivent sur le parcours Algoréa sans utiliser spontanément la boucle, le passage de la séquence à la boucle s'effectue lors de la session. C'est le cas pour 100% des élèves de collège, 87% des élèves de CM et 76% des élèves de CE.

Concernant les élèves de collège pour lesquels nous disposons des résultats sur trois tours d'une même année, nous pouvons affirmer de plus que cette acquisition se fait très majoritairement lors de la première confrontation avec un parcours Algoréa et qu'elle est ensuite stable dans le temps. Nos données ne

nous permettent pas de l'affirmer à cette échelle pour les sujets scolarisés en CE et CM et il serait intéressant de mener une étude complémentaire sur ce point.

Ce que montre en revanche l'évolution des quelques sujets scolarisés en CE et CM que nous avons suivi sur deux années, c'est que les difficultés rencontrées sont relatives à la construction du schème d'usage du bloc *répéter* pour exprimer la répétition (encadré ci-dessous).

Schème d'usage du bloc *répéter*

- **but** : utiliser le bloc *répéter*
- **règles de conduite de l'action** :
  - **prise d'information** : dénombrer en se référant à la grille
  - **action proprement dite** : placer le bloc *répéter* dans l'éditeur, cliquer sur le champ numérique du bloc *répéter* pour l'ouvrir, cliquer sur le clavier virtuel le résultat du dénombrement, cliquer sur la touche verte pour valider et fermer le clavier virtuel, placer ce qu'il faut répéter à l'intérieur du bloc *répéter*.
  - **contrôle** : parfois dénombrer à nouveau les cases de la grille, exécuter le programme édité
- **invariants opératoires**
  - **concepts-en-acte** : nombre dans son aspect cardinal
  - **théorèmes-en-acte** :
    - « ce qui est placé dans le bloc *répéter* est exécuté autant de fois que renseigné ; le nombre portant l'information, on ne met qu'une seule occurrence de ce qui est à répéter à l'intérieur du bloc »,
    - « le terme *répéter* sous-tend l'ensemble des répétitions, y compris la première »,
    - « on peut remplacer le nombre 10 indiqué par défaut, par un nombre qu'on choisit »
- **représentations**
  - représentation symbolique d'une collection par une écriture chiffrée

Le tutoriel accessible par un bouton ne suffit pas pour aider certains jeunes sujets à construire ce bloc *répéter* comme instrument. Un accompagnement humain s'avère nécessaire. À la suite de cet étayage initial, tous les sujets que nous avons suivis sont alors capables de programmer une boucle en autonomie

et de réussir les puzzles suivants. De plus, cette acquisition est stable dans le temps.

La spécificité du bloc *répéter* pour exprimer le résultat d'un dénombrement est que ce qui est compté doit être explicitement renseigné, en insérant des blocs dans le bloc *répéter*. Dans ce contexte précis, les éléments dénombrés sont identiques, ce qui n'est pas le cas pour un dénombrement quelconque. Mais il n'est pas exclu que ce qui est compté et renseigné ne soient pas de même nature, c'est pourquoi nous avons utilisé l'expression *ce qui est compté*. Par exemple, le sujet compte des cases de la grille et place un bloc d'action dans le bloc *répéter*. Ce décalage est source d'erreurs, sur lesquelles nous reviendrons (12.5.1).

12.3 Passage d'une à plusieurs instructions dans le corps de la boucle

Lorsque la case de la grille délimite le motif visuel, le motif algorithmique est de longueur 1 ou 2 dans les situations de notre protocole expérimental. Dans cette section, nous étudions ce passage de une à deux instructions dans le corps de la boucle. Nous examinons à cette fin les confrontations avec les puzzles p4v1 qui ont été validées avec une séquence (exemple sur la figure 12.4).

12.3.1 Analyse quantitative

Sujets scolarisés au collège en 2022

Lors de l'édition 2022 du concours Algoréa, la programmation d'une boucle n'est pas contrainte pour les puzzles p4v1, et quelques sujets scolarisés au collège ont validé ce puzzle avec une séquence d'instructions.

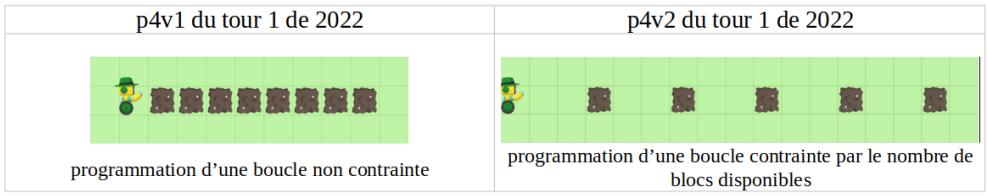


FIGURE 12.4 – Exemples de puzzle de type p4v1 et p4v2 de l'édition 2022 du concours Algoréa

Quel est le parcours antérieur dans la session pour ces sujets ? Ont-ils validé le puzzle p3v2 que nous prenons comme repère pour la maîtrise de la

boucle simple ? Valident-ils le puzzle p4v2 qui est le suivant dans l'ordre du parcours (exemple figure 12.4) et pour lequel l'utilisation d'une boucle est contrainte par le nombre de blocs disponibles ?

Lors du tour 1 de 2022, 9 sujets sur 189 (4,8%) valident le puzzle p4v1 avec une séquence. Au tour 2, il n'y a plus aucune validation de ce type de puzzle avec une séquence. Il y en a une seule au tour 3, sur 123 sujets qui ont abordé le puzzle.

Nous étudions de manière plus précise le parcours des 9 sujets qui ont validé le puzzle p4v1 du tour 1 avec une séquence (Tableau 12.7).

Identifiant du sujet / Classe	2022 tour 1			2022 tour 2			2022 tour 3		
	p3v2 Planter des fleurs	p4v1 Planter des fleurs	p4v2 Planter des fleurs	p3v2 Ranger les billes	p4v1 Chauffer le château	p4v2 Chauffer le château	p3v2 Pousser les caisseS	p4v1 Peindre le dessin	p4v2 Peindre le dessin
35 : 6ème	expert	indéfini	accomm.	ajust.	accomm.	expert	ajust.	ajust.	échec
21 : 6ème	ajust.	accomm.	échec	expert	accomm.	accomm.			
31 : 6ème	accomm.	accomm.	expert	ajust.	accomm.	accomm.	accomm.	ajust.	échec
40 : 6ème	ajust.	accomm.	échec	expert	accomm.	échec	accomm.	expert	accomm.
126 : 5ème	ajust.	indéfini	ajust.	expert	accomm.	expert	expert	expert	accomm.
133 : 5ème	expert	indéfini	indéfini	expert	ajust.	expert	expert	expert	expert
91 : 4ème	ajust.	accomm.	expert	expert	accomm.	ajust.	expert	ajust.	passé
125 : 4ème	accomm.	accomm.	échec	expert	ajust.	accomm.	ajust.	expert	accomm.
260 : 3ème	accomm.	accomm.	accomm.						

Tableau 12.7 – Parcours des sujets qui ont validé le puzzle 2022t1p4v1 avec une séquence

Parmi ces sujets, tous ont une première maîtrise de la boucle simple. En effet, ils ont validé le puzzle p3v2 lors de la même session.

Sept d'entre eux valident le puzzle suivant, p4v2, qui cette fois requiert une boucle. Pour ces sujets, la situation nouvelle a provoqué une régression passagère vers la séquence lors de la première confrontation, mais ensuite, le passage à la boucle avec plusieurs instructions est réalisé au cours de la session, et la maîtrise reste ensuite acquise.

Pour les trois sujets qui ont un profil échec lors du tour 3 alors qu'ils avaient déjà résolu le même type de puzzle lors du tour 1, nous avons regardé plus finement la session. Il s'avère que pour les trois, ils ne se sont pas attardés sur ce puzzle, et ont validé soit le puzzle suivant (p4v3) soit une version du problème p5 qui requiert de maîtriser la notion de boucle avec plusieurs instructions dans le corps de la boucle.

Trois sujets sur les neuf qui valident le puzzle p4v1 avec une séquence, sont en échec pour passer ensuite à la boucle à deux instructions. Cependant, ils valident le puzzle p4v1 avec une boucle lors des tours suivants. L'un d'entre eux est en échec pour le puzzle p4v2 lors du tour 2, révélant une acquisition fragile.

Nous en concluons qu'au niveau du collège, le passage à la boucle avec plusieurs instructions est spontané, car réalisé dans une situation où le recours à la boucle n'est pas contraint, pour la très grande majorité des sujets. Concernant les quelques sujets pour lesquels ce n'est pas le cas lors de la première confrontation, cette acquisition est réalisée au cours de l'année scolaire. De plus, lorsque la maîtrise est observée sur un puzzle, elle est pérenne, au moins à l'échelle de plusieurs mois, dans le même environnement de programmation.

Sujets scolarisés à l'école élémentaire en 2021

Nous complétons en étudiant les confrontations du premier tour de 2021 pour des sujets scolarisés à l'école élémentaire et quelques sujets scolarisés en 6ème. Les problèmes p4 de 2021 ont une particularité intéressante dans cette optique : les situations des versions 1 et 2 sont exactement les mêmes (Figure 12.5). Seuls changent le nombre de motifs visuels (et donc le nombre d'itérations) et le nombre de blocs disponibles. Ce nombre de blocs permet une validation avec une séquence pour la version 1 et contraint la boucle en version 2. Nous remarquons aussi que ces puzzles de 2021 sont directement comparables avec 2022t1p4v1.

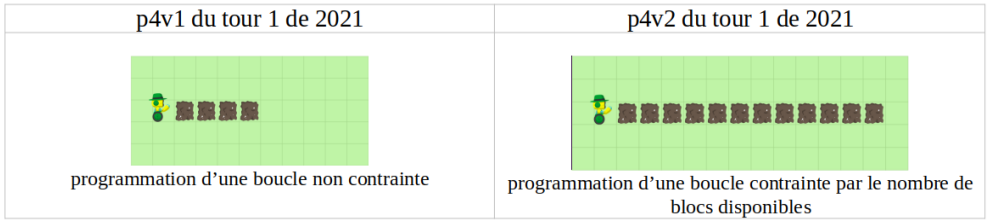


FIGURE 12.5 – Exemples de puzzle de type p4v1 et p4v2 de l'édition 2021 du concours Algoréa

En CE, seuls 61% des sujets (42 sur 69) abordent le puzzle p4v1 dans le parcours. Dans les 45 minutes que dure la session, les 39% de sujets qui n'atteignent pas ce puzzle sont très probablement confrontés à des difficultés en amont qui ont occupé le temps alloué. La proportion de sujets qui abordent le puzzle p4v1 atteint 90% des sujets en CM et 100% en 6ème, signe que les difficultés en amont ont majoritairement été résolues pour ces niveaux de classe.

Le tableau 12.8 montre le type de programme soumis pour les sujets qui ont abordé le puzzle p4v1 lors de ce premier tour de 2021. Tous les sujets qui sont confrontés au puzzle p4v1 sauf un le valident. Pour ce dernier, la fin de la session a interrompu l'activité sur ce puzzle. Nous notons à nouveau une forte progression de l'usage spontané de la boucle au cours de la scolarité élémentaire.

		2021t1p4v1	
		Effectif	Fréquence
CE	boucle	10	0,24
	séquence	31	0,74
	programme non valide	1	0,02
CM	boucle	24	0,55
	séquence	20	0,45
	programme non valide	0	0
6ème	boucle	9	1,00
	séquence	0	0
	programme non valide	0	0

Tableau 12.8 – Type de programme soumis par les sujets sur le puzzle 2021t1p4v1

En CE, 24% des sujets valident le puzzle p4v1 en utilisant spontanément une boucle. Cette proportion passe à 55% en CM et 100% en 6ème.

Nous nous intéressons au parcours des sujets qui ont validé le puzzle p4v1 avec une séquence. Pour 74% des CE (23 sur 31) et 85% des CM (17 sur 20), une première maîtrise de la boucle est observée à travers la résolution du puzzle p3v2. Lorsque, pour le puzzle p4v2, la boucle est à nouveau contrainte, 65% des CE (20 sur 31) et 90% des CM (18 sur 20) la mobilisent et réussissent le puzzle.

Synthèse

Ainsi, que ce soit pour les élèves scolarisés au collège ou à l'école élémentaire, nous observons une régression passagère à la séquence lors du puzzle p4v1 pour lequel la boucle n'est pas contrainte, et un passage à la boucle avec deux instructions dans son corps lors du puzzle p4v2 pour lequel la boucle est à nouveau contrainte. Plus les sujets sont jeunes, plus la proportion de sujets concernés par cette régression est importante. Des investigations complémentaires sont nécessaires pour comprendre ce qui provoque ce recours à une séquence.

12.3.2 Analyse qualitative de quelques confrontations

Lorsque les sujets sont confrontés à un puzzle pour lequel le motif algorithmique comporte au moins deux éléments, ils ont déjà résolu des problèmes avec une seule instruction dans le bloc *répéter*, souvent de manière experte. Quelle est la nature de la difficulté qu'ils rencontrent lors du passage de une à deux instructions dans le corps de la boucle, c'est à dire lorsque la longueur du motif algorithmique passe de 1 à 2, la taille du motif visuel restant inchangée ?

Nous cherchons à comprendre la cause de cette régression passagère de la

boucle vers la séquence par une analyse qualitative de quelques confrontations représentatives.

Sam (6ème) : passage spontané à la boucle après un début en séquence

Sam, dont c'est la première session Algoréa, édite deux motifs algorithmiques en séquence pour le puzzle p4v1, puis il remplace aussitôt la deuxième occurrence du motif par un bloc *répéter* qu'il place autour du premier motif. Il achève ensuite la résolution de manière experte (Sam, 6ème, 2022t1p4v1). Pour Sam, le fait de voir la suite de deux occurrences de représentation du motif algorithmique dans son début de programme a suffi pour déclencher l'utilisation du bloc *répéter*.

Lou (CM2) : passage spontané à la boucle à la fin de la résolution du puzzle p4v1

Lou, dont c'est la première session Algoréa, a édité un programme en séquence pour le puzzle p4v1 (Lou, CM2, 2021t1p4v1 et v2). Avant de valider, elle teste si elle peut placer deux blocs à l'intérieur du bloc *répéter*. Elle exprime verbalement qu'elle a compris qu'elle pouvait utiliser une boucle, ce qu'elle fait aussitôt dans la version suivante. La régression passagère de Lou à la séquence est due au fait qu'elle ne savait pas qu'elle pouvait mettre deux blocs dans le bloc *répéter*.

Ali (CM1) : passage à la boucle après une consultation spontanée du tutoriel

Ali a validé le puzzle p4v1 avec une séquence. Lorsqu'il ouvre le puzzle p4v2, il exprime immédiatement qu'il se heurte à une difficulté (Ali, CM1, 2021t1p4v2).

Ensuite, Ali lit le tutoriel, indique verbalement qu'il a compris, puis édite la solution de référence avec une boucle. Pour Ali, c'est donc une consultation spontanée du tutoriel qui amène le passage à la boucle avec deux instructions.

Tom (CE1) : synthèse de l'écriture d'une répétition avec un bloc *répéter* non robuste

Tom maîtrise la boucle avec une seule instruction, il vient de valider les trois premières versions du problème p3. Cependant, il valide le puzzle p4v1 avec une séquence, et met plus de 9 minutes à valider le puzzle p4v2, avec plusieurs

interventions de l'expérimentatrice (Tom, CE1, 2021t2p4v2). À l'ouverture du puzzle, Tom exprime que le nombre de dominos à ramasser va le mettre en difficulté, mais il édite quand même une séquence d'instructions, jusqu'à avoir utilisé le nombre de blocs autorisé. À ce moment, Tom consulte le tutoriel une première fois, mais il ne le comprend pas. La difficulté déjà repérée au tour précédent lors du passage de la séquence à la boucle simple ressurgit. Que le nombre renseigné dans le bloc *répéter* représente le nombre de motifs algorithmiques n'est pas encore assez ancré pour permettre d'élargir la classe des situations qui peuvent être résolues avec une boucle. Tom a à nouveau besoin d'un étayage conséquent de l'expérimentatrice pour surmonter la difficulté.

Inès (CE1) : explicitation de la difficulté lors de l'entretien a posteriori

Inès n'a pas réussi à valider le puzzle 2022t1p4v2 pendant le temps imparti. L'ordre d'exécution des instructions n'est pas totalement acquis. La synthèse sous forme d'écriture chiffrée ne vaut que pour l'instruction de déplacement (Inès, CE1, 2022t1p4v1, son coupé à quelques moments pour préserver l'anonymat). Inès a été interrompue en fin de session et le puzzle est repris avec l'expérimentatrice lors de la phase de *debrief*. L'échange suivant met en évidence qu'Inès a une première connaissance du fonctionnement du bloc *répéter*, mais que celle-ci est très contextualisée.

EXPERIMENTATRICE: Est-ce que tu peux m'expliquer comment tu l'utilises maintenant le bloc répéter ?
INES: Ben maintenant dès que je veux avancer mais que j'ai pas assez de blocs.. ben je fais le bloc répéter
EXPERIMENTATRICE: Oui.. et comment tu t'en sers ?.. comment tu fais ?
INES: Ben je compte les cases et après j'appuie sur le répéter dix fois
EXPERIMENTATRICE: Mm
INES: Et il met le nombre que je voudrais mettre
EXPERIMENTATRICE: D'accord.. et c'est tout ?.. et dedans ?
INES: Et je mets le bloc avancer
EXPERIMENTATRICE: Toujours le bloc avancer ?
INES: Ben.. euh.. j'mets le bloc qui faut mettre.. l'bloc que j'ai envie de mettre comme avancer
EXPERIMENTATRICE: D'accord.. donc
INES: C'est souvent avancer

Inès pense qu'on ne peut placer que le bloc *avancer* à l'intérieur du bloc *répéter*. Elle a construit un théorème-en-acte, « C'est toujours le bloc avancer dans le bloc répéter », beaucoup trop contextuel, et dont la non remise en cause l'empêche de

résoudre la nouvelle situation. La compréhension passe par une régression vers la séquence, celle-ci étant laissée dans un premier temps à l'intérieur du bloc *répéter* (Inès, CE1, 2022t1p4v1).

12.3.3 Analyse des difficultés rencontrées en termes de genèses instrumentales et conceptuelles

À partir de ces quelques exemples, nous analysons les difficultés rencontrées en termes de genèses instrumentales et conceptuelles. Le passage d'une à plusieurs instructions dans le corps de la boucle relève-t-il d'un schème d'usage du bloc *répéter* ou d'un schème d'action instrumentée relatif au concept de répétition ?

Difficultés liée à l'extension du schème d'usage du bloc *répéter*

Lou et Ali ont déjà repéré le motif algorithmique lors de la version 1 étoile. La verbalisation pour Ali, les mouvements de souris pour Lou en témoignent. Mais ils ne synthétisent pas l'écriture de la suite de motifs avec une boucle.

L'utilisation du bloc *répéter* dans cette situation implique qu'ils construisent une extension du schème d'usage de ce bloc qui code la boucle en Scratch. Auparavant ils tenaient pour vrai le théorème-en-acte « un seul bloc à l'intérieur du bloc *répéter* ». Cette règle, non pertinente, peut être induite par l'espace disponible dans le bloc lorsqu'il est vide. Le sujet ne voit pas que l'espace s'adapte au nombre de blocs présents. Lou et Ali questionnent cette règle à travers leurs manipulation et leurs verbalisations, soit de manière autonome (Lou) soit avec l'aide du tutoriel (Ali), afin de la faire évoluer. L'efficacité de placer sur l'interface une représentation visuelle du bloc *répéter* avec deux blocs dedans, comme dans le tutoriel consulté par Ali, est montré par Vaníček et al. (2023).

Pour ces sujets, le passage d'une à plusieurs instructions à l'intérieur du corps de la boucle relève clairement de l'extension du schème d'usage du bloc *répéter*. Ils font évoluer le schème d'usage de ce bloc en intégrant qu'il est possible de placer plusieurs blocs à l'intérieur. Dans toutes les confrontations suivantes de ces sujets, ce type de puzzle est résolu directement avec une boucle même lorsque la résolution avec une séquence d'instructions est possible.

Difficultés d'ordre conceptuel

En revanche, chez certains jeunes sujets, scolarisés en CE1, le passage d'une à plusieurs instructions est beaucoup plus laborieux, voire provoque un blocage

que le tutoriel ne lève pas, ni même l'étayage de l'expérimentatrice. Pour ces sujets, nous identifions des difficultés liées à des conceptualisations antérieures, qui nécessitent d'être mobilisées, mais qui manquent de robustesse : caractère séquentiel du programme (Inès), représentation symbolique d'une collection par une écriture chiffrée (Tom et Inès). Pour Inès, nous retrouvons ces difficultés lors des tours suivants (nous n'avons pas cette information pour Tom, qui n'a pas poursuivi l'expérimentation).

Synthèse

Ainsi, lorsque la difficulté du passage d'une à plusieurs instructions est d'ordre instrumental, elle est résolue assez facilement, mais que ce n'est pas le cas lorsque cette difficulté est due à des prérequis insuffisamment ancrés. Parmi ces prérequis, la compréhension de l'ordre d'exécution des instructions par rapport à leur placement dans le programme est une compétence de base du domaine de la programmation, associée à la pensée algorithmique. En revanche, la représentation symbolique d'une collection par une écriture chiffrée, relève plus du domaine des mathématiques.

12.4 Analyse des procédures expertes pour les situations de la classe 1

Dans cette section, nous cherchons à déterminer l'organisation invariante de la conduite lors de la programmation d'une boucle pour les situations de la classe 1. Nous étudions les résolutions des sujets dont le profil est expert selon notre catégorisation, c'est-à-dire qui ont validé le puzzle en un seul essai avec un indicateur de rapidité normalisée inférieur à 0,5.

Quelles actions le sujet fait-il sur l'interface ? Dans quel ordre ? Les actions du sujet comprennent la manipulation des blocs du langage de programmation, mais aussi les éléments ou zones de la grille pointés ou survolés, et les éventuelles verbalisations. Afin de disposer de ces éléments nécessaires à l'analyse, nous nous situons à l'échelle du sujet. Nous étudions les extraits vidéos des puzzles p3v1 (lorsque le tutoriel n'a pas été ouvert) et p3v2 des éditions 2021 et 2022, les puzzles p4v1 (si une boucle a été utilisée) et p4v2 de l'édition 2021, et les puzzles p4v1 de l'édition 2022, ce qui fait 147 confrontations exploitables.

12.4.1 Principales actions du sujet sur l'interface

L'observation des enregistrements vidéo permet de repérer des actions du sujet presque systématiquement présentes lors des résolutions expertes : placer le bloc répéter dans l'éditeur (R), placer des blocs à l'intérieur du bloc répéter (M), changer le nombre dans le champ du bloc répéter (N), pointer ou verbaliser les cases de la grille correspondantes à un dénombrement (D) (Tableau 12.9).

Ordre des actions	Nombre d'occurrences
R D N M	43
D R N M	32
D N R M	13
D R M N	9
R M D N	9
R N M	6
R M N	4

Tableau 12.9 – Actions du sujet lors de la programmation d'une boucle pour une situation de la classe 1

Les trois premières actions sont absolument requises pour la programmation d'une boucle. Le pointage correspondant au dénombrement des cases ne l'est pas, mais il est présent pour 132 des 147 confrontations, soit 90%. Il correspond à une phase de prise d'information, afin de renseigner le nombre d'itérations dans le bloc *répéter*. Nous en déduisons que le schème de dénombrement (4.4) instrumenté par le pointage de la souris fait partie du système hiérarchique de schèmes mobilisé lors de la programmation d'une boucle.

12.4.2 Ordre des actions

La programmation d'une boucle implique de dénombrer les itérations et d'identifier les instructions à répéter lors de chaque itération. Dans notre contexte, cette deuxième sous-tâche revient à identifier le motif visuel sur la grille et en déduire le motif algorithmique correspondant. Nous examinons l'ordre dans lequel le sujet réalise le dénombrement et la conception de la représentation du motif algorithmique dans le langage de programmation. Dans 76% des cas (112 sur 147), la phase de conception du motif algorithmique (M) est en dernière position. Cela signifie que le sujet se focalise d'abord sur le nombre d'itérations avant de renseigner ce qu'il répète. Cette organisation est très majoritairement observée (exemple : Alix, CM2, 2022t3p4v1). Le motif est identifié avant le dénombrement pour seulement 14 résolutions, soit environ 10% (exemple : Lou,

6ème, 2022t1p4v1). De manière assez surprenante, nous observons aussi 13 résolutions pour lesquelles la phase de motif (M) est imbriquée dans le schème de détermination du nombre d'itérations, entre la phase de dénombrement (D) et la phase d'édition du nombre trouvé dans le bloc *répéter* (N).

Dans un petit nombre de cas, la phase de dénombrement est présente plusieurs fois, le sujet éprouvant le besoin de vérifier le résultat du dénombrement (6 occurrences).

Nous nous demandons à présent si l'organisation est invariante pour un sujet donné. Le tableau 12.10 montre la répartition pour les 10 sujets dont au moins 7 des résolutions sont expertes. Par souci de lisibilité, nous ne montrons que les organisations qui apparaissent au moins deux fois pour un même sujet. Nous constatons que l'organisation n'est pas complètement invariante pour un sujet donné, mais qu'une organisation est souvent privilégiée, de manière fort logique l'une des deux plus fréquentes.

Sujet	Classe	RDNM	DRNM	DNRM	RMDN	DRMN	NRM	Nombre de résolutions expertes
Alexis	CM2-6ème	4	5	0	0	2	0	14
Colin	CM2-6ème	5	0	0	2	2	0	12
Élana	CE2-CM1	2	6	0	1	1	0	11
Alix	CM1-CM2	3	5	0	0	0	0	11
Léa	CE1-CE2	0	2	5	0	0	2	10
Lou	CM2-6ème	3	0	0	3	1	0	10
Ali	CM1-CM2	4	5	0	0	0	0	9
Louna	CM2-6ème	6	1	0	0	0	0	7
Noé	4ème-3ème	6	1	0	0	0	0	7
Ugo	CM1	0	0	5	0	0	0	7

Tableau 12.10 – Répartition de l'organisation des résolutions pour les sujets dont au moins 7 des résolutions sont expertes

Selon Rogalski et Samurçay, qui s'appuient sur les résultats de travaux antérieurs : « L'ordre dans lequel les opérations apparaissent dans la planification d'une boucle est le suivant : la description de l'action, l'indication de répétition et le contrôle ; le compteur lorsqu'il est explicité, est placé après la description de l'action. » (Rogalski & Samurçay, 1986).

Nous ne retrouvons pas majoritairement cet ordre, mais l'ordre inverse, pour lequel le compteur est renseigné avant la description des actions à réaliser. Nous questionnons ce résultat divergent par rapport aux travaux antérieurs.

Une hypothèse est qu'il s'agit d'un effet de la particularité des situations en jeu. En effet, la dimension visuelle occupe une place centrale dans les situations de programmation d'un robot virtuel sur une grille. La régularité des motifs

sur la grille attire en premier l'attention du sujet, et l'incite à traiter leur dénombrement en premier. Ce résultat pose un certain nombre de questions qui vont au-delà de cette étude : est-ce que l'ordre de réalisation que nous trouvons est attaché à la résolution de puzzles simples ou est-ce qu'il perdure pour des puzzles plus complexes ? Si c'est le cas, à quel moment se fait la bascule dans l'ordre d'accomplissement des sous-tâches ? Est-ce qu'une telle modification dans l'ordre des sous-tâches est de nature à provoquer une difficulté lors du passage à des situations de programmation où l'aspect visuel est moins prégnant (des situations impliquant des nombres par exemple) ? Dans l'optique de répondre à ces questions, il serait intéressant de comparer l'ordre de réalisation des sous-tâches pour différentes situations de programmation impliquant une boucle, la structure du programme restant similaire.

12.4.3 Nature des verbalisations accompagnant l'action

Sur les 147 résolutions, 47 sont accompagnées de verbalisations. La phase la plus souvent verbalisée est le dénombrement (47 fois), loin devant l'édition de la représentation du motif algorithmique (17 fois) et l'édition du champ numérique du bloc répéter (16 fois). La phase de placement du bloc *répéter* dans l'éditeur n'est verbalisée que 3 fois.

L'explicitation de « répéter » est omise dans les verbalisations. Nous relions ce résultat au statut d'instrument du bloc *répéter*. Dans le cas des situations de la classe 1 de notre étude, la programmation d'une boucle revient conceptuellement à un dénombrement, la représentation du résultat étant instrumentée par le bloc *répéter*. Elle s'appuie donc sur la mobilisation de ce schème de dénombrement. Le bloc *répéter* est utilisé comme support pour synthétiser l'écriture du résultat du dénombrement dans ce contexte particulier, mais l'attention du sujet ne se focalise pas dessus.

12.4.4 Pointage ou survol de cases de la grille

50 des 147 résolutions comprennent au moins une phase d'analyse de la grille, repérée par un pointage de cases ou un survol de la grille. Nous remarquons une différence sur cet aspect entre les situations selon que le motif algorithmique est de longueur un ou deux.

Dans le cas où le motif algorithmique est de longueur un, le survol ou le pointage de cases de la grille concerne exclusivement des puzzles pour lesquels il est nécessaire de placer des instructions avant ou après la boucle (32 des 50 résolutions avec pointage ou survol). Nous n'observons pas de repérage sur la grille concernant la conception du motif algorithmique. Cela n'est pas surprenant car l'identification du motif, qui correspond à la case, est triviale.

Nous en déduisons un schème de pointage, instrumenté par la souris, dont le but est la prise d'information à propos des limites de la suite de motifs visuels, lorsque celles-ci ne sont pas évidentes. En effet, il est nécessaire pour le sujet de se représenter la position du robot juste avant, ou à l'issue de l'exécution de la boucle. L'extrait vidéo de la résolution d'Alix (CM2, 2022t3p3v2) montre à titre d'exemple une résolution experte verbalisée, avec schème de pointage pour repérer les limites de la suite de motifs.

Dans la mesure où les coordonnées du robot sur la grille et l'orientation de celui-ci sont stockés en arrière-plan dans des variables, la position du robot juste avant l'exécution de la boucle correspond à la détermination de l'état initial des travaux de Rogalski et Samurçay (1986). Cependant, le fait que ces variables soient implicites pour le sujet amène celui-ci à faire un repérage visuel et gestuel, par contraste avec un repérage numérique.

Pour les sujets qui accompagnent leur activité de verbalisations, le placement des instructions après la boucle est souvent marqué verbalement : « et », « et maintenant », « après », « puis », « après on va mettre ».

Pour une fraction des sujets, le pointage ou le survol de cases de la grille en rapport avec l'édition du motif apparaît avec le passage à un motif algorithmique de longueur deux (13 résolutions sur 42). Dans la majorité des cas, la prise d'information consiste à pointer la première case avec un motif, celle juste devant le robot. Ce pointage met en évidence la prise en compte de la case comme motif visuel.

Du point de vue de la documentation des schèmes mobilisés, les gestes du sujet avec sa souris nous renseignent sur les phases de prise d'information, en nous indiquant, souvent de manière très précise, sur quelle partie de la grille est focalisée l'attention du sujet.

12.4.5 Synthèse

L'analyse des procédures expertes nous conduit à identifier un ensemble de gestes sur l'interface lors de la programmation d'une boucle bornée, lorsque le motif visuel est délimité par la case. Nous en déduisons un système hiérarchique de schèmes qui rend compte de l'organisation de cette activité du sujet dans les situations de programmation de robot virtuel sur une grille (Figure 12.6). Le schème de niveau hiérarchique le plus élevé, que nous désignons par *schème de boucle bornée*, est structuré, souvent avec une organisation privilégiée, quoique pas totalement invariante.

Le système de schèmes est amené à se complexifier avec la complexification de la situation. Pour la classe de situation 1 étudiée dans ce chapitre, les deux complexifications repérées concernent le passage d'une à deux instructions dans le corps de la boucle, et la nécessité d'instructions avant et après la boucle. Dans

ces deux cas, nous constatons l'apparition de gestes de pointage ou de survol de la grille qui rendent visibles les phases de prise d'information.

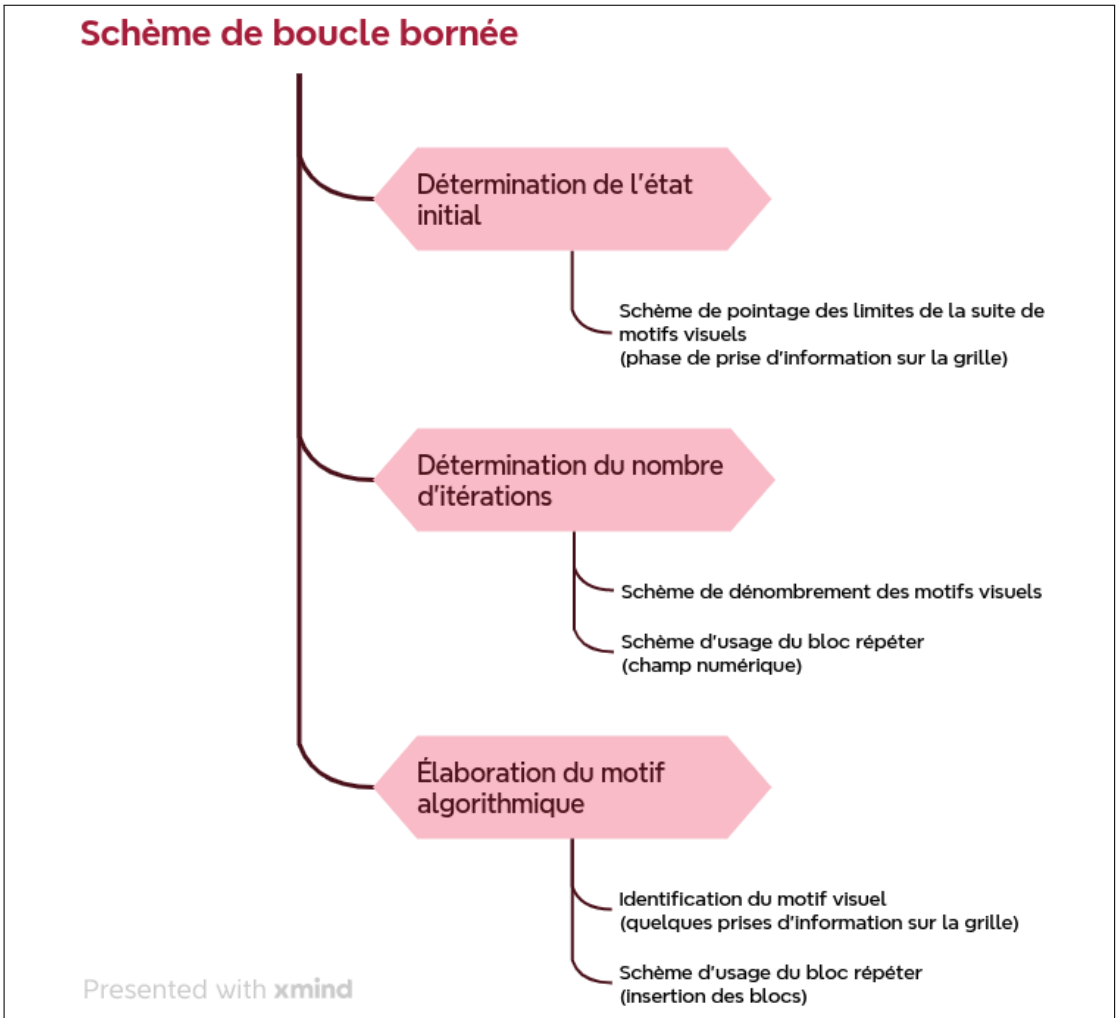


FIGURE 12.6 – Ébauche du système hiérarchique de schèmes pour la programmation d'une boucle bornée en langage Scratch, dans la situation de programmation d'un robot virtuel sur une grille

12.5 Analyse des programmes invalides pour les situations de la classe 1

Nous complétons par une étude quantitative des programmes invalides, exécutés à l'échelle des classes en 2022 pour les situations de la classe 1. Les programmes invalides sont de deux types. Soit ce sont des programmes incomplets, mais qui constituent un début de solution correcte, soit ils comportent une erreur proprement dite, c'est-à-dire qu'en ajoutant des instructions au programme, on n'aboutit pas à un programme correct qui a été exécuté par un sujet appartenant à l'échantillon.

Pour les programmes erronés, notre objectif est d'identifier les types d'erreurs les plus fréquemment commises, d'abord pour un motif algorithmique de longueur 1, puis de longueur 2. Pour les programmes partiels, nous visons d'analyser ce que révèle le recours à la fonctionnalité d'exécution en termes de stratégie de résolution.

12.5.1 Programmes erronés lorsque le motif algorithmique est de longueur 1

Les figures 12.7 et 12.8 montrent les types d'erreurs commis par au moins 5% des sujets pour les problèmes p3v1 et p3v2. Pour chaque puzzle, nous donnons la grille, la solution attendue, la catégorie d'erreur avec un exemple représentatif (le plus fréquent contenant une seule erreur), la fréquence des sujets ayant commis cette catégorie d'erreur. Nous mentionnons aussi le nombre de sujets et de programmes prototypiques (programmes replacés aux coordonnées d'origine de l'éditeur, sans les éventuels blocs détachés) que cela représente.

L'erreur la plus fréquemment commise ne concerne pas directement la programmation d'une boucle. En effet, nous retrouvons l'erreur d'ordre instrumental liée à la prise en main du bloc *pousser la caisse* (sur fond bleu), déjà décrite (10.3.2).

Pour ce qui concerne précisément la programmation de la boucle, les erreurs en rapport avec la sous-tâche de dénombrement sont les plus fréquentes (sur fond rose). Parmi ces erreurs, la plus commune pour tous les puzzles consiste en une itération en plus ou en moins, le nombre renseigné correspondant au nombre de cases vides sur la grille. Nous en déduisons que le sujet dénombre le nombre de cases au lieu du nombre de déplacements du robot. Pour les versions 1, il manque une itération, le sujet ne compte pas la case d'arrivée qui n'est pas vide. Pour les versions 2, nous avons une itération en trop, le sujet comptant à la fois la case de départ qui est vide au moment où il édite son programme et la case d'arrivée.

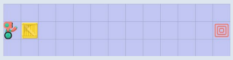
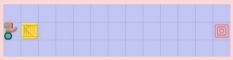
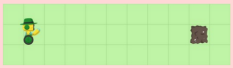
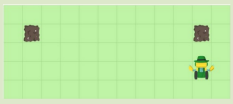
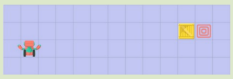
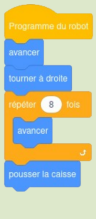
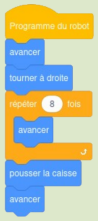
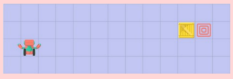
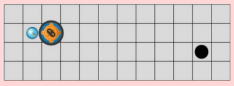
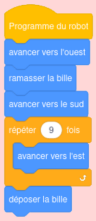
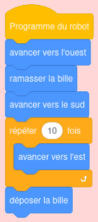
Puzzle / Grille	Solution attendue	Exemple d'erreur	Catégorie d'erreur	Fréquence	Nombre de sujets	Nombre de prog. prot.
2022t3p3v1 			Erreur liée à un contexte spécifique	**0.848**	112	86
2022t3p3v1 			Erreur concernant le nombre d'itérations	**0.689**	91	93
2022t1p3v1 			Erreur concernant le nombre d'itérations	**0.256**	52	20
2022t1p3v2 			Erreur concernant les instructions hors de la boucle	**0.229**	46	46
2022t3p3v2 			Erreur concernant les instructions hors de la boucle	**0.206**	27	28
2022t3p3v2 			Erreur concernant le nombre d'itérations	**0.183**	24	20
2022t2p3v2 			Erreur concernant le nombre d'itérations	**0.161**	30	17

FIGURE 12.7 – Types d'erreur les plus fréquemment commis pour les puzzles nécessitant une boucle avec un motif algorithmique de longueur 1 (p3v1 et p3v2 en 2022) - 1/2

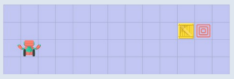
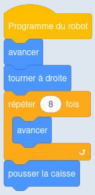
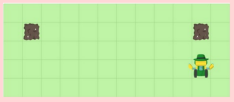
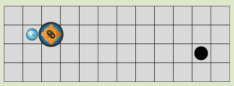
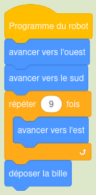
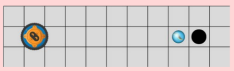
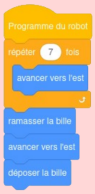
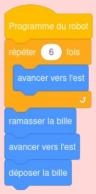
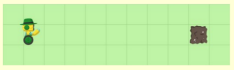
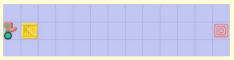
Puzzle / Grille	Solution attendue	Exemple d'erreur	Catégorie d'erreur	Fréquence	Nombre de sujets	Nombre de prog. prot.
2022t3p3v2 			Erreur liée à un contexte spécifique	**0.107**	14	21
2022t1p3v2 			Erreur concernant le nombre d'itérations	**0.1**	20	12
2022t2p3v2 			Erreur concernant les instructions hors de la boucle	**0.086**	16	11
2022t2p3v1 			Erreur concernant le nombre d'itérations	**0.086**	16	9
2022t1p3v1 			Erreur concernant l'identification de motif	**0.069**	14	14
2022t3p3v1 			Erreur concernant l'identification de motif	**0.068**	9	20

FIGURE 12.8 – Types d'erreur les plus fréquemment commis pour les puzzles nécessitant une boucle avec un motif algorithmique de longueur 1 (p3v1 et p3v2 en 2022) - 2/2

Ces erreurs révèlent le décalage de nature entre ce qui est l'objet du schème de dénombrement instrumenté par la souris (des éléments visuels), et ce qui est inséré dans le bloc *répéter* (des actions à effectuer). En effet, la correspondance entre les deux n'est effective qu'à une unité près.

Les erreurs concernant l'identification de motif sont peu nombreuses et très dispersée (nous avons mis un exemple dans le tableau, mais qui est beaucoup moins représentatif que pour les autres types d'erreur).

Une part significative des erreurs concerne les instructions hors de la boucle. Parmi les plus communes pour cette catégorie d'erreur, nous relevons l'absence d'une action à faire réaliser au robot en plus du déplacement. Nous faisons l'hypothèse que le sujet est concentré sur la programmation du déplacement du robot qui est l'aspect le plus difficile, et il oublie de gérer aussi les actions à programmer sur certaines cases.

12.5.2 Programmes erronés lorsque le motif algorithmique est de longueur 2

À nouveau, l'erreur la plus fréquente est d'ordre instrumental. Elle concerne la prise en main d'un bloc spécifique à un contexte (10.3.2).

Lorsque le motif algorithmique est de longueur 2, des erreurs concernant la sous-tâche d'édition du motif algorithmique apparaissent (Figure 12.9). La plus fréquente dans cette catégorie est de placer un seul bloc dans le bloc *répéter* au lieu des deux attendus (erreur commise avec une fréquence d'environ 0,16 sur le puzzle 2022t3p4v1). Cette erreur tend à confirmer que le théorème-en-acte erroné observé chez quelques sujets à l'échelle individuelle, et qui consiste à penser qu'on ne peut placer qu'un seul bloc dans le bloc *répéter*, est assez répandu. L'erreur sur le puzzle 2022t2p4v1, qui consiste à programmer une séquence de boucles à la place d'une boucle avec plusieurs instructions, peut être rapprochée de celle mentionnée dans l'article de Grover et Basu sur les idées fausses lors de l'initiation à la programmation. Les auteurs rapportent que, dans le sens du décodage, les sujets interprètent une boucle avec deux instructions comme une séquence de deux boucles (Grover & Basu, 2017). Les auteurs n'explorent pas le sens du codage. Nos résultats montrent que lorsque le sujet conçoit le programme, nous retrouvons cette erreur dans une proportion comparable (5% des sujets vs 8% pour Grover et Basu).

Concernant les erreurs relatives au dénombrement des motifs, elle régressent fortement lorsque le nombre de motifs algorithmiques correspond au nombre d'éléments saillants (zones de terre, points noirs), ce qui est logique et conforte notre interprétation précédente, puisque dans ce cas la correspondance est exacte avec le nombre de cases dénombrées.

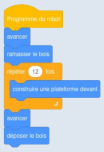
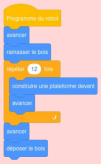
Puzzle / Grille	Solution attendue	Exemple d'erreur	Catégorie d'erreur	Fréquence	Nombre de sujets	Nombre de prog. prot.
2022t2p4v1 			Erreur liée à un contexte spécifique	**0.548**	102	85
2022t2p4v1 			Erreur concernant les instructions hors de la boucle	**0.414**	77	38
2022t2p4v1 		 (cumul avec une erreur en sortie de boucle)	Erreur concernant le nombre d'itérations	**0.398**	74	79
2022t3p4v1 			Erreur concernant l'identification de motif	**0.158**	21	21
2022t3p4v1 			Erreur concernant le nombre d'itérations	**0.143**	19	6
2022t1p4v1 			Erreur concernant le nombre d'itérations	**0.085**	17	10
2022t2p4v1 			Erreur concernant l'identification de motif	**0.054**	10	10

FIGURE 12.9 – Types d'erreur les plus fréquemment commis pour les puzzles nécessitant une boucle avec un motif algorithmique de longueur 2 (p4v1 en 2022)

En revanche, la valeur du compteur de boucle laissée à la valeur par défaut laisse penser que l'attention est plus focalisée sur l'identification du motif, jusqu'à en oublier la tâche de dénombrement (notamment pour le puzzle

2022t1p4v1 pour lequel la différence avec le nombre à renseigner est de deux unités).

Dans la catégorie des erreurs concernant les instructions hors de la boucle, l'erreur la plus fréquente est la mauvaise anticipation de la position du robot à l'issue de l'exécution de la boucle pour le puzzle 2022t2v4p1. Les sujets omettent le bloc *avancer* qui suit la boucle, se représentant le robot directement sur la cheminée après avoir construit la dernière plateforme. La résolution d'Alexis illustre ce cas (Alexis, 6ème, 2022t2p4v1). En première intention, il corrige son programme en modifiant le nombre d'itérations. Il a besoin d'une exécution en mode pas à pas pour identifier la position du robot à l'issue de l'exécution de la boucle, et corriger alors son erreur.

12.5.3 Programmes partiels (motif algorithmique de longueur 1 et 2)

Nous désignons un programme comme *partiel*, si, en lui ajoutant des blocs à la fin, on obtient une solution valide. Dans notre catégorisation, nous avons distingué ces programmes incomplets des erreurs. En effet, bien que ceux-ci soient évalués comme faux par le système, ils ne relèvent pas, pour la plupart, de représentations erronées. La figure 12.10 montre les puzzles pour lesquels la fréquence de programmes partiels est la plus élevée.

Dans le cas présent, l'exécution du programme relève d'une stratégie de vérification et de prise d'information. Pour certains puzzles, notamment ceux qui nécessitent plusieurs boucles en séquence, les programmes partiels sont quasiment aussi fréquents que les véritables erreurs. Nous donnons en exemple la visualisation des programmes invalides les plus fréquemment exécutés pour le puzzle 2022t1p3v3 (Figure 12.11).

Les exécutions de programme incomplet ont lieu majoritairement soit après la programmation d'une action sur un élément saillant (*semer une graine*, *déposer une bille*), soit après la programmation d'une boucle. L'exécution de ces programmes partiels nous renseigne sur la stratégie de décomposition du problème par le sujet.

Nous montrons la résolution du puzzle 2022t1p3v3 par Gina à titre d'exemple (Gina, CE1, 2022t1p3v3). Gina, dont c'est la première session Algoréa et qui vient juste d'acquérir la maîtrise de la boucle avec une instruction lors des deux puzzles précédents, a besoin de vérifier régulièrement si le robot exécute bien ce qu'elle a planifié, en contrôlant l'exécution du programme avec le mode pas à pas. Elle lance une exécution après chaque dépôt d'une graine, après l'édition d'une boucle, et aussi pour contrôler le demi-tour du robot après qu'il ait semé la deuxième graine.

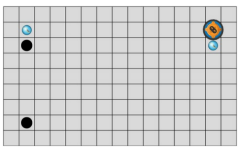
Puzzle / Grille	Solution attendue	Exemple de programme partiel	Fréquence programmes partiels	Fréquence programmes erronés
2022t3p3v3 			**0,585**	0,846
2022t2p3v3 			**0,511**	0,516
2022t1p3v3 			**0,480**	0,540
2022t2p4v1 			**0,446**	0,774

FIGURE 12.10 – Programmes partiels exécutés

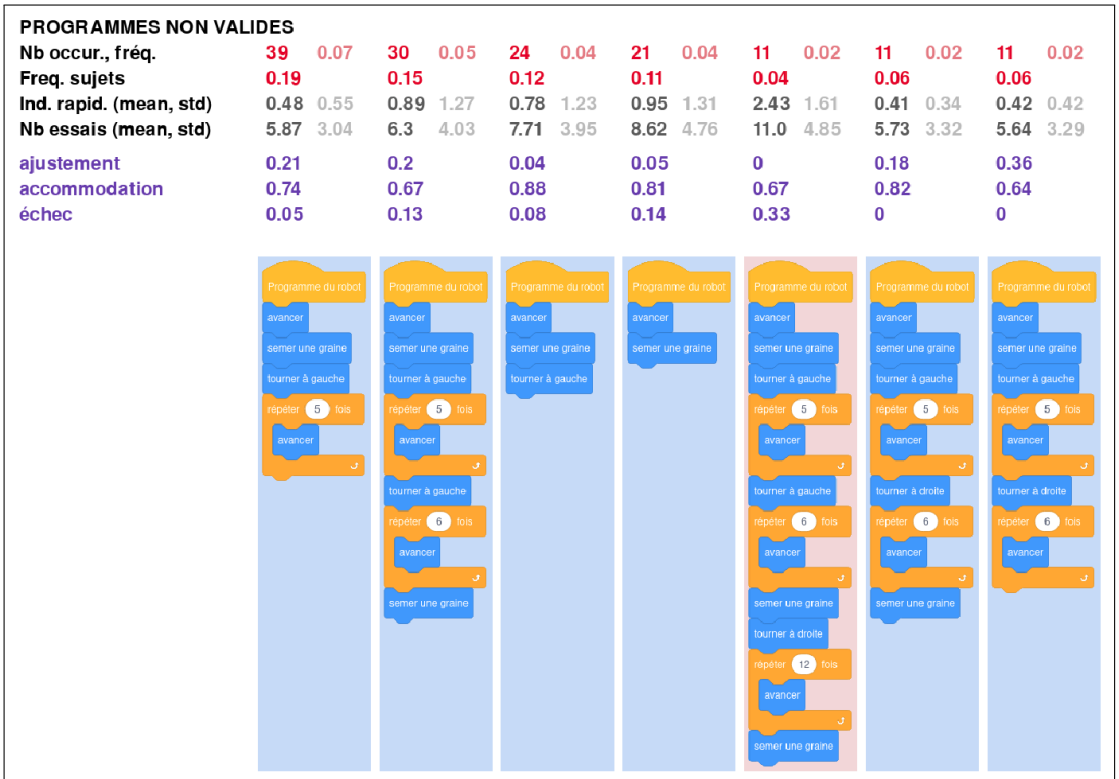


FIGURE 12.11 – Programmes invalides les plus fréquemment soumis pour le puzzle 2022t1p3v3

Ces exécutions relèvent d'une instrumentalisation de la fonctionnalité d'exécution. Le sujet sait que son programme ne va pas valider le puzzle, mais l'exécution lui sert de vérification et de prise d'information pour la poursuite de sa résolution. Notamment, l'exécution d'un programme partiel lui permet de repérer la position du robot à l'issue de l'exécution d'une boucle. Elle montre aussi une certaine fragilité dans la représentation de l'exécution, ou plutôt le degré de complexité du problème pour lequel le sujet est capable à un moment donné de se représenter l'exécution. Nous reviendrons sur ces points dans le chapitre 13.

12.6 Synthèse

Au terme de notre analyse de l'activité du sujet, nous documentons le schème de boucle bornée (encadré ci-dessous), dont nous avons repéré la structure hiérarchique (Figure 12.6). Dans les situations de ce chapitre, ce schème est mobilisé dans le cas d'un motif visuel délimité par la case de la grille.

Schème de boucle bornée

- **but** : programmer la répétition d'une ou deux actions du robot en utilisant un nombre restreint de blocs (et donc respecter la contrainte en nombre de blocs disponibles)
- **règles de conduite de l'action** :
  - **prise d'information** :
    - repérer la position du robot juste avant la boucle (en pointant éventuellement l'endroit sur la grille avec la souris - schème de pointage)
    - dénombrer en se référant à la grille (schème de dénombrement)
  - **action proprement dite** :
    - placer le bloc répéter dans l'éditeur
    - renseigner le champ numérique du bloc répéter par le nombre trouvé
    - placer l'instruction de déplacement, puis le cas échéant l'action du robot sur la case, à l'intérieur du bloc *répéter*
  - **contrôle** : parfois dénombrer à nouveau les éléments de la grille, exécuter le programme édité
- **invariants opératoires**
  - **concepts-en-acte** : séquentialité du programme, correspondance terme à terme
  - **théorèmes-en-acte** :
    - « les instructions placées dans le bloc *répéter* sont exécutées autant de fois que renseigné »
    - « il y a correspondance entre le nombre d'éléments dénombrés et le nombre de motifs algorithmiques »
- **représentations**
  - représentation symbolique d'une collection par une écriture chiffrée

Les différents schèmes du système hiérarchique interviennent à titre de composantes du schème de boucle bornée. Ils sont pour certains imbriqués les uns dans les autres.

Il apparaît que, dans le cas d'un motif visuel délimité par la case de la grille, la programmation d'une boucle relève pour une large part d'une extension du schème de dénombrement déjà documenté dans la littérature (Vergnaud (1991, 2007, 2012), principes de Gelman et Gallistel (1978)). Le champ numérique du bloc *répéter* est utilisé pour renseigner l'écriture chiffrée qui est la représentation symbolique du dénombrement. L'unité pour le dénombrement, qui correspond au motif algorithmique, nécessite d'être spécifiée.

Pour une majorité de sujets (76%), les actions constitutives du schème de dénombrement sont effectuées en premier, avant d'éditer le motif algorithmique. Pour les sujets qui maîtrisent le dénombrement et sa synthèse sous forme d'écriture chiffrée, les quelques difficultés relevées sont d'ordre instrumental, liées à la prise en main du bloc *répéter*. Une large part des erreurs commises lors de ces puzzles concernent le dénombrement, notamment à cause de la nature de ce qui est dénombré par le sujet (cases vs déplacements).

Le repérage du motif visuel sur la grille reste implicite, car évident, surtout lorsque le motif algorithmique ne comporte qu'une seule action. Quelques mouvements de souris relevant d'une prise d'information à propos du motif visuel apparaissent lorsque le motif algorithmique est de longueur 2, ainsi que quelques erreurs relatives à son édition, mais la fréquence en reste relativement limitée (environ 0,07). Les mouvements de souris que nous avons analysés correspondent surtout au repérage du début et de la fin de la suite de motifs, lorsque des instructions doivent être placées avant ou/et après la boucle.

Sur la trajectoire d'apprentissage de Rich et al. (2017) relative au concept de répétition, l'introduction de la boucle bornée correspond au niveau débutant. Nos investigations confirment que, dans l'environnement de programmation Algoréa, maîtriser la boucle bornée en langage Scratch est accessible dès l'école élémentaire, mais nécessite un accompagnement pour la genèse instrumentale du bloc *répéter* pour certains sujets.

Une spécificité des contextes utilisés pour le concours Algoréa (ainsi que de certains contextes de code.org) est que les blocs qui commandent les déplacements et pivotements du robot ne sont pas paramétrés. Ils le sont en revanche dans d'autres environnements (Scratch, Scratch Jr) (Figure 12.12).

Cette spécificité induit qu'il est nécessaire d'utiliser le bloc *répéter* pour programmer une répétition de déplacements, la seule instruction de déplacement se trouvant alors dans le corps de la boucle. Dans les environnements Scratch et Scratch Jr, le bloc *répéter* est utilisé quasi exclusivement pour programmer des boucles avec deux instructions ou plus, les boucles avec une seule instruction

FIGURE 12.12 – Programmation de la répétition de plusieurs déplacements dans l'environnement Algoréa vs avec le logiciel Scratch

étant remplacées par le bloc paramétré.

Nous n'avons pas trouvé de travaux dans la littérature qui explorent spécifiquement ce point, les articles trouvés relevant la difficulté sans préciser comment elle est traitée. Par exemple, concernant Scratch Jr, TOULOUPAKI et BARON, qui ont mené une étude exploratoire auprès d'élèves de CP, mentionnent que ceux-ci sont en difficulté pour identifier un motif à répéter et l'insérer dans le bloc dédié (TOULOUPAKI & BARON, 2019). Cette difficulté les a amenés à faire évoluer leur scénario pédagogique en introduisant la notion par étapes (qui ne sont pas précisées dans l'article). Une perspective intéressante serait de comparer les deux approches, blocs de déplacement paramétrés ou non, en termes de paliers de difficulté dans l'introduction de la notion de boucle.

Chapitre 13

L'identification et le dénombrement de motifs lors de la programmation d'une boucle

Sommaire du présent chapitre

Dans le chapitre précédent, nous avons analysé l'activité des sujets lorsqu'ils sont confrontés à des puzzles de programmation pour lesquels la case de la grille délimite le motif visuel, c'est-à-dire à des situations de la classe 1, telle que définie dans le chapitre 11. Nous investiguons dans ce chapitre l'activité de ces mêmes sujets pour les situations des autres classes de situations, notamment les classes 2 et 3, en comparaison de leur activité sur les puzzles de la classe 1. Pour rappel, les puzzles de la classe de situation 2 sont caractérisés par une correspondance directe entre élément de la grille et action à faire exécuter au robot. Les actions sont de deux natures : déplacement (dans une seule direction ou en orientation absolue) et ramassage/dépôt d'un objet (le fait de peindre peut être vu comme un dépôt de peinture). Les actions de déplacement peuvent être repérées par la limite entre deux cases. Les cases de ramassage ou dépôt sont repérées par un élément saillant dédié. Pour la classe 3, nous avons en plus des actions de pivotement du robot, non repérables par un élément de la grille, et qui rendent partielle la correspondance entre motif visuel et motif algorithmique.

Nous avons déjà constaté dans le chapitre 11 que, à l'échelle nationale, le passage d'une à plusieurs cases pour le motif visuel entraîne une chute significative de la fréquence de réussite au puzzle.

Que se passe-t-il du point de vue de l'activité du sujet lorsque le motif visuel s'étend sur plusieurs cases et plus sur une seule ? Quelles sont les évolutions en termes de schèmes que cette complexification de la situation entraîne ?

Comme pour le chapitre précédent, nous procédons à l'analyse quantitative des procédures expertes. Nous poursuivons avec une analyse quantitative des erreurs commises, que nous regroupons par catégories correspondant aux sous-tâches identifiées à la fois dans la littérature et de manière expérimentale dans le chapitre précédent. Nous détaillons ensuite l'activité des sujets pour chaque sous-tâche, en articulant analyses quantitatives et qualitatives. Nous insistons alors sur l'importance de la simulation de l'exécution dans ces situations, avant de compléter le chapitre par une ouverture sur d'autres classes de problèmes dans lesquelles le concept de motif est impliqué, et par une réflexion sur le caractère aidant et les obstacles engendrés par certaines fonctionnalités de l'EIAH. Comme pour les chapitres précédents, le détail des analyses concernant les exemples mentionnés dans le texte sont disponibles en annexe : annexe 9 pour les extraits vidéo, annexe 11 pour l'analyse quantitative.

13.1 Analyse quantitative des procédures expertes

Comme dans le chapitre précédent, nous étudions spécifiquement la procédure de résolution des puzzles pour les sujets dont le profil est expert. Nous disposons de 38 procédures expertes pour la classe 2 et de 26 procédures expertes pour la classe 3 (contre 147 pour la classe 1). Cette moindre quantité est logique étant donnée la montée en complexité des puzzles et la durée limitée de la session. Malgré ce faible volume, qui constitue un facteur limitant à nos analyses quantitatives, nous parvenons à repérer des évolutions dans les procédures du sujet entre les différentes classes de situation.

13.1.1 Ordre des sous-tâches

Nous commençons par analyser l'ordre des deux principales sous-tâches nécessaires à la programmation d'une boucle (Figure 13.1). Nous constatons que le dénombrement des motifs est plus souvent réalisé avant la conception de la représentation du motif algorithmique avec les blocs du langage. Par rapport à la classe 1, nous observons cependant une répartition moins déséquilibrée pour les classes 2 et 3 (nous avons écarté les solutions avec boucles imbriquées, qui sont plus complexes à analyser). Pour ces deux classes, environ 60% des élèves commencent par dénombrer les motifs et 40% par éditer le motif algorithmique, alors que la répartition est de 75%/25% pour la classe 1.

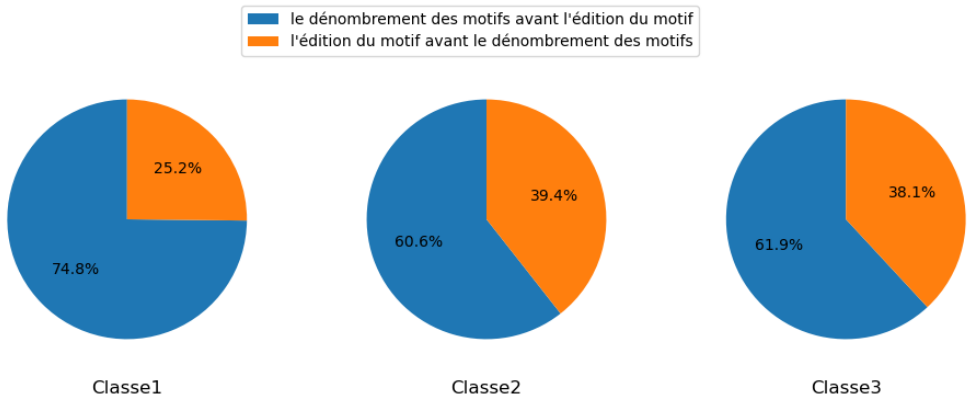


FIGURE 13.1 – Répartition de l'ordre entre l'édition du motif et le dénombrement de ceux-ci selon les différentes classes de situation

À nouveau, ce résultat est en décalage avec les résultats rapportés par ROGALSKI et SAMURÇAY (1986) sur l'ordre des sous-tâches : la description des actions à répéter, l'indication de répétition, puis le contrôle d'arrêt. Cependant, la réduction de l'écart en faveur de l'édition du motif algorithmique comme première sous-tâche réalisée va dans le sens d'un effet de la complexité de la situation. Lorsque le motif visuel est délimité par la case, que son identification est triviale et que la programmation d'une boucle s'apparente à un dénombrement, celui-ci focalise en premier lieu l'attention du sujet. Lorsque la situation devient plus complexe avec un motif algorithmique qui s'étend sur plusieurs cases, l'identifier devient une tâche à part entière, et cela amène certains sujets à inverser l'ordre de réalisation.

Ce résultat renforce aussi notre hypothèse selon laquelle l'ordre des sous-tâche serait dépendant de la nature de la situation.

13.1.2 Prise d'information sur la grille

Nous considérons que les mouvements de souris rendent compte pour une part de la focalisation de l'attention du sujet, et donc de ses phases de prise d'information. En conséquence, nous quantifions la présence d'actions de pointage et de survol de la grille, autres que le pointage accompagnant le dénombrement d'éléments de la grille (Figure 13.2).

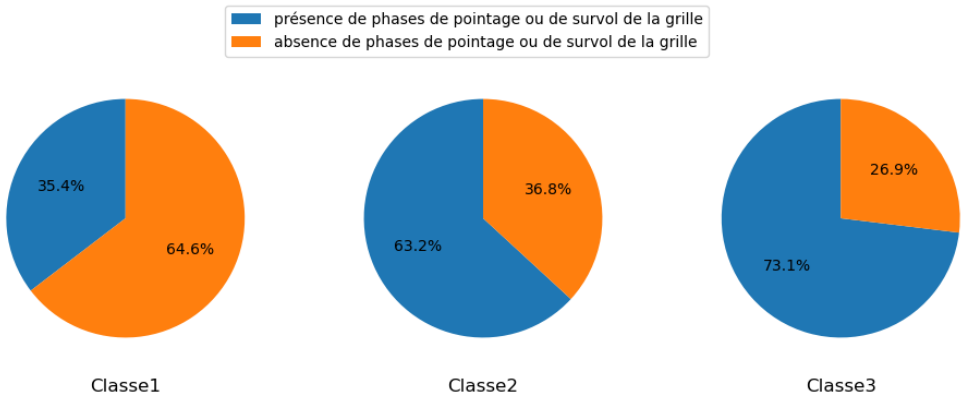


FIGURE 13.2 – Présence / absence d'actions de pointage ou survol de la grille autres que ceux liés au dénombrement

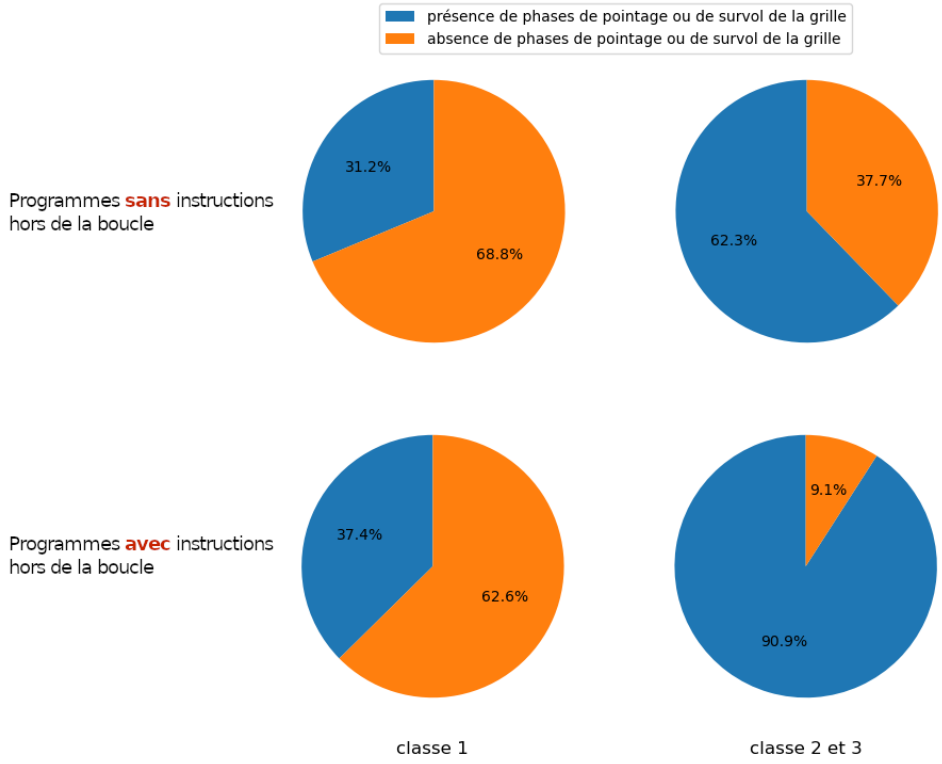


FIGURE 13.3 – Présence / absence d'actions de pointage ou survol de la grille selon que le programme comporte ou non des instructions hors de la boucle

Plus la difficulté des puzzles augmente, plus les phases de prise d'information sur la grille sont présentes. Cette prise d'information devient même presque systématique chez les experts pour les classes 2 et 3 lorsque le programme soumis comporte des instructions hors de la boucle : 10 programmes sur 11 i.e. 91%, contre 62% lorsque ce n'est pas le cas (Figure 13.3).

Nous aurions souhaité caractériser plus finement ces phases de prise d'information, en cherchant si elles sont placées à des moments particuliers du processus de résolution. Nous serions en mesure, avec notre codage des actions, de déterminer le nombre de prises d'information sur la grille à des moments particuliers : juste avant de placer un bloc dans la boucle, juste avant de commencer à programmer la boucle, juste après avoir placé tous les blocs dans la boucle. Pour les deux derniers, il nous est nécessaire de distinguer les puzzles qui nécessitent des instructions hors de la boucle. Or, nous ne disposons que de très peu de puzzles, et donc de procédures expertes, pour ces cas de figure : seulement quelques unités. Une étude quantitative aurait de ce fait peu de sens.

13.2 Analyse quantitative des catégories d'erreurs

Afin de caractériser la difficulté respective des différentes classes de situations, nous poursuivons par une étude quantitative des catégories d'erreur.

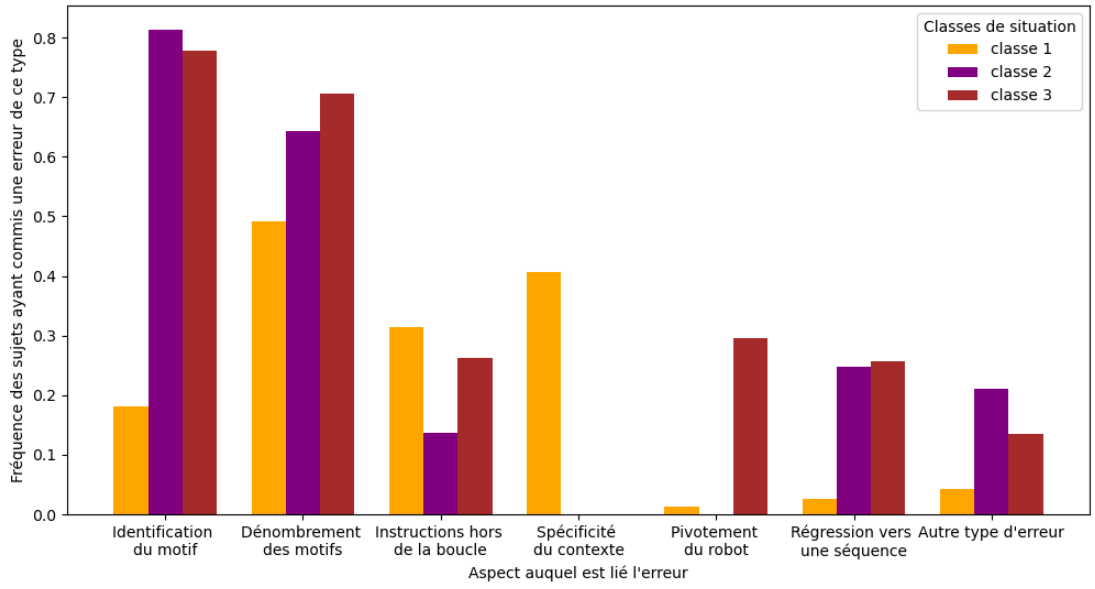


FIGURE 13.4 – Fréquence des erreurs regroupées par catégorie, selon les classes de situations

La figure 13.4 montre, pour chaque classe de situations et chaque catégorie d'erreurs, la fréquence des sujets qui ont commis une erreur dans cette catégorie. Nous constatons des évolutions concernant la fréquence de ces catégories d'erreur, majoritairement entre la classe 1 d'une part, et les classes 2 et 3 d'autre part.

Premièrement, les erreurs qui concernent la prise en main d'un contexte spécifique ont totalement disparu pour les classes 2 et 3. Cela signifie qu'arrivés à ce stade dans le parcours, les sujets ont assimilé les règles des différents contextes et ont une bonne représentation de l'exécution des blocs mis à disposition. Nous remarquons toutefois une exception, qui concerne les blocs de pivotement du robot, et que nous avons traitée séparément. Cette catégorie d'erreur est persistante. Pour la classe 1, les puzzles où un pivotement est nécessaire sont rares. Par définition, ils sont inexistants pour la classe 2. Les erreurs relatives aux pivotements du robot ressurgissent pour les puzzles de la classe 3, pour lesquels un pivotement est systématiquement nécessaire. Nous en déduisons que la représentation de l'exécution des blocs de pivotement, construite lors des premiers puzzles du parcours, n'est pas robuste pour environ un tiers des sujets.

Deuxièmement, les erreurs concernant l'identification du motif apparaissent massivement pour les classes 2 et 3. Avec une fréquence de l'ordre de 0,8, ce sont les erreurs les plus commises. Ce saut en fréquence des erreurs relatives à l'identification de motif confirme le palier de difficulté observé à large échelle. Nous en caractériserons les difficultés dans la section 13.3 par une analyse qualitative des erreurs et des théorèmes-en-actes erronées sous-jacents.

Troisièmement, les erreurs relatives au dénombrement persistent, et même s'accentuent, alors que le domaine numérique mobilisé est plus restreint. Les investigations de la section 13.4 apporteront des éléments d'explication sur ce point.

Enfin, les difficultés attachées à ces erreurs, que nous caractériserons dans les sections suivantes, provoquent une régression vers un programme sous forme de séquence à la place d'une boucle attendue chez environ un quart des sujets pour les puzzles des classes 2 et 3.

13.3 L'identification du motif

Pour les situations de la classe 1 étudiées dans le chapitre 12, le repérage du motif visuel, qui correspond à la case, ne pose aucune difficulté au sujet. Il est peu détectable à travers son activité sur l'interface, en-dehors du placement de blocs (un ou deux) dans le bloc *répéter*. L'analyse quantitative des procédures expertes a permis de montrer que le passage aux classes de situation 2 et 3 est concomitant à des actions de pointage et de survol de la grille plus nombreuses. Dans cette

section, nous cherchons à affiner l'analyse par une étude qualitative de quelques confrontations représentatives. Dans un premier temps, nous répertorions les procédures mises en œuvre par le sujet pour identifier le motif visuel et concevoir le motif algorithmique correspondant. Nous poursuivons en caractérisant les difficultés rencontrées lors de cette sous-tâche.

13.3.1 Procédures mises en œuvre par le sujet

Nous explorons les procédures observables des élèves (mouvements de souris et verbatims), en rapport avec l'identification du motif visuel sur la grille et la construction de la représentation du motif algorithmique correspondante. Notre but est de caractériser un ou des schèmes relatifs à cette sous-tâche. Nous interprétons ces procédures observables dans le cadre théorique de Vergnaud, en inférant notamment les invariants opératoires sous-jacents.

Quelles sont les actions réalisées par le sujet sur l'interface ? Comment s'effectue la prise d'information ? Peut-on repérer des inférences, des règles de vérification ?

Pointage d'éléments de la grille et simulation de l'exécution avec le curseur de souris

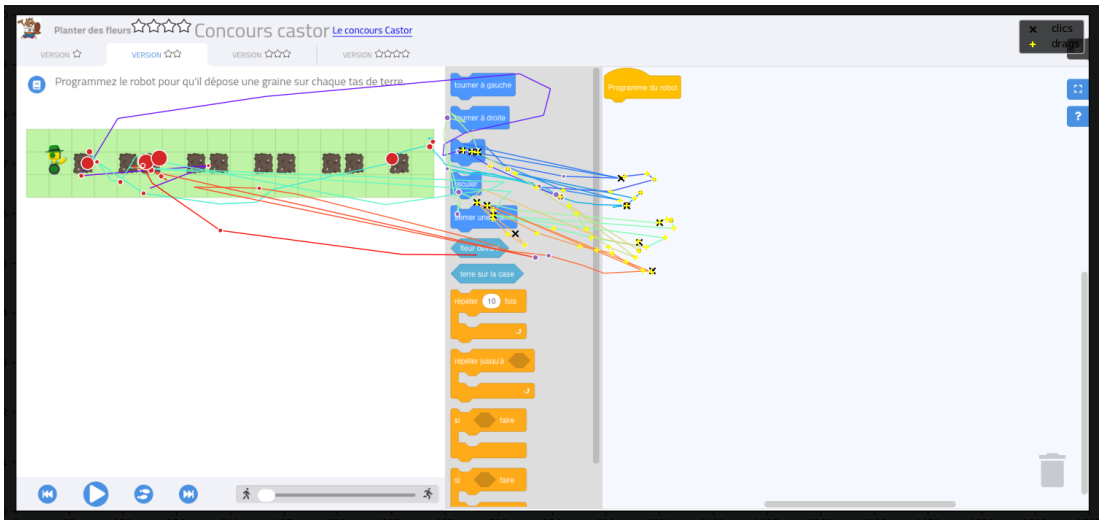


Figure 13.5 – Visualisation des mouvements de souris lors de l'identification du motif visuel et la construction du motif algorithmique correspondant

Une procédure largement observée consiste en des allers-retours avec le curseur de souris entre la grille et l'éditeur au cours de la construction de la

représentation du motif algorithmique avec les blocs. Dans ce cas, le but du schème de pointage est de contrôler la correspondance entre le motif visuel et le motif algorithmique. La figure 13.5 montre une visualisation des mouvements de souris lors de l'identification du motif visuel et de la construction de la représentation correspondante du motif algorithmique dans le langage de programmation (Alix, CM2, 2022t4p1v2). Pour repérer le sens du mouvement, le début du parcours du curseur de souris est en bleu, la fin est en rouge.

Plus précisément, le pointage des éléments de la grille est observé lors de différentes phases de l'élaboration du motif algorithmique.

- Observé en amont de l'édition du motif algorithmique, il relève d'une prise d'information globale, c'est-à-dire en une seule fois pour l'ensemble du motif visuel (Louane, CM2, 2022t2p4v2).

- Lorsque le pointage successif des cases de la grille s'apparente à une simulation de l'exécution en cours de construction du motif algorithmique, il peut être considéré comme une inférence. Le sujet détermine en situation la position courante du robot. Pour chaque case pointée, il ajoute au motif algorithmique les blocs correspondant au traitement de cette case (Robin, CM1, 2022t3p4v3). Dans la confrontation de Robin, l'inférence est marquée verbalement par l'expression « On en est là ». Cette procédure peut être considérée comme l'extension d'un schème de correspondance terme à terme, où l'un des termes est la case, et l'autre les actions à faire effectuer au robot par rapport à cette case. L'extension porte sur le fait que ce traitement peut éventuellement impliquer plusieurs blocs.

- Lorsque la simulation de l'exécution en pointant les positions successives du robot sur la grille est effectuée à l'issue de l'édition de la représentation du motif algorithmique, elle relève d'une règle de vérification, qui peut amener le cas échéant à une rectification de celle-ci. La figure 13.6 en montre un exemple (Lou, 6ème, 2022t4p3v2).

Au cours des phases de prise d'information, nous repérons donc des actions de deux types : soit le sujet pointe des cases particulières, soit il effectue une simulation de l'exécution, le curseur jouant le rôle d'un *robot pion*. Nous reviendrons sur l'importance de la simulation de l'exécution dans la section 13.6.

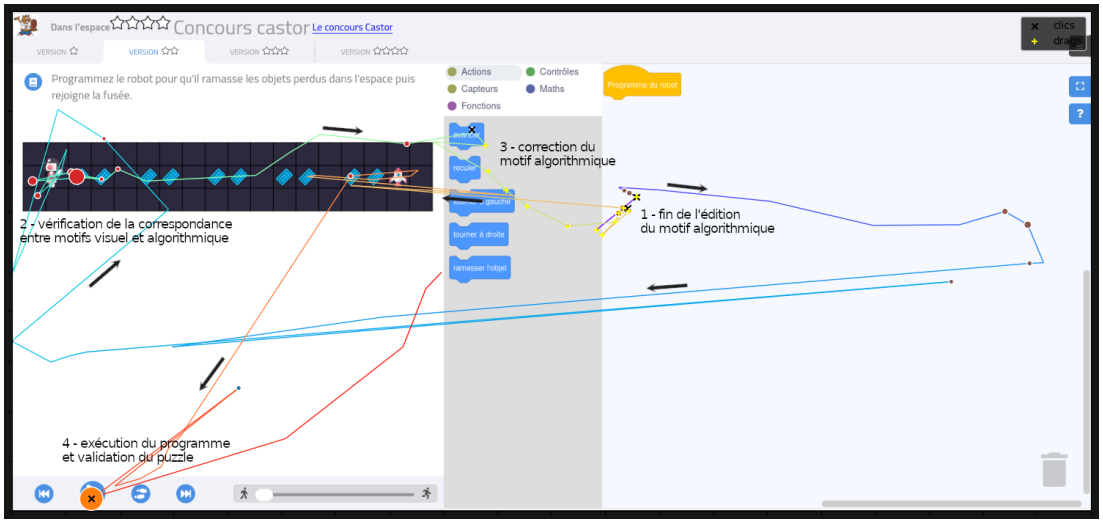


FIGURE 13.6 – Vérification du motif visuel sur la grille, puis correction du motif algorithmique

Prise d'information pour l'édition de la représentation du motif algorithmique sans mouvements de souris associés

Dans un petit nombre de cas, une identification du motif visuel en amont de l'édition du motif algorithmique est réalisée visuellement, sans accompagnement gestuel avec des mouvements de souris. Cette identification est alors repérable par un temps de latence, c'est-à dire plusieurs secondes pendant lesquelles le curseur de souris est immobile, plus rarement par une verbalisation de l'anticipation des actions à faire faire au robot (Robin, CM1, 2022t4p3v2). Dans le cas de Robin, l'identification effectuée uniquement visuellement n'est pas assez précise, le replacement du robot n'est pas fait sur la bonne case.

Test avec un seul motif avant de le placer dans une boucle

Pour quelques sujets, la construction du motif a lieu en amont de la mobilisation de la boucle (Jade, 3ème, 2021t1p5v2). Dans le cas de Jade, le profil est compté comme ajustement à cause de l'exécution du premier motif, alors que l'extrait vidéo montre clairement que cette exécution fait partie d'une règle de vérification. Jade a construit la fonctionnalité d'exécution comme instrument dans le but de vérifier qu'elle a bien identifié le bon motif algorithmique. La procédure est experte.

Identification du motif algorithmique à partir d'un programme en séquence

Certains sujets éditent d'abord un programme en séquence, ou un programme avec plusieurs occurrences de représentation du motif algorithmique, laissées hors de la boucle. Lorsqu'ils se heurtent à la limite en nombre de blocs, ils synthétisent l'écriture de ce programme avec une boucle. Ils identifient le motif algorithmique dans l'ensemble plus grand de la suite de blocs, sans s'occuper du motif visuel au moment où ils réalisent cette synthèse (Alexis, 6ème, 2022t2p5v2).

Le paramétrage actuel de l'interface n'encourage pas cette stratégie. En effet, l'apparition à l'écran du message indiquant que le nombre de blocs autorisé est atteint coupe le sujet dans son élan. Nous reviendrons sur l'effet de la limite de blocs dans la section 13.7.

Synthèse

Cette variété de procédures nous conduit à envisager plusieurs schèmes alternatifs dont le but est le même : élaborer la représentation du motif algorithmique dans le langage de programmation, ce qui, exprimé en termes d'actions, revient à placer l'agencement de blocs adéquat dans le bloc *répéter*. Nous les regroupons en deux catégories, qui correspondent aux deux cas identifiés dans l'analyse a priori (11.2.2).

- schème de mise en correspondance du motif visuel et du motif algorithmique, soit globalement, soit case par case, en simulant l'exécution.

- schème d'identification de la redondance dans le programme déjà édité

Ces schèmes correspondent à ceux mentionnés par Declercq et Zeyringer : « schème descendant où la répétition est saisie dès le début », « schème ascendant où c'est d'abord le corps de la boucle qui est construit, avant d'être intégré dans une répétition » (Declercq & Zeyringer, 2018).

Nous complétons en conséquence le diagramme du système hiérarchique de schèmes, qui se complexifie de manière concomitante à la complexification de la situation (Figure 13.7).

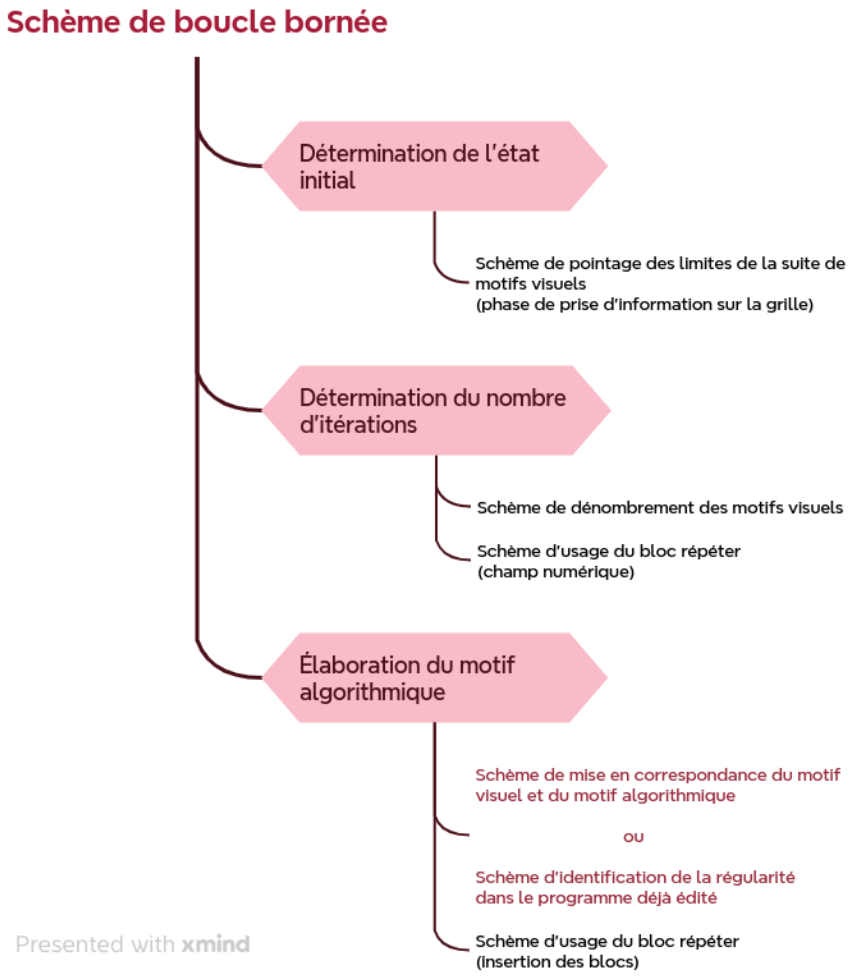


FIGURE 13.7 – Intégration des schèmes relatifs à l'élaboration du motif algorithmique dans le système hiérarchique de schèmes

13.3.2 Analyse des erreurs et difficultés relatives à l'élaboration du motif algorithmique

13.3.2.1 Relevé des erreurs commises

Comme dans le chapitre précédent pour la classe de situations 1, nous relevons les erreurs les plus fréquemment commises pour les classes de situations 2 et 3 à l'échelle des classes. Afin de les mettre en évidence, nous isolons les erreurs relatives à la représentation du motif algorithmique dans le langage de programmation, qui sont les plus répandues pour ces classes de situations (Figures 13.8, 13.9 et 13.10).

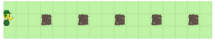
Puzzle / Grille	Solution attendue	Exemple d'erreur	Description de l'erreur	Fréquence	Nombre de sujets	Nombre de prog. prot.
2022t2p4v3 			Il manque une instruction en fin de boucle, qui correspond à la remise du robot dans la même position par rapport au motif visuel qu'en entrée de boucle.	**0.617**	108	11
2022t1p4v3			Deux interprétations possible : - il manque le repositionnement du robot - non distinction des deux motifs visuels	**0.616**	114	22
2022t1p4v3			Le motif est décalé, ce qui ne laisse plus assez de blocs pour terminer le programme	**0.395**	73	28
2022t3p4v2 			Une seule action est prise en compte dans le corps de la boucle ; une case est traitée à la fois, l'invariance du motif algorithmique fait défaut	**0.336**	44	53
2022t1p4v2 			Il manque des instructions de déplacements dans la boucle, comme si l'instruction avancer s'entendait « avancer jusqu'à la case de terre »	**0.24**	47	38

FIGURE 13.8 – Erreurs en rapport avec le concept de motif, les plus fréquentes pour la classe de situation 2

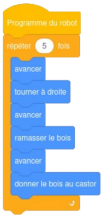
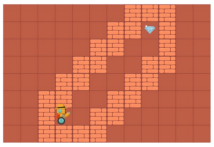
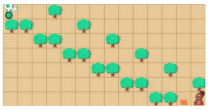
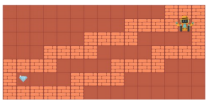
Puzzle / Grille	Solution attendue	Exemple d'erreur	Description de l'erreur	Fréquence	Nombre de sujets	Nombre de prog. prot.
2022t2p5v1 			Il manque une instruction de pivotement en fin de boucle, qui correspond à la remise du robot dans la même orientation qu'en entrée de boucle	**0.594**	104	49
2022t1p5v1 				**0.439**	83	27
2022t2p5v2 		 (plus sortie de boucle non traitée)		**0.385**	62	25
2022t1p5v2 				**0.38**	62	28

FIGURE 13.9 – Erreurs en rapport avec le concept de motif, les plus fréquentes pour la classe de situation 3 - 1/2

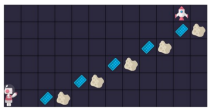
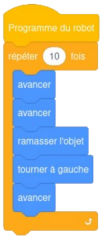
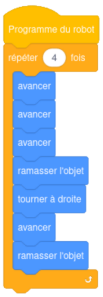
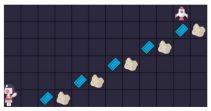
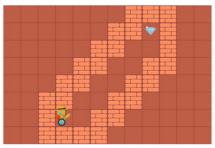
Puzzle / Grille	Solution attendue	Exemple d'erreur	Description de l'erreur	Fréquence	Nombre de sujets	Nombre de prog. prot.
2022t3p5v1 		 (plus compteur à la valeur par défaut)	Il manque une instruction de pivotement en fin de boucle, qui correspond à la remise du robot dans la même orientation qu'en entrée de boucle	**0.339**	43	34
2022t3p5v2 				**0.295**	31	33
2022t3p5v1 			Le motif est décalé, ce qui provoque une erreur en sortie de boucle, ou ne laisse plus assez de blocs pour terminer le programme	**0.26**	33	36
2022t1p5v1 			Trop d'instructions dans le corps de la boucle, qui correspondent à une séquence trop longue mais correcte	**0.217**	41	14

FIGURE 13.10 – Erreurs en rapport avec le concept de motif, les plus fréquentes pour la classe de situation 3 - 2/2

Pour la classe 3, l'erreur qui consiste à omettre l'instruction de pivotement en dernière position du motif algorithmique est de loin la plus fréquente. Nous la retrouvons quel que soit le puzzle de type p5, pour les versions v1 et v2 analysées. Selon le puzzle, entre 30% et 60% des sujets commettent cette erreur. Nous la retrouvons aussi sous une forme un peu différente pour certains puzzles de la classe 2, lorsqu'il est nécessaire de replacer le robot après une action sur une case comportant un élément saillant, avant d'aborder l'itération suivante (2022t2p4v3, 2022t1p4v3 de manière moins certaine). La fréquence d'apparition est comparable à celle des puzzles de la classe 3. Nous en déduisons que le pivotement du robot, s'il ajoute de la difficulté par ailleurs, n'est sûrement pas, comme anticipé dans l'analyse a priori, le facteur qui déclenche cette erreur.

Nous approfondissons l'analyse dans la section suivante, en consultant des confrontations à l'échelle du sujet.

Les autres erreurs fréquentes relevées pour la classe 3 concernent l'identification des limites du motif (2022t1p5v1), et la prise en compte d'un motif décalé qui ne permet pas de concevoir un programme valide avec le nombre de blocs autorisé. Pour la classe 2, nous trouvons aussi le décalage de motif, mais aussi des erreurs pour lesquelles la règle selon laquelle le motif algorithmique est le même pour toutes les itérations, ou celle selon laquelle un bloc code une seule action, n'est pas respectée. Nous abordons quelques erreurs de ce type dans les sections 13.3.2.3 et 13.3.2.4.

13.3.2.2 Analyse de la jonction entre deux itérations successives

L'erreur la plus répandue que nous avons repérée consiste à omettre la dernière instruction de la boucle (plus rarement les deux dernières). Cette erreur met en évidence une difficulté à gérer la jonction entre deux itérations successives. Elle est en lien avec l'identification de la limite entre deux motifs visuels, et avec la mise en correspondance entre motif visuel et motif algorithmique. Cette difficulté a fait l'objet d'une étude spécifique publié dans l'ouvrage collectif du projet IE-CARE, en s'appuyant sur la confrontation de Léa avec le puzzle 2022t2p5v1 (Léa, CE2, 2022t2p5v1). Nous traitons cette difficulté dans cette section, qui, au regard de sa fréquence et de ce qu'elle apporte comme éclairage sur notre analyse de la conceptualisation-en-acte, lui est dédiée.

La difficulté de jonction entre deux itérations successives survient lorsque, après la dernière action faisant partie du motif algorithmique, le robot ne se trouve pas en position adéquate pour aborder le motif suivant. Il est nécessaire d'ajouter un bloc de pivotement ou/et un bloc de déplacement pour repositionner le robot. Ce qui est particulier, c'est que deux prises d'information doivent être réalisées à deux moments différents, qui correspondent à deux endroits particuliers sur la grille. La première phase de prise d'information doit être

faite en se représentant la position du robot juste avant l'exécution de la boucle, et la deuxième prise d'information, qui constitue aussi une vérification se fait à un autre endroit de la grille, en se représentant la position du robot après l'exécution d'une itération. Visuellement, on a donc une opération mentale qui est de l'ordre de la translation, l'ensemble translaté étant constitué du robot et du motif visuel.

Repérage de la difficulté dans quelques confrontations représentatives

La difficulté est présente pour quelques confrontations de la classe 2, bien qu'encore assez discrète. Par exemple, nous la repérons dans la confrontation de Tom sur le puzzle 2021t3p4v3 (Tom, CE1, 2021t3p4v3). Tom repère bien l'unité visuelle dès le premier essai, mais le placement du robot par rapport à cette unité pose problème. Tom surmonte la difficulté en sortant le ramassage du premier objet de la boucle. Ainsi, la dernière instruction dans la boucle est l'action de ramasser l'objet et le robot est correctement replacé pour aborder l'itération suivante sans qu'un replacement ultérieur soit nécessaire.

Un puzzle particulièrement intéressant pour étudier cette difficulté de jonction entre les itérations est le puzzle 2022t2p4v3, pour lequel un replacement du robot après la dernière action est requis, sans impliquer de pivotement. L'erreur la plus fréquente sur ce puzzle est justement d'omettre l'instruction de déplacement après avoir fait déposer le bois au robot (fréquence de 0.62). La confrontation d'Ugo est représentative de la difficulté à replacer le robot dans la position adéquate après le dépôt du bois (Ugo, CM1, 2022t2p4v3). Après avoir exécuté la première itération en mode pas à pas, il affirme avoir compris son erreur, et, sur sollicitation de l'expérimentatrice, l'explicite :

UGO: Alors.. là je ramasse le bois.. je recule.. j'ai compris mon erreur
EXPERIMENTATRICE: Alors c'est quoi ton erreur ?
UGO: Ben en fait.. là *ça va tout de suite reprendre*..

Si la cause de l'erreur semble claire, sa correction fait l'objet de plusieurs essais. Ugo sait qu'il est nécessaire d'ajouter un bloc *avancer*, mais il ne sait pas s'il doit l'ajouter en début ou en fin séquence dans la boucle. Il fait une tentative de placer le bloc au début, tout en manifestant que ce n'est probablement pas la bonne solution : « mais si je mets avancer d'abord.. ça va faire.. un drôle de truc.. ». Ensuite, Ugo ne se représente pas précisément la case sur laquelle le robot doit être replacé. Il le replace dans l'espace vide, avant d'enlever, par ajustement, le bloc *avancer* mis en trop.

Règle de vérification mise en œuvre par les experts

Une règle presque systématiquement mise en œuvre par les experts et qui fait défaut chez les sujets en difficulté est de vérifier la jonction entre les deux premières itérations de la boucle.

- Alix explicite la nécessité du replacement du robot « au même endroit » lors de la phase de prise d'information en amont de l'édition du programme (Alix, CM2, 2022t3p5v1).
- Timéo utilise le mode pas à pas pour vérifier la jonction entre les deux premières itérations avant de poursuivre l'exécution avec le bouton *play* (Timéo, 5ème, 2022t2p5v1).
- À l'issue de l'exécution du motif algorithmique conçu hors de la boucle, Jade pointe la case où le robot était initialement, afin de comparer avec la position atteinte, relativement au motif suivant (Jade, 3ème, 2021t1s5v2).

Analyse en termes de conceptualisation-en-acte

Surmonter cette difficulté revient à comprendre, en acte, le caractère cyclique de la boucle, c'est-à-dire le concept d'itération. Lorsque ce concept sous-jacent n'est pas construit, nous relevons des tentatives de correction inadaptées et des blocages. En revanche, lorsque le sujet comprend le caractère cyclique de la boucle, il parvient à corriger son programme.

- L'extrait de la confrontation d'Élana avec le puzzle 2022t3p5v1, confrontation qui dure en fait 11'50, montre qu'elle ne maîtrise pas le caractère cyclique de la boucle (Élana, CM1, 2022t3p5v1). Élana a repéré que les deux derniers blocs *avancer* entraînent que le robot va trop loin, mais elle ne fait pas le lien avec les instructions avant la boucle. En première intention, elle essaie de changer le nombre d'itérations, en en mettant une de moins, mais se heurte à la limite en nombre de blocs disponibles. Ensuite, elle fait entrer un bloc *avancer* en début de boucle, mais n'enlève pas le bloc correspondant en fin de boucle.
- Après plus de 5 minutes de recherche et consultation du tutoriel, Ali a compris la nécessité de bien replacer le robot par rapport au motif et la jonction entre les itérations. Lors de l'édition du motif algorithmique, il met un bloc *avancer* en trop à la fin de la boucle, puis se ravise en disant « Non le avancer je l'ai déjà fait », montrant ainsi qu'il a compris que l'action d'avancer va être prise en charge lors de l'itération suivante par le premier bloc placé dans la boucle (Ali, CM1, 2021t2p5v1).

- Pour le même puzzle, Sven conçoit le même programme qu'Ali en première intention, mais sa correction consiste à sortir le premier bloc de la boucle (Sven, CM2, 2021t2s5v1). Bien que la solution soit non optimale du point de vue du nombre de blocs, il montre ainsi qu'il a compris le caractère cyclique de la boucle.

- Au cours de sa confrontation avec le puzzle 2022t2p5v2, Alexis montre qu'il a compris le caractère cyclique de la boucle (Alexis, 6ème, 2022t2p5v2). En effet, afin d'économiser des blocs, il supprime le bloc *avancer* avant la boucle, et déplace aussitôt en première position dans la boucle le bloc *avancer* qui était en dernière position.

Ces quelques confrontations montrent que les sujets qui réussissent à corriger leur programme appliquent le théorème-en-acte « On peut sortir un bloc à une extrémité de la boucle si on compense en l'ajoutant à l'autre extrémité ».

Nous retrouvons systématiquement la nécessité de replacer le robot par rapport au motif, et la difficulté associée, pour les puzzles de la classe 3, le replacement impliquant une action de pivotement. Dans le système d'orientation relative, la position du robot comprend la case de la grille sur la quelle il se trouve (ses coordonnées en arrière-plan) et son orientation. Lorsqu'il se représente l'exécution, le sujet doit donc maintenir deux informations en mémoire, dont l'une, l'orientation du robot, n'est pas visualisable avant l'exécution effective du programme, ce qui augmente la difficulté. La fréquence des erreurs relatives aux blocs de pivotement donnent une indication dans ce sens (Figure 13.4). Cependant, la fréquence de l'erreur de non repositionnement du robot sur le puzzle 2022t2p4v3 de la classe 2 montre que la difficulté est due à la nécessité de repositionner le robot, plus qu'à la nature du bloc omis.

Si nous interprétons ce replacement du robot en position adéquate pour aborder le motif de l'itération suivante par rapport à la théorie des champs conceptuels, il peut être associé au concept-en acte d'itération dans son aspect cyclique. Dans la situation présente, ce concept-en-acte est associé à un théorème-en-acte « Au début de chaque itération, le robot est toujours positionné de la même manière par rapport au motif qu'il va parcourir ». Nous mettons en lien ce théorème-en-acte avec le concept plus avancé d'invariant de boucle, qui peut être défini simplement comme une propriété qui est vraie avant et après chaque itération. Une hypothèse est que le fait que le programme soit perçu comme commandant une suite d'actions du robot virtuel sur la grille, et non comme une succession d'états du système, états liée à l'évolution de variables au cours de l'exécution du programme, accentue cette difficulté de conceptualisation. La dialectique entre exécution séquentielle d'actions et succession de situations qui présentent des propriétés statiques a été identifiée par Rogalski dans le contexte

de l'initiation à la programmation textuelle au lycée (Rogalski, 1987). Dans le contexte de la programmation en langage Scratch d'un robot virtuel sur une grille, la succession d'états est très peu visible pour l'utilisateur. Lorsqu'il lance l'exécution de son programme, celui-ci perçoit une succession de déplacements et d'actions du robot virtuel plutôt qu'une succession d'états de la grille dont le robot, avec sa position et son orientation, fait partie. Le replacement du robot par rapport au prochain motif avant chaque itération n'est pas perçu comme une action. Il fait partie du motif algorithmique mais pas du motif visuel. Ce replacement du robot est donc souvent ignoré, au moins en première intention.

L'analyse que nous venons de mener nous amène à rectifier la variable qui distingue les classes de situations 2 et 3 dans notre analyse a priori. Plus que la nécessité de pivotements du robot, c'est la nécessité de repositionner le robot par rapport au motif visuel qui différencie le traitement de la situation par le sujet. Les deux puzzles à être impactés par cette modification sont les puzzles 2022t2p4v3 et 2022t1p4v3, qui passent de la classe 2 à la classe 3.

13.3.2.3 Difficultés conduisant à des solutions non optimales

Certaines difficultés, légères, conduisent à des solutions valides mais non optimales du point de vue de l'expert, optimal du point de vue de l'expert s'entendant comme un programme le plus synthétique possible.

Représentation de ce qu'est un programme optimal

Pour certains sujets, optimal peut s'entendre comme un minimum d'actions du robot, et non comme un minimum de blocs dans le programme. Ces sujets n'incluent pas le dernier motif visuel dans la boucle si celui-ci peut être codé avec moins de blocs, i.e. en ne codant pas le replacement du robot. La confrontation de Lou avec le puzzle 2021t1p4v3 est représentative de ce point de vue (Lou, CM2, 2021t1p4v3).

Besoin de prendre un élément saillant comme repère pour l'identification du motif

La prise de repère par rapport aux éléments saillants peut conduire à des stratégies non optimales. Par exemple, Tom éprouve le besoin de se placer sur le premier élément saillant, après l'avoir traité hors de la boucle, avant de commencer à coder le motif. Le sujet se repère aux objets, la dernière action codée, après les déplacements, est celle qui traite l'élément saillant. Cette stratégie lui permet de contrôler le replacement du robot. Tom renseigne bien le compteur

de boucle avec une unité de moins que le nombre d'objets sur la grille, signe que la procédure est volontaire et maîtrisée (Tom, CE1, 2021t3p4v3, 3ème essai).

Cette procédure permet d'aboutir lorsque le nombre de blocs mis à disposition est assez large par rapport à la solution optimale. Dans le cas contraire, considérer un motif décalé empêche d'aboutir à un programme valide (exemple sur la figure 13.8).

Allongement de la séquence conduisant à plusieurs motifs dans le bloc *répéter*

La stratégie de certains sujets est d'ajouter des instructions dans le bloc *répéter* jusqu'à tomber sur un motif algorithmique qui fonctionne. Cette stratégie conduit souvent à une séquence qui comprend plusieurs motifs algorithmiques, comme dans l'exemple de la confrontation d'Élana (Élana, CE2, 2021t3p5v1).

Cette stratégie montre que le sujet s'appuie sur la séquentialité du programme pour construire le motif algorithmique. La rupture qu'introduit la structure de boucle avec la linéarité de l'agencement du programme, telle que l'expriment Rogalski et Vergnaud (1987), n'est pas encore bien établie.

13.3.2.4 Difficultés spécifiques à certaines caractéristiques des puzzles

Certaines difficultés sont spécifiques à des caractéristiques particulières de puzzles.

Éléments saillants qui ne contraignent pas le parcours du robot

Lorsque les éléments saillants ne contraignent pas le parcours du robot, celui-ci, et donc le motif visuel, doivent être représentés mentalement. L'espace des solutions est plus large, ce qui est source de difficulté supplémentaire dans la mise en correspondance entre motif visuel et motif algorithmique. Par exemple, Alix n'établit pas correctement la correspondance entre motif visuel et motif algorithmique (Alix, CM1, 2021t1p5v2). Pour Alix, seuls comptent le passage sur les cases avec un élément saillant. Elle n'analyse pas le passage du robot sur les cases vides malgré plusieurs exécutions avec le mode pas à pas.

Présence de plusieurs éléments saillants identiques au sein du motif

La présence de plusieurs éléments saillants identiques au sein du motif entraîne une difficulté particulière.

- Léa utilise une boucle pour déposer une graine sur deux zones de terre consécutives (Léa, CE1, 2021t1p4v3). La verbalisation « Du coup, je fais ça plusieurs fois » indique que le motif est identifié, qu'il est créé en tant qu'unité. Mais Léa n'accède pas à la représentation de cette unité avec le bloc *répéter*. Il est probable que le bloc *répéter* déjà utilisé dans le programme constitue un obstacle, car il est nécessaire de placer celui-ci à l'intérieur d'un autre, c'est à dire à imbriquer deux boucles. Il s'agit d'une extension du schème de répétition au cas où le motif comporte lui-même une répétition. En l'état actuel de ses compétences, le sujet applique le théorème-en-acte « On ne peut placer que des blocs en séquence à l'intérieur du bloc répéter ».

- Au cours de sa confrontation avec le puzzle 2021t1s4v3, Alexis expérimente les deux manières d'utiliser la boucle, d'une part pour répéter le dépôt d'une graine sur une case, et d'autre part pour répéter l'ensemble du motif algorithmique. Mais il ne les combine pas pour accéder à une imbrication (Alexis, CM2, 2021t1p4v3). Cette stratégie est persistante lors du tour suivant.

- Ugo passe presque 20 minutes sur le puzzle 2022t1p4v2. Pourtant, il a identifié le motif visuel dès les premières minutes comme le montrent les mouvements de souris (il entoure le motif visuel), et il sait expliciter où se situe la difficulté particulière qu'il rencontre (Ugo, CM1, 2022t1p4v2) :

  UGO: Le bloc répéter autant de fois que tu veux te permet d'économiser des blocs mais c'est justement ce que je faisais.. mais comme il n'y en a pas à chaque.. y'a pas un bloc de terre à chaque.. les uns à côté des autres..

  EXPERIMENTATRICE: Mais quand les blocs de terre ne sont plus les uns à côté des autres..Qu'est-ce qui change ?

  UGO: Ben on peut [utiliser le bloc répéter].. ça nous permet quand même d'économiser mais moins..

Ainsi, lorsque plusieurs éléments saillants identiques sont présents au sein du motif, le sujet mobilise la même stratégie que pour les puzzles de la classe 1. Il considère la case comme motif pour la programmation de la répétition, et édite le programme en utilisant une séquence de boucles et d'instructions intercalées. La difficulté porte sur le changement d'unité à considérer pour le motif visuel, qui n'est plus la case comme dans les puzzles précédents, mais un ensemble de plusieurs cases adjacentes.

Présence de deux éléments saillants identiques qui ne font pas partie du même motif

Ces situations sont catégorisées dans la classe 4 lorsque le motif optimal est contraint, mais suivant le nombre de blocs disponibles, une solution non optimale qui valide le problème peut se rapporter aux classes 2 ou 3.

Lors de sa confrontation avec le puzzle 2021t2p4v3, Alix prend le premier domino comme point de repère pour l'état initial à l'entrée de la boucle : elle ajoute un bloc répéter avant la boucle pour que le robot se trouve sur ce domino (Alix, CM1, 2021t2p4v3). Néanmoins, elle ignore ensuite ce premier domino, dont le ramassage n'est pas codé. Elle pose même la question à l'expérimentatrice sur la nécessité de le ramasser. C'est le groupe de deux dominos bleus qui est considéré comme motif visuel, à partir duquel est envisagé le motif algorithmique. Une exécution en mode pas à pas conduit Alix à rectifier, en ajoutant le bloc *ramasser le domino* manquant. Ensuite, il faut encore environ huit minutes à Alix pour aboutir à une solution valide, à cause d'une difficulté à dénombrer les motifs (examinée dans la section 13.4 dédiée).

Robin met en œuvre une stratégie mixte de résolution, comportant une part d'analyse et une part d'essai-erreur (Robin, CM1, 2022t3p4v3).

Dans un premier temps, Robin ne prend pas en compte la première zone de terre, qui le dérange. Il se rend compte que le motif est « décalé », ce qu'il exprime à deux reprises. Il exprime aussi que ce sont la première et la dernière zone de terre, qui sont isolées, qui lui posent problème. Il voit bien qu'il doit les traiter dans la boucle : il survole le corps de la boucle déjà constitué avec un bloc *peindre la case* qui correspond à l'action de peindre la première zone de terre en disant « Lui je sais pas où le mettre ». Robin sent la nécessité de placer cette instruction dans la boucle, mais il ne sait pas délimiter le motif visuel afin de construire le motif algorithmique correspondant.

Plusieurs tentatives avec le mode pas à pas sont nécessaires, et il semble que Robin trouve un peu par hasard, car à aucun moment il ne montre ou mentionne la limite entre les motifs visuels sur la grille, ce qui est confirmé par la difficulté à les dénombrer. Le pointage des cases constitutif du schème de dénombrement révèle que le premier et le dernier motif comportent trois zones de terre, sans que cette incohérence avec le nombre de blocs *peindre la case* dans la représentation du motif algorithmique ne soit exprimée. Ce pointage révèle que Robin n'a pas construit l'invariance dans l'exécution du corps de la boucle : dans la représentation que révèlent ses gestes de pointage, ce n'est pas toujours le même motif algorithmique qui est exécuté, celui-ci n'est pas répété à l'identique. Lorsque l'expérimentatrice l'interpelle, il met en cause la bonne mise en œuvre du schème de dénombrement, tout en indiquant qu'il compte « les répétitions » et procède par ajustement. Dans cette confrontation, la stratégie est

approximative, le motif visuel n'est pas identifié, la correspondance avec le motif algorithmique n'est que partiellement construite, la nécessité de l'invariance du motif pour la programmation d'une boucle n'est pas construite. La résolution aboutit positivement grâce à une part d'essai-erreur.

Le motif visuel peu lisible a déstabilisé Robin dans ses connaissances-en-acte en cours de construction.

Motifs visuels qui ne sont pas tous exactement identiques

Le fait que les motifs visuels ne soient pas tous exactement identiques demande des capacités d'abstraction au sujet, afin de ne prendre en considération que les éléments pertinents. Cette caractéristique est ce qui spécifie la quatrième classe de situation identifiée à partir des résultats à l'échelle nationale, mais nous repérons déjà cette difficulté pour certains puzzles des autres classes. Notamment, la présence du robot sur la case initiale entraîne que le premier motif est visuellement légèrement différent des autres.

Cela a par exemple gêné Alexis, l'a amené à placer le premier motif algorithmique hors de la boucle, avant de se rendre compte par comparaison qu'il pouvait être inclus dans la boucle. Alexis explicite clairement que c'est la différence d'apparence entre le premier motif et les suivants qui l'a mis en difficulté (Alexis, 6ème, 2022t2p5v2).

13.3.3 Synthèse sur l'identification de motif

Nous avons relevé un certain nombre de stratégies et de difficultés relatives à l'élaboration de la représentation du motif algorithmique. Certaines de ces difficultés sont générales à l'ensemble des puzzles d'une classe de situations, d'autres sont spécifiques à des caractéristiques particulières de puzzles.

Ces analyses nous permettent de documenter des schèmes qui visent un même but : élaborer une représentation du motif algorithmique dans le langage de programmation. Nous employons le pluriel, car nous avons constaté plusieurs variations dans l'organisation de l'activité du sujet pour atteindre ce but. En conséquence, nous développons les composantes de ces schèmes en mentionnant ces alternatives, la mise sous la forme de la définition analytique de Vergnaud s'avérant peu lisible dans ce cas.

Schème de mise en correspondance du motif visuel et du motif algorithmique

Un premier schème consiste à concevoir l'agencement de blocs à insérer dans le bloc *répéter* en mettant les déplacements, pivotements (classe 3) et actions à faire réaliser au robot virtuel en correspondance avec le motif visuel sur la grille. Pour ce schème, plusieurs modalités sont observées pour la prise d'information, qui est plus ou moins fractionnée.

Une première modalité consiste à effectuer la prise d'information en une seule fois. Dans ce cas, le sujet repère la position du robot juste avant la boucle (en pointant l'endroit sur la grille avec la souris - schème de pointage), puis parcourt ou pointe les cases du motif visuel afin d'en repérer les limites, notamment l'endroit correspondant à la fin du motif algorithmique. Cette modalité de prise d'information entraîne une règle d'action qui consiste à agencer consécutivement tous les blocs correspondant au motif algorithmique.

Une deuxième modalité consiste à effectuer la prise d'information case par case et à construire le motif algorithmique de manière concomitante à une simulation de l'exécution. Au cours de cette simulation, le sujet pointe la case courante atteinte par le robot, puis il ajoute à l'agencement de blocs ceux qui correspondent à ce que doit faire le robot par rapport à cette case. Il décide aussi s'il faut arrêter ou poursuivre la simulation, c'est-à-dire si le motif algorithmique est entièrement conçu ou pas (inférence).

La mise en correspondance du motif visuel et du motif algorithmique s'appuie sur le concept de séquentialité de l'exécution des instructions, construit en amont à travers des situations impliquant la programmation d'une séquence d'instructions.

L'élaboration du motif algorithmique est en soi indépendante du bloc *répéter*, même si dans la plupart des cas, le sujet place les blocs correspondant au motif algorithmique directement dans le bloc *répéter*. Nous avons cependant relevé un petit nombre de cas où le motif algorithmique est conçu avant que le bloc *répéter* ne soit placé dans l'éditeur.

Schème d'identification de la redondance dans le programme déjà édité

Le second schème consiste à concevoir ce motif algorithmique à partir d'un programme édité qui comporte de la redondance. Dans ce cas, nous relevons une prise d'information qui consiste à parcourir le programme déjà édité avec la souris. Ensuite le sujet détache une ou plusieurs parties de son programme, qui correspondent au motif repéré visuellement dans l'agencement de blocs. Il place côte à côte ces morceaux de programme afin de vérifier qu'ils sont bien identiques. Si c'est bien le cas, il supprime toutes les occurrences sauf une. Soit celle qu'il garde est déjà dans un bloc *répéter* et il incrémente le compteur, soit il introduit un bloc *répéter* pour la placer dedans.

Invariants opératoires

Les concepts-en-acte et théorèmes-en-acte sont les mêmes que pour les situations de la classe 1, mais leur domaine de validité nécessite d'être étendu aux situations des classes 2 et 3. Nous y ajoutons l'invariance du motif algorithmique pour toutes les itérations, ce qui se traduit par la règle « Ce sont les mêmes blocs qui sont exécutés dans le bloc *répéter* ». Avoir construit cet invariant opératoire de manière robuste devient essentiel lorsque la taille du motif visuel augmente, ou que les limites de celui-ci sont moins claires.

Règle de vérification de la jonction entre les deux premières itérations

Une fois le motif algorithmique édité, une règle consiste à vérifier la jonction avec le motif suivant. Pour cette vérification, certains sujets procèdent avec le mode pas à pas, d'autres en simulant l'exécution avec le pointeur de souris.

Cette règle permet de déclencher la correction du corps de la boucle en cas d'erreur due à la correspondance partielle entre motif visuel et motif algorithmique.

Schème de repositionnement du robot relativement au motif visuel

Compenser la correspondance partielle entre motif visuel et motif algorithmique revient à positionner le robot à l'identique en début et en fin de boucle par rapport au motif visuel à parcourir ensuite. Cette sous-tâche constitue un palier dans le traitement de la situation. Elle distingue les classes de situation 2 et 3. En effet, pour la classe 2, le robot est bien positionné par rapport au motif suivant à l'issue de la dernière action du motif algorithmique par rapport à un élément saillant. En revanche, le repositionnement du robot est nécessaire pour les situations de la classe 3. Nous documentons ce schème de repositionnement du robot par rapport au motif, qui, s'il n'est pas mis en œuvre, conduit à échouer pour les situations de la classe 3.

Schème de repositionnement du robot relativement au motif

- **but** : replacer le robot en position adéquate pour aborder l'itération suivante
- **règles de conduite de l'action** :
  - **prise d'information** : repérer la position du robot une fois qu'il a parcouru le motif visuel ; puis repérer la position que doit avoir le robot pour aborder le motif suivant
  - **action proprement dite** : à la séquence de blocs déjà présents dans le bloc répéter, ajouter les blocs qui permettent d'amener le robot dans la position voulue
  - **contrôle** : lancer une exécution du début du programme (en mode pas à pas) ou se représenter mentalement cette exécution, afin de vérifier la bonne jonction entre le traitement des deux premiers motifs
- **invariants opératoires** :
  - **concept-en-acte** : concept d'itération (dans son aspect cyclique, qui peut être mis en regard du concept d'invariant de boucle plus avancé)
  - **théorèmes-en-acte** : « Au début de chaque itération, le robot est toujours positionné de la même manière par rapport au motif qu'il va parcourir. »

Nous ajoutons un niveau de décomposition à notre système hiérarchique de schèmes (Figure 13.11).

Du point de vue méthodologique, l'analyse très fine, permise par les traces d'interaction à l'échelle des classes et par les enregistrements vidéo à l'échelle du sujet, a permis de rectifier la variable de situation qui différencie les classes 2 et 3. Le fait que nous ayons été en mesure de détecter cette erreur dans l'analyse a priori donne un indicateur de l'efficacité de la méthode adoptée. Cette rétroaction nous permet d'ajuster l'analyse à l'échelle nationale, ce que nous ferons à très court terme en intégrant les puzzles de 2023 et 2024 comme déjà indiqué dans le chapitre 11.

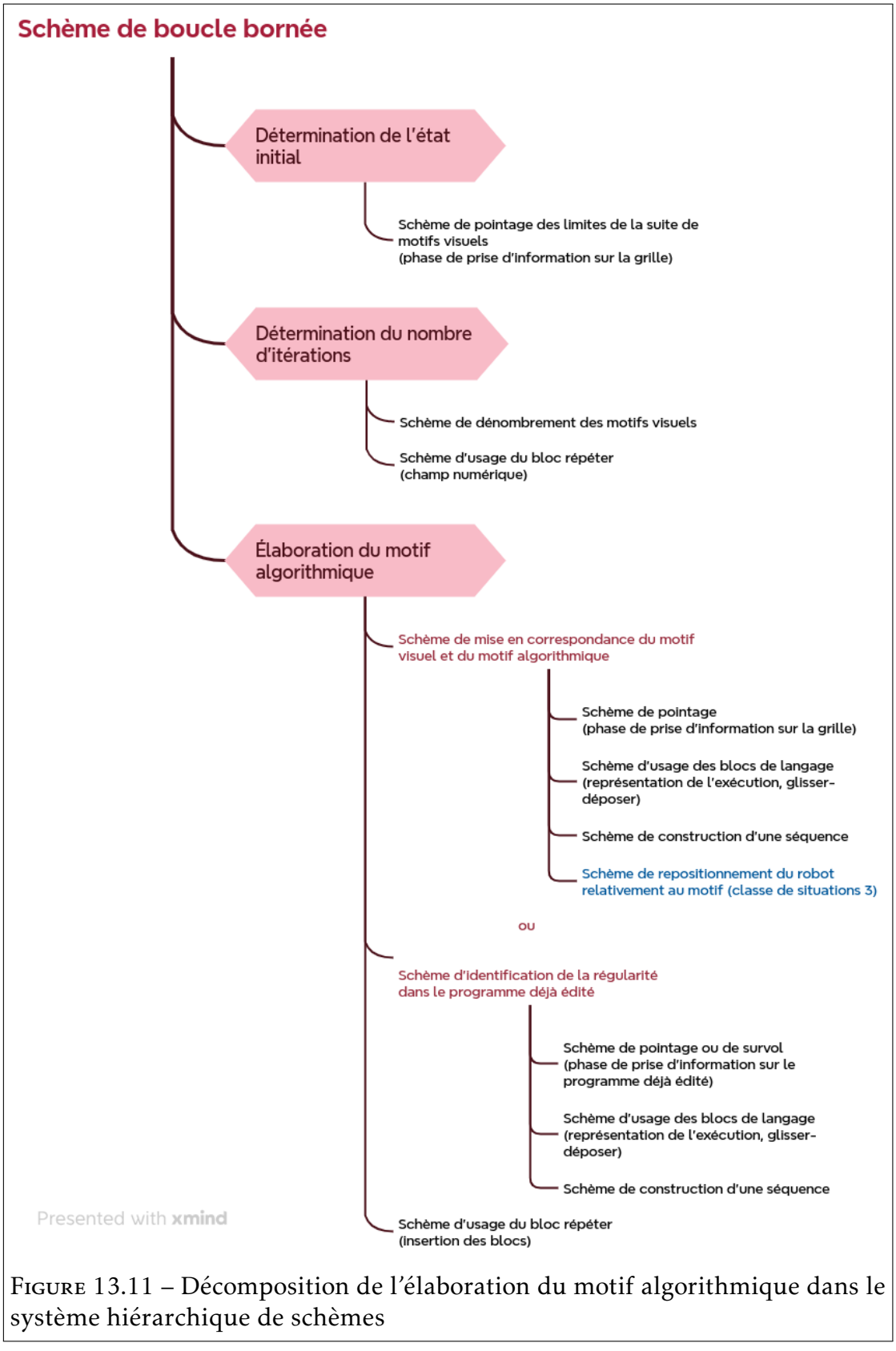


FIGURE 13.11 – Décomposition de l'élaboration du motif algorithmique dans le système hiérarchique de schèmes

13.4 Dénombrement des motifs

Le passage d'une à plusieurs cases pour le motif visuel engendre une rupture au niveau du dénombrement des motifs. L'unité pour le dénombrement n'est plus la case, mais un groupe de plusieurs cases. Nous cherchons à déterminer ce que cette modification de l'unité entraîne comme changements dans le processus de résolution de la situation et comme éventuelles difficultés.

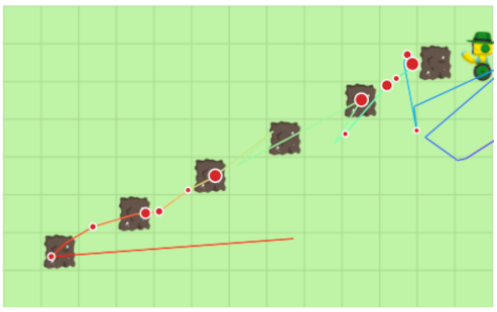
13.4.1 Procédures pour dénombrer les motifs

Nous répertorions plusieurs procédures expertes qui visent à dénombrer les motifs, puis renseigner le champ numérique du bloc *répéter*.

L'une d'elles consiste à simuler le parcours du robot en s'arrêtant sur une certaine case du motif, toujours la même, mais qui peut parfois être une case vide. Par exemple, Lou parcourt le trajet du robot, elle marque un temps d'arrêt sur les cases vides avant les groupes de deux pierres, cases qui correspondent à l'emplacement du robot au début de chaque itération. Simultanément à ce temps d'arrêt qui correspond à un pointage, Lou numérote verbalement de un à six, et termine en totalisant en ajoutant « fois » au numéro attribué (Lou, CM2, 2021t1p5v2). Nous voyons parfaitement dans cette confrontation quel point de repère le sujet prend pour dénombrer les motifs.

Une autre procédure consiste à pointer et numéroter les éléments saillants ou une case particulière du motif sans procéder à la simulation du parcours. La figure 13.12 montre les mouvements de souris associés à chacune de ces deux procédures.

Simulation du parcours du robot lors du dénombrement (Alix, CM2)



Pointage des éléments saillants lors du dénombrement (Élana, CM1)

FIGURE 13.12 – Procédures différentes pour le dénombrement sur le puzzle 2022t4p1v4

Sur la visualisation des mouvements de souris d'Alix (Alix, CM2, 2022t4p1v4), le parcours du robot est simulé avec le curseur, les arrêts sont très peu marqués. En revanche, sur celle d'Élana, la trajectoire va directement d'une zone de terre à l'autre, suivant la diagonale, avec des arrêts plus marqués sur ces zones de terre (Élana, CM1, 2022t4p1v4).

Lorsqu'une case vide sépare deux motifs consécutifs de manière nette, les motifs peuvent aussi être dénombrés globalement, sans recourir à la simulation de l'exécution ou au pointage de cases particulières (Ugo, CM1, 2022t4p3v2).

Plus rarement, ce sont des éléments de décor, disposés régulièrement, qui sont pris comme points de repère pour le dénombrement (Robin, CM1, 2022t2p5v2).

Sur quelques confrontations, nous observons quelques actions qui constituent une prise d'information par rapport à la case qui va être prise comme repère pour dénombrer les motifs. Par exemple, Lou parcourt les blocs du motif dans son programme, puis elle simule l'exécution de ces blocs avec son curseur pour repérer la première case à dénombrer. Ensuite, elle continue à simuler le parcours du robot en marquant un temps d'arrêt et en numérotant à chaque fois qu'elle passe sur la dernière case d'un motif (Lou, CM2, 2021t1p5v2).

Toutes les procédures précédentes concernent le pointage, composante du schème de dénombrement. Dans notre contexte, le schème de pointage est non seulement instrumenté par la souris, mais il peut être aussi articulé avec la simulation mentale de l'exécution du programme, ce qui en constitue une complexification.

13.4.2 Erreurs fréquentes et difficultés associées

Nous avons vu dans la section 13.2 que les erreurs concernant le dénombrement de motifs ne régressent pas pour les classes de situation 2 et 3 par rapport à la classe 1, alors que le domaine numérique est réduit (4 à 6 unités dans la plupart des puzzles). Nous relevons les erreurs fréquentes dans cette catégorie, afin de les analyser et de comprendre les difficultés sous-jacentes.

Dénombrement des cases au lieu des motifs

Une erreur très fréquente est de dénombrer les cases. Cette erreur est commise respectivement par 12%, 7% et 5% des sujets lors des tours 1 à 3 de 2022. Un théorème-en-acte erroné que le sujet applique est « Il faut indiquer un nombre de cases dans le champ du bloc répéter ». Les confrontations suivantes viennent en appui de notre propos.

- Lorsque l'expérimentatrice demande à Lou d'anticiper le résultat de l'exécution avec 1 dans le compteur, elle indique la case juste devant le robot (Lou, CM2, 2021t1p4v3).
- En première intention, Ugo compte les cases, qui correspondent aux déplacements du robot. Puis il se rend compte que le fait que le robot doive s'arrêter pour effectuer une action sur certaines cases empêche de procéder de cette manière (Ugo, CM1, 2022t1p4v2).
- Parfois, le schème de dénombrement est mis en œuvre de manière automatique pour dénombrer les cases, avant que le sujet ne se reprenne et dénombre les motifs visuels (Ugo, CM1, 2022t2p4v3).

Dénombrement des éléments saillants au lieu des motifs

Lorsque le motif contient plusieurs éléments saillants identiques, une erreur est de dénombrer ces éléments saillants à la place des motifs. Plusieurs confrontations à l'échelle du sujet présentent cette caractéristique, par exemple celle d'Ali avec le puzzle 2021t1p4v3 (Ali, CM1, 2021t1p4v3). Malheureusement, comme le nombre d'éléments saillants, dix, est aussi la valeur par défaut dans le champ du bloc *répéter*, nous ne pouvons pas précisément isoler cette erreur, et donc la quantifier à l'échelle des classes.

Difficulté de la construction du motif comme unité pour le dénombrement

Les erreurs relevées précédemment révèlent une difficulté à interpréter le sens du nombre à placer dans le champ du bloc *répéter* lorsqu'il ne correspond plus au nombre de cases.

L'unité prise pour le dénombrement n'est pas en correspondance avec le motif visuel identifié, et le motif algorithmique construit à partir de celui-ci. Il y a discordance entre l'unité constituée dans le programme (séquence d'instructions) et l'unité du dénombrement (cases). La mise en correspondance des unités dans les deux représentations fait défaut. Le sujet ne se rend pas compte qu'il ne compte pas la même chose. Une hypothèse est que pour le schème du dénombrement, le théorème-en-acte suivant est tenu pour vrai : « Les éléments à compter sont les plus petites unités visibles », ce qui est la situation la plus courante. Il n'y a pas, dans un premier temps, création mentale de l'unité que constitue le motif. Le sujet prend l'unité prégnante déjà disponible.

Cette erreur est persistante. Alix, qui fait l'erreur et la corrige avec aide lors du tour 1 de 2021 (Alix, CM1, 2021t1p4v3) la reproduit lors du tour 2 (Alix, CM1, 2021t2p4v3). Pour Alix, ce qui est compté dans le bloc *répéter* sont

les exécutions de blocs individuels, et non les itérations. Les commentaires qui accompagnent l'exécution pas à pas le mettent en évidence. Le sujet dit « Première fois » au moment où la fenêtre pop-up marquant le début de la première itération apparaît, puis « deuxième » au moment de l'exécution du bloc suivant.

13.4.3 Stratégies face aux difficultés de dénombrement des motifs

Nous observons plusieurs stratégies mises en œuvre dans le but de corriger cette erreur. Les tentatives de correction, font intervenir des schèmes déjà constitués, ce qui est de l'ordre d'un processus d'assimilation, ou amènent à modifier certains schèmes lors d'un processus d'accommodation. Les confrontations de Lou avec le puzzle 2021t1p4v3 (Lou, CM2, 2021t1p4v3) et d'Alix avec le puzzle 2021t2p4v3 (Alix, CM1, 2021t2p4v3) montrent l'évolution des stratégies mises en œuvre par les sujets. Nous les détaillons ci-après.

Ajustement du nombre d'itérations en ajoutant ou enlevant une unité au compteur

Ajuster le nombre d'itérations en ajoutant ou enlevant une unité au compteur est le schème spontanément mis en œuvre le plus souvent, dès qu'une erreur dans le compteur d'itérations est détectée. Ce schème relève du processus d'assimilation. Il sous-entend que l'erreur est due à une erreur de mise en œuvre du schème de dénombrement.

Nous retrouvons ce schème à de nombreuses reprises dans les confrontations d'Alix, par exemple pour le puzzle 2021t2p4v3 (Alix, CM1, 2021t2p4v3), mais aussi de Lou (Lou, CM2, 2021t1p4v3).

Ugo explicite cette stratégie d'ajustement lors du debrief après le tour 1 de 2021 (Ugo, CM1, 2022t1p5v1) :

UGO: J'avais mis quatre mais je ne sais pas si c'est bon
EXPERIMENTATRICE: Mm.. donc tu l'as mis un peu au hasard quoi
UGO: Non.. cinq fois.. je me suis dit que c'était un peu trop
EXPERIMENTATRICE: Mais comment tu t'es dit que c'était un peu trop ?
UGO: Ben parce que ça.. Une fois arrivé au diamant.. ça poussait l'obstacle

Cette stratégie permet de réussir à valider le puzzle dans un certain nombre de cas, mais elle n'assure pas la compréhension de ce qui provoque l'erreur. Le sujet ne progresse pas du point de vue de la conceptualisation.

Ajustement en enlevant un bloc en fin de boucle

Cette stratégie constitue une extension du schème d'ajustement du dénombrement dans une situation où ce schème est inopérant. Le sujet ne tient pas compte de la règle « Si on enlève une instruction dans le corps de la boucle, cette instruction est enlevée pour toutes les itérations ». Il ne raisonne qu'avec la dernière itération sans tenir compte des autres. On peut dire que dans ce cas, le caractère invariant de l'exécution du corps de la boucle n'est pas acquis (Alix, CM1, 2021t1p4v3).

Évolution du schème de dénombrement par un processus d'accommodation

Lors du processus d'accommodation, il y a remise en cause du théorème-en-acte « la case est systématiquement l'unité pour le dénombrement ». Le sujet essaie plusieurs unités pour le dénombrement (notamment éléments saillants), avant d'aboutir positivement au dénombrement des motifs visuels.

Par exemple, alors qu'il a correctement identifié le motif visuel, Ali peine à les dénombrer : il dénombre les cases, puis les éléments saillants, avant de parvenir à dénombrer les motifs (Ali, CM1, 2021t1p4v3).

Une aide qui se révèle efficace pour surmonter cette difficulté est de suggérer au sujet de lancer une exécution avec 1 dans le champ du bloc *répéter* (Lou, CM2, 2021t1p4v3). Dans de rares cas, ce passage par l'unité est mobilisé de manière spontanée (Robin, CM1, 2022t3p4v2).

13.4.4 Synthèse sur le dénombrement des motifs

Avec cette analyse, nous avons montré que la rupture entre d'une part les classes de situations 2 et 3 et d'autre part la classe de situations 1 entraîne une complexification du traitement de la situation. Pour les classes 2 et 3, la principale difficulté pour le dénombrement de motif est de considérer le motif visuel comme unité pour le dénombrement, et donc de faire abstraction de la case comme unité alors que celle-ci est toujours disponible.

Le dénombrement de motifs constitue donc une extension du schème de dénombrement, dans la mesure où il est nécessaire de déterminer les unités à dénombrer. Le motif joue en quelque sorte le rôle de mesure étalon, c'est-à-dire d'unité à considérer pour une mesure de grandeur. Vergnaud mentionne une fois le lien entre concept de nombre et mesure des grandeurs : « Le concept de nombre est un autre exemple [de construction déjà élaborée], puisqu'il n'existe pas d'objet nombre dans le monde matériel et que le nombre résulte en premier

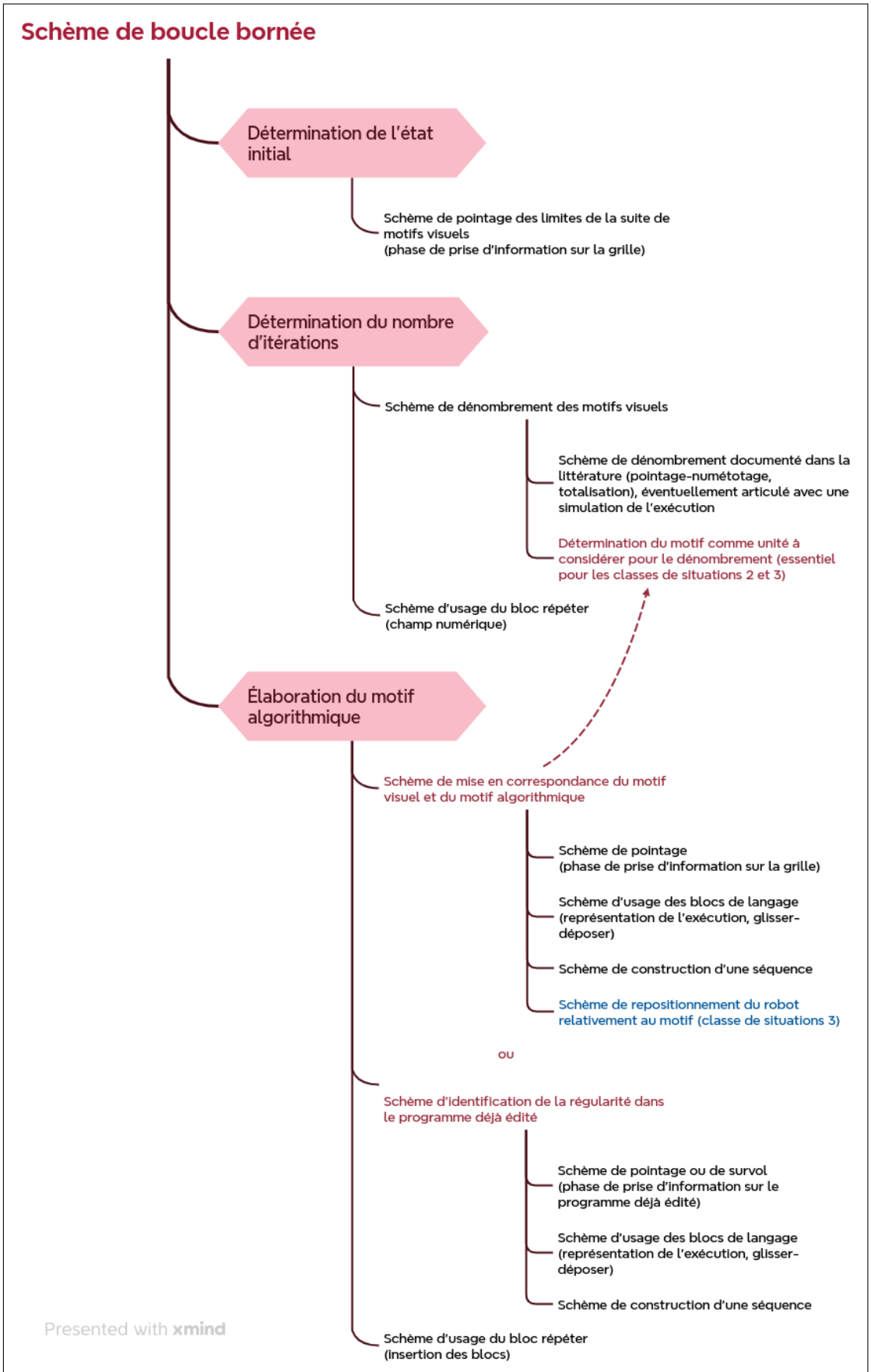


FIGURE 13.13 – Décomposition du schème de dénombrement des motifs visuels dans le système hiérarchique de schèmes

lieu de la mesure, puis de la séparation entre mesure et objet mesuré, dans les activités de dénombrement des quantités discrètes et de mesure des grandeurs. » (Vergnaud, 2013b)

Nous affinons en conséquence le système hiérarchique de schèmes impliqué dans la programmation d'une boucle bornée (13.13).

Il nous semble que notre travail constitue un apport sur ce point. En effet, et contrairement à l'identification de motifs, nous n'avons pas trouvé de travaux qui abordent le dénombrement de motifs dans la littérature relative au concept de motif. Une perspective est de croiser ces analyses avec des investigations dans le champ des mesures et grandeurs en mathématiques.

13.5 Extensions des schèmes construits

Les schèmes construits lors de la programmation d'une boucle sont remobilisés pour d'autres classes de problèmes. Nous avons collecté des données pour des puzzles impliquant des instructions conditionnelles, des variables et des procédures. Cependant, leur traitement s'est avéré trop ambitieux dans le cadre de la présente recherche doctorale. L'analyse de ces données constituent des perspectives pour la continuation de notre recherche. Nous donnons néanmoins un aperçu de la remobilisation des schèmes relatifs au concept de motif identifiés, lorsque cette remobilisation intervient spontanément pour des puzzles que nous avons analysés.

13.5.1 Extension vers l'imbrication de boucles

Le passage à une imbrication de boucles est assez spontanée lorsque la boucle interne est une boucle simple, c'est-à-dire revient à un dénombrement comme nous l'avons montré dans le chapitre 12. Par exemple, 24% des sujets de l'échelle des classes valident le puzzle 2022t1p5v2 avec une boucle imbriquée (Figure 13.14).

Dans ce cas, les schèmes mobilisés pour la programmation d'une boucle sont imbriqués les uns dans les autres. En reprenant le codage introduit au chapitre 12, nous donnons un exemple de succession des actions lors de la programmation d'une boucle imbriquée lors d'une confrontation avec le puzzle ci-dessusedonedondancedance (Louna, 6ème, 2022t1p5v2) : R D N M Ni Ri Ni Mi M. Le i marque les actions correspondant à la boucle imbriquée. Nous constatons que, de manière logique, les actions pour programmer la boucle imbriquée sont enchâssées dans l'édition du motif algorithmique. Pour les sujets qui recourent à l'imbrication de boucle, ce passage constitue souvent une stratégie pour respecter

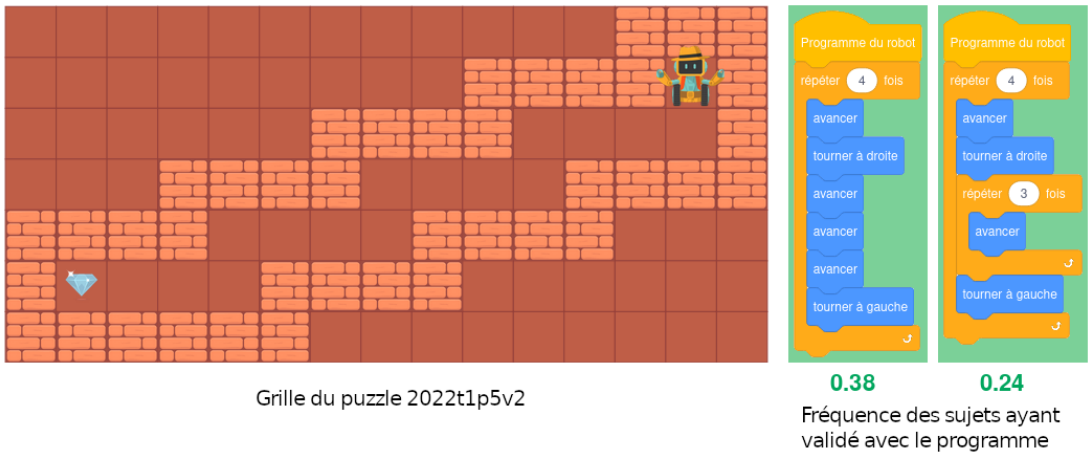


FIGURE 13.14 – Mobilisation spontanée d'une imbrication de boucles pour le puzzle 2022t1p5v2

la contrainte en nombre de blocs, lorsqu'ils ne trouvent pas le motif optimal (Timéo, 5ème, 2022t3p4v3).

Nous assistons à l'extension du schème d'usage du bloc *répéter*. Le sujet fait évoluer la règle de ce qu'il peut placer à l'intérieur du bloc, d'instructions simples à l'imbrication d'un bloc *répéter*.

En revanche, l'accès à la boucle imbriquée constitue un palier lorsque la boucle interne est elle-même une boucle avec plusieurs instructions. Par exemple, le programme le plus fréquent comprenant deux boucles imbriquées pour le puzzle 2022t3p4v2 a été soumis par seulement 2% des sujets à l'échelle des classes (Figure 13.15), à comparer avec les 24% pour le puzzle 2022t1p5v2 de la figure 13.14.

Une perspective est d'étudier plus précisément ce passage de la boucle sans imbrication aux boucles imbriquées en analysant les confrontations de la catégorie jaune du concours.

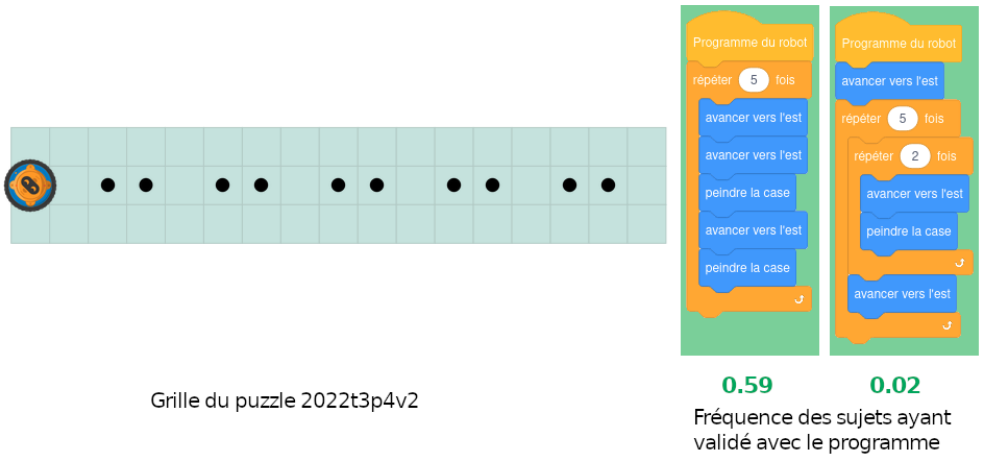


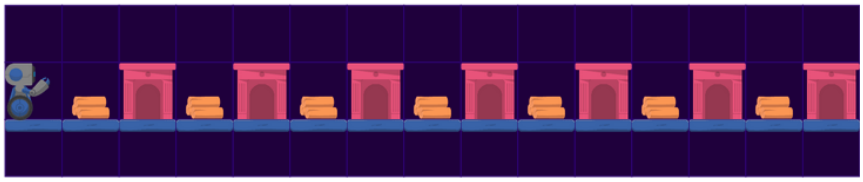
FIGURE 13.15 – Mobilisation spontanée d'une imbrication de boucles pour le puzzle 2022t3p4v2

13.5.2 Extension vers la programmation de procédures

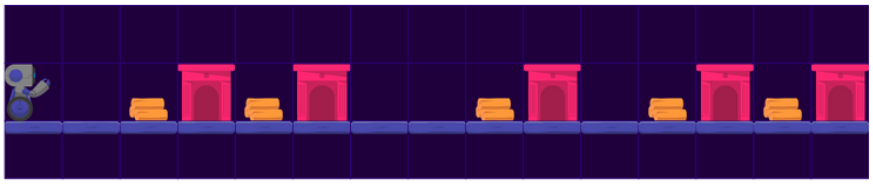
Nous avons un exemple de passage spontané à une procédure lors de l'expérimentation complémentaire (Timéo, 5ème, 2022t4p4v1). Bien que le puzzle puisse être résolu avec une boucle comme pour tous les puzzles que nous avons étudiés, Timéo isole le motif algorithmique dans un bloc de procédure. Il appelle ensuite cette fonction dans le bloc *répéter*.

La classe de problèmes qui impliquent la notion de procédure est plus large que celle qui implique la notion de boucle bornée. La rupture se situe au niveau de la distribution des motifs visuels sur la grille. Lorsque les motifs sont consécutifs, une boucle bornée peut être utilisée. Ce n'est plus le cas lorsque la distribution des motifs sur la grille devient irrégulière. Dans ce cas, le schème de mise en correspondance du motif visuel et du motif algorithmique, ou son alternative d'identification de la régularité dans le programme déjà édité restent opérants, mais ils doivent être associés à l'usage d'un bloc de procédure et non plus à l'usage d'un bloc *répéter*.

Pour approfondir l'étude de la mobilisation de l'identification de motif pour la programmation d'une procédure, il faudrait étudier les puzzles de la catégorie jaune visant l'introduction des fonctions (Figure 13.16). Ces investigations dépassent le cadre de cette recherche doctorale et constituent une perspective de travail.



Catégorie blanche, 2022t2p4v2 : les motifs sont consécutifs, le puzzle peut être résolu avec une boucle.



Catégorie jaune, 2021t2p6v1 : les motifs ne sont pas consécutifs, le puzzle doit être résolu avec une procédure.

FIGURE 13.16 – Rupture dans la répartition des motifs sur la grille pour les puzzles impliquant une boucle bornée vs une procédure

13.5.3 Extension vers la programmation d'une structure conditionnelle imbriquée dans une boucle

Lors de l'expérimentation complémentaire, nous remarquons la mobilisation spontanée d'une structure conditionnelle imbriquée dans une boucle. Les sujets ont déjà été confrontés aux blocs de structure conditionnelle dans les problèmes p6 de la catégorie blanche pendant le concours Algoréa, problèmes qui ne sont pas étudiés dans le cadre de la présente étude.

Le puzzle 2022t4p1v2 est particulièrement intéressant, car il est équivalent au puzzle 2022t3p4v2. Lors de l'expérimentation complémentaire, quelques sujets tels que Noé, changent de stratégie entre ces deux puzzles. Ils mobilisent une structure conditionnelle imbriquée dans une boucle pour le puzzle 2022t4p1v2 (Noé, 3ème, 2022t4p1v2), alors qu'ils avaient mobilisé une boucle voire une boucle imbriquée pour le puzzle 2022t3p4v2 (Noé, 3ème, 2022t3p4v2).

Ces sujets reviennent à la case comme unité, en traitant la variation d'état de cette case par une structure conditionnelle, plutôt que de s'appuyer sur la disposition régulière des éléments sur la grille. Cette stratégie est plutôt mise en œuvre par des experts, ce qui nous questionne sur la pertinence d'imposer l'identification de motif dans les situations de la classe 2 par la non mise à disposition des blocs de structure conditionnelle. La mise à disposition de ces blocs entraîne un changement de classe de problèmes pour ces situations : classe de problèmes pour lesquelles le traitement mobilise à la fois une répétition et

une structure conditionnelle. En revanche, cette stratégie n'est pas pertinente pour les puzzles de la classe 3.

Mettre à disposition les blocs de structure conditionnelle pour résoudre des situations où les motifs sont disposés linéairement entraînerait un déplacement du palier de difficulté du passage de une à plusieurs cases pour l'identification du motif. Des expérimentations seraient nécessaires afin d'étudier les parcours d'apprentissage selon le choix effectué dans l'ordre d'exposition aux situations.

13.5.4 Synthèse

Ces quelques confrontations nous laissent entrevoir comment, en s'adaptant aux blocs de langage mis à disposition et au nombre de blocs disponibles, le sujet construit un système de schèmes qui s'étend au-delà de ce que nous avons pu analyser dans cette thèse. Ces observations sont cohérentes avec le champ conceptuel de l'analyse a priori, délimité pour le langage Scratch (11.1.3). Dans ce champ conceptuel, élaborer le motif algorithmique occupe une place centrale, que nous retrouvons dans ces quelques confrontations.

Nous proposons l'ébauche d'une modélisation du système de schèmes qui est construit au fur et à mesure de la complexification des situations de programmation (Figure 13.17).

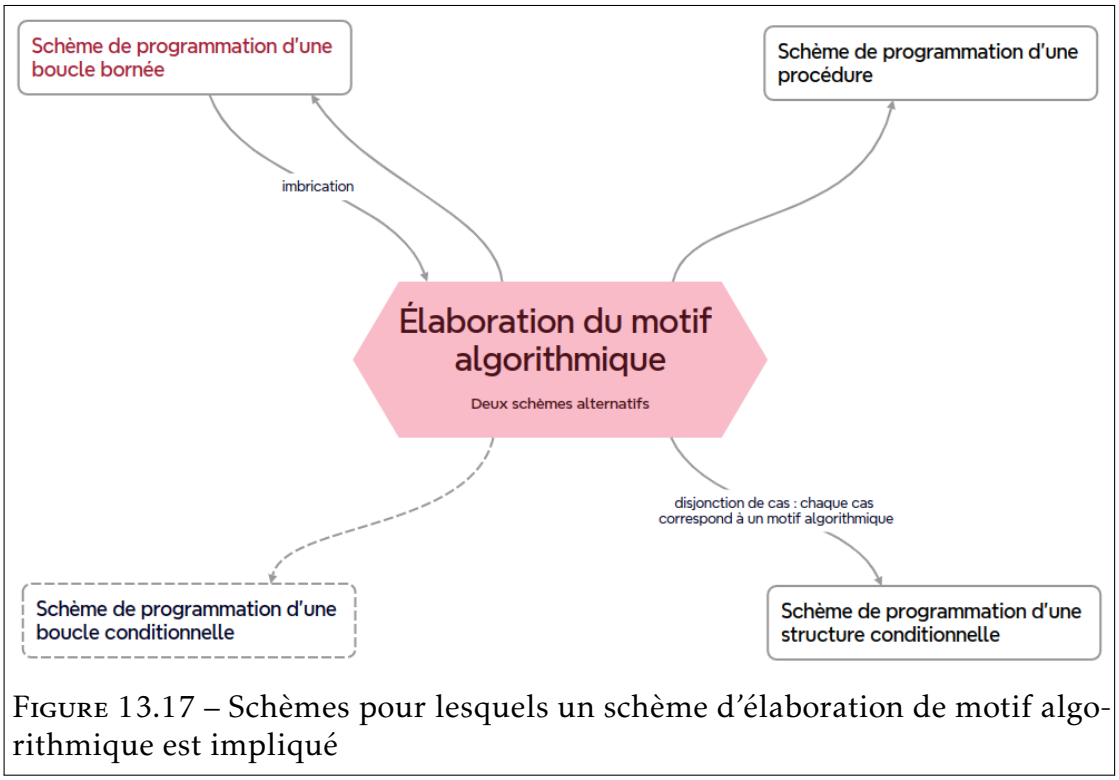


FIGURE 13.17 – Schèmes pour lesquels un schème d'élaboration de motif algorithmique est impliqué

Une flèche reliant deux schèmes signifie que le schème à l'arrivée de la flèche inclut celui au départ de la flèche comme composante. En plus des schèmes observés sur des extraits vidéo, nous avons ajouté la boucle conditionnelle par déduction en pointillés, bien que nous n'ayons pas eu l'occasion d'analyser de situation impliquant cette notion.

Cette modélisation ne montre pas les schèmes qui sont articulés avec ceux d'élaboration du motif algorithmique. Analyser finement l'activité du sujet dans ces situations constitue une perspective de recherche à court terme. Nous projetons d'utiliser la même méthodologie. Nous disposons des données à l'échelle nationale et à l'échelle du sujet. Cependant, nous disposons d'un volume de données moins conséquent à l'échelle des classes pour les notions plus avancées telles que les procédures. Une nouvelle collecte serait nécessaire à cette échelle afin de pouvoir mener des analyses quantitatives.

Le système de schèmes construit n'implique pas explicitement la notion de variable. C'est une différence notable avec les champs conceptuels relatifs à l'initiation à la programmation relevés dans la littérature (El Rouadi, 1999; Rogalski & Vergnaud, 1987). Nous sommes en présence d'une transposition didactique visant à simplifier les premiers pas en programmation. Dans ce contexte, la notion de variable appartient au niveau conceptuel supérieur. L'étude de son introduction et de l'évolution induite du système de schèmes constitue une autre perspective de recherche.

13.6 Place de l'exécution du programme dans l'activité du sujet

Cette section et la suivante complètent notre travail sur la conceptualisation-en-acte en abordant deux points plus transversaux.

Le premier point, abordé dans cette section, est l'exécution du programme. Elle est centrale dans l'initiation à l'informatique dans le sens où elle porte le passage du *faire* au *faire faire*, c'est-à-dire l'automatisation du traitement de l'information constitutif de l'informatique. Que l'exécution soit effective ou simulée avec le curseur de souris, elle prend une large place dans l'activité du sujet pour les situations de programmation que nous étudions. Nous avons déjà abordé l'exécution du programme comme composante de certains schèmes identifiés. Nous apportons quelques éléments supplémentaires concernant la manière dont intervient l'exécution dans l'activité du sujet.

13.6.1 Plusieurs modalités pour exécuter le programme

Nous relevons plusieurs modalités pour exécuter un programme édité, qui sont à mettre en regard d'une part avec la manière dont le sujet construit la fonctionnalité d'exécution comme instrument, et d'autre part avec sa capacité à se représenter mentalement cette exécution.

Exécutions avec le bouton *play*

Une première façon d'exécuter le programme est de le lancer avec le bouton *play*. Dans ce cas, l'exécution peut constituer une délégation à la machine de la validation du programme (Rogalski, 1985). Il se peut que le sujet n'en ait alors pas une représentation précise. Il raisonne à partir du feedback renvoyé par le système à l'issue de l'exécution, notamment la position et l'orientation du robot virtuel sur la grille.

Lors de sa confrontation avec le puzzle 2022t1p5v1, Robin verbalise le fait qu'il délègue la validation de son programme à la machine. La stratégie se rapproche de l'essai-erreur. Cependant, le feedback du système à l'issue de l'exécution oriente la suite de ses actions, sans qu'il ait une représentation claire de ce qu'il est en train de faire (Robin, CM1, 2022t1p5v1).

Exécution en mode pas à pas

Une autre possibilité est l'exécution en mode *pas à pas*. Le sujet délègue l'exécution du programme à la machine, mais il en contrôle les étapes. Il peut ainsi vérifier que l'exécution effective est conforme à ce qu'il a anticipé. Ce mode d'exécution est de nature à aider le sujet à repérer ses erreurs.

Par exemple, dans la confrontation de Mickaël avec le puzzle 2022t3p5v1, le mode pas à pas lui permet de corriger deux erreurs, l'une après l'autre. L'utilisation experte du mode pas à pas compense la représentation mentale de l'exécution qui n'est pas efficace (Mickaël, 4ème, 2022t3p5v1).

Simulation de l'exécution avec le curseur de souris

Une troisième possibilité est de simuler l'exécution avec le curseur de souris. Dans ce cas, il ne s'agit pas d'une exécution effective, mais le curseur de souris joue le rôle d'un robot pion. L'exécution effective du programme par le système n'intervient alors que comme demande de validation du puzzle (Élana, CE2, 2021t2p5v1). Les mouvements de souris révèlent la simulation mentale de l'exécution.

Simulation mentale de l'exécution sans mouvements de souris associés

Probablement que certains sujets simulent complètement mentalement l'exécution, et nous n'avons en conséquence pas accès à leur procédure. L'enregistrement des mouvements de souris nous permet de repérer automatiquement ces confrontations sans phase de survol de la grille (Figure 13.18), comme celle de Mickaël avec le puzzle 2022t4p1v4 (Mickaël, 4ème, 2022t4p1v4).

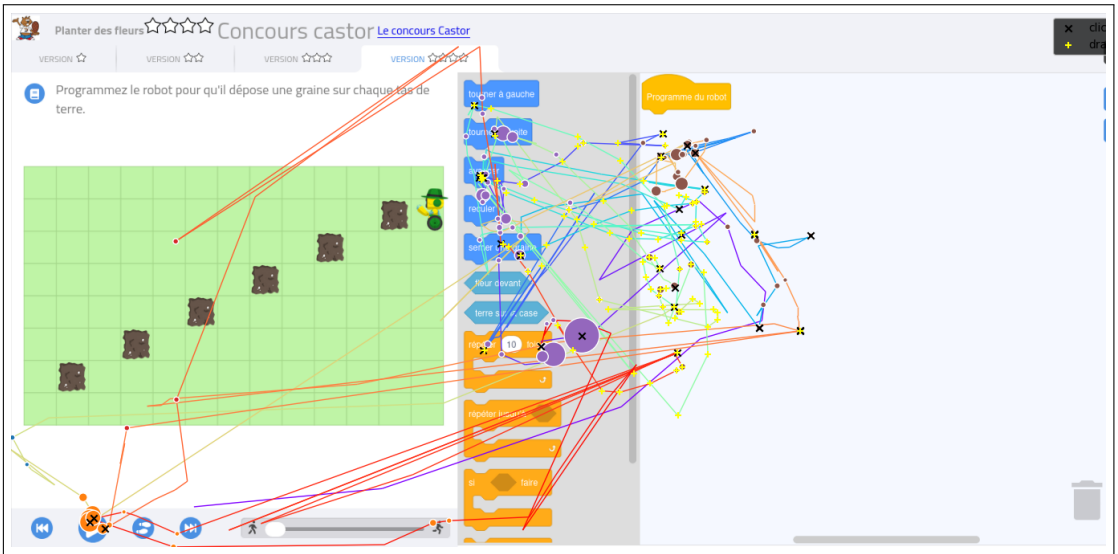


FIGURE 13.18 – Confrontation sans survol de la grille

Le degré d'instrumentalisation des fonctionnalités d'exécution de l'EIAH par le sujet diffère selon les différentes modalités. Plusieurs stratégies peuvent co-exister chez un même sujet, notamment selon son degré d'expertise et la complexité du puzzle. Nous détaillons dans les prochaines sections quelques points saillants par rapport à l'activité du sujet relative à l'exécution du programme.

13.6.2 Exécution comme révélateur de la difficulté à se représenter la position du robot à l'issue de l'exécution d'une boucle

Dans la confrontation d'Alix avec le puzzle 2022t2p5v2, nous voyons nettement une erreur de représentation de la position du robot après l'exécution de la boucle (Alix, CM2, 2022t2p5v2). L'erreur est cohérente avec le pointage lors du dénombrement. Le fait qu'Alix détache les blocs après la boucle montre qu'elle utilise l'exécution comme procédure de vérification du positionnement

du robot. Cependant trois vérifications sont nécessaires pour faire évoluer la représentation d'Alix.

Nous observons également cette stratégie de prise d'information de la position du robot après l'exécution de la boucle chez Léa. Elle enlève l'instruction placée après la boucle avant de lancer une exécution (Léa, CE1, 2021t3p4v3).

Le besoin d'exécuter le programme révèle un repérage approximatif des limites du motif visuel et de sa correspondance avec le motif algorithmique.

13.6.3 Exécution de programmes partiels et stratégie de décomposition du problème

L'exécution de programmes partiels nous renseigne sur la stratégie de décomposition du problème comme nous l'avons montré dans la section 12.5.3 du chapitre 12. Nous retrouvons ce résultat pour le petit nombre de puzzles des classes 2 et 3 qui nécessitent des instructions hors de la boucle ou plusieurs boucles en séquence. Par exemple, le programme partiel constitué de la seule boucle est le programme le plus fréquemment exécuté pour le puzzle 2022t2p5v2 (35% des sujets).

L'exécution de programmes partiels nous renseigne aussi sur le degré de complexité de la situation pour lequel le sujet est capable à un moment donné de se représenter entièrement l'exécution. Pour illustrer notre propos, nous montrons les programmes exécutés par Louna pour les puzzles 2022t1p3v3 et 2022t1p5v3 (Figure 13.19). Comme la frise du début de la session le montre, Louna résout tous les puzzles avec un profil expert jusqu'au puzzle 2022t1p5v3. Pour le puzzle 2022t1p3v3 qui n'implique que des boucles avec une instruction (classe de situation 1), Louna ne lance qu'une exécution pour valider, la résolution est experte. La décomposition est supportée par la simulation mentale de l'exécution avec le curseur de souris (Louna, 6ème, 2022t1p3v3). En revanche, pour le puzzle 2022t1p5v3 lors de la même session, donc quelques minutes plus tard, Louna adopte une stratégie de décomposition du problème, et exécute un programme partiel après chaque édition de boucle (Louna, 6ème, 2022t1p5v3). Dans ce dernier cas, Louna vérifie la position du robot à l'issue de la boucle avant de continuer, révélant une représentation de l'exécution non encore experte de la position du robot à l'issue d'une boucle pour les situations de la classe 3, mais aussi une instrumentalisation de la fonctionnalité d'exécution au service de l'avancée de sa résolution de la situation.

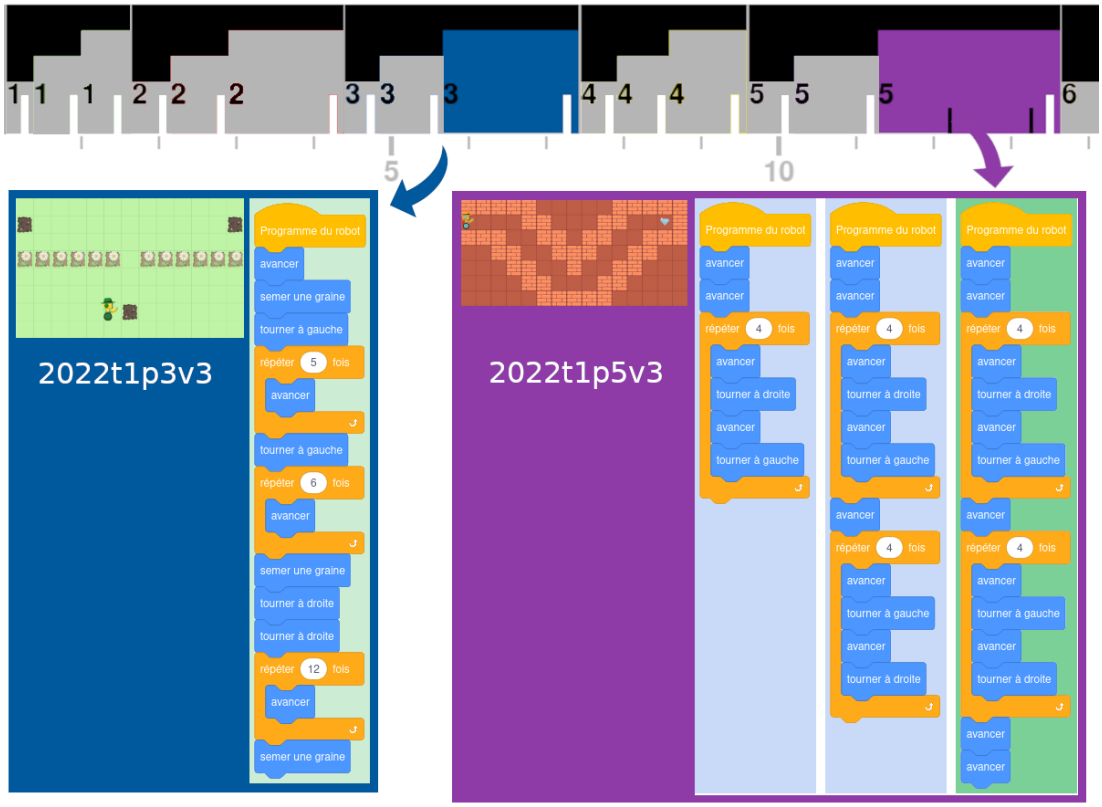


FIGURE 13.19 – Résolution des puzzles 2022t1p3v3 et 2022t1p5v3 par Louna, 6ème

13.6.4 Schème de complétion associé à la décomposition du problème

Nous repérons un schème de complétion, dont le but est de compléter le programme, et qui peut se définir comme suit : lancer une exécution afin de repérer la position du robot à l'issue de celle-ci, prendre l'information de cette position sur la grille, placer à la suite de ceux déjà dans le programme les blocs correspondant aux actions restant à faire faire au robot. Dans sa stratégie de décomposition, le sujet considère en fait une nouvelle situation, qui est de programmer le robot à partir de sa position à l'issue de l'exécution, en ne s'occupant plus de ce qui a été programmé auparavant. Cette stratégie conduit souvent, lorsque la contrainte en nombre de blocs le permet, à des solutions non optimales, comme dans le cas de Léa (Léa, CE1, 2021t3p4v3).

L'exécution est alors construite comme instrument de la prise d'information. Le théorème-en-acte tenu pour vrai est « le robot a exécuté tout le programme

déjà dans l'éditeur quand l'exécution s'arrête ». Ce schème est efficace lorsque le programme dans l'éditeur correspond à un programme partiel correct. Mais lorsque le programme comporte une erreur et que l'exécution s'interrompt à l'endroit de cette erreur, ce théorème n'est plus vrai, et induit le sujet en erreur. Il repart de la position du robot pour continuer le programme, alors qu'elle ne correspond pas à la position de celui-ci à la fin de l'exécution du programme qui se trouve dans l'éditeur.

Le schème de complétion est mobilisé, notamment pour compléter le programme avec les instructions après une boucle. Il constitue une stratégie pour contourner la difficulté de se représenter la position et l'orientation du robot à l'issue de l'exécution de la boucle.

13.6.5 Synthèse

Les analyses de cette section montrent toute la complexité du statut de l'exécution du programme, entre délégation complète à la machine et simulation mentale intégrale. La construction d'une représentation de l'exécution correcte est nécessaire, et se traduit par une simulation mentale. Le sujet étaie cette simulation en instrumentant les fonctionnalités de l'EIAH, et en les intégrant dans son processus de résolution du problème. Rogalski parle de *communication opérative* entre le sujet et le dispositif informatique (Rogalski, 1988b).

Lorsque le sujet fait fonctionner les éléments d'un système, il crée du sens (Rogalski, 1987). Cependant, Rogalski met en garde sur le fait que les savoirs opérationnels construits de cette manière sont locaux et prennent beaucoup de temps.

Ces résultats nous amènent à questionner notre calcul de profils du chapitre 9. Les résolutions qui ont été calculées comme expertes par ce calcul de profils sont-elles les seules à prendre en considération ? Certaines exécutions de programme ne pourraient-elles pas être considérées comme une règle d'action, dans une stratégie experte de décomposition du problème ?

Par exemple, le profil calculé est *ajustement* pour la confrontation de Louna avec le puzzle 2022t1p5v3 (13.6.3). Il ne s'agit cependant pas d'un ajustement par rapport à une erreur, mais plutôt d'un ajustement de la stratégie en rapport avec la capacité du sujet à traiter la complexité de la situation. Ce sont deux ajustements de nature différente dont la distinction n'est pas capturée par notre modélisation, ce qui en constitue une limite. Une perspective est d'affiner cette modélisation en prenant en compte la nature du programme soumis (programme partiel vs programme erroné) dans le calcul des profils.

13.7 Étayage et obstacles liés à l'EIAH

Dans cette section, nous cherchons à appréhender l'influence de l'EIAH dans la construction des schèmes mobilisés pour la programmation d'une boucle. Nous discutons l'étayage qu'apporte au sujet certaines fonctionnalités de l'EIAH, et les limites que constituent certains paramétrages pour nos analyses. Bien que nous serions en mesure d'analyser une pluralité de fonctionnalités, nous nous limitons à l'étude de la limite en nombre de blocs, de la restriction de la nature des blocs disponibles, et à l'analyse du mode d'exécution pas à pas qui sont directement en lien avec la présente recherche.

Pour comparer l'effet de la limite de blocs et de la mise à disposition d'un panel de blocs, nous nous appuyons sur des comparaisons entre des puzzles du concours Algoréa et des puzzles équivalents de l'expérimentation complémentaire : 2022t4p1v2 et 2022t3p4v3, 2022t4p3v2 et 2021t1p4v3, 2022t4p1v4 et 2021t2p5v2. En effet, nous avons modifié ces paramètres pour l'expérimentation complémentaire : nombre de blocs illimité, palette de blocs plus large.

13.7.1 Limite en nombre de blocs disponibles pour éditer le programme

Nous avons montré dans le chapitre 12 que pour les quelques puzzles où la boucle n'est pas contrainte, les sujets pour lesquels le schème d'usage du bloc *répéter* n'est pas encore construit, ou de manière non robuste valident le puzzle 2022t1p4v1 (classe 1) avec une séquence. Cela représente globalement 5% des sujets à l'échelle des classes.

Mais pour l'essentiel des puzzles du concours Algoréa, le sujet dispose d'un nombre limité de blocs pour concevoir son programme. Cette limite assure que lorsque le puzzle est validé, le sujet a bien utilisé une boucle. Pour certains puzzles, ce nombre limité de blocs contraint aussi l'identification d'un motif optimal. En revanche, lors de l'expérimentation complémentaire de l'été 2022, nous avons supprimé cette contrainte en nombre de blocs.

Tous les sujets qui ont validé les puzzles du concours Algoréa avec une boucle ont aussi validé ceux de l'expérimentation complémentaire de cette manière, à une exception près (régression due à un autre facteur, indépendant). Cependant, nous constatons quelques régressions vers un motif non optimal, comme celle d'Alix sur le puzzle 2022t4p1v2 (Alix, CM2, 2022t4p1v2) par rapport au puzzle 2022t3p4v3 (Alix, CM2, 2022t3p4v3).

Nous constatons que se heurter à la limitation du nombre de blocs, si elle déstabilise le sujet dans un premier temps, amène ensuite une évolution positive des procédures. Être confronté à cette limite conduit Robin à consulter le tutoriel,

ce qui l'amène à faire évoluer sa stratégie en plusieurs étapes, en se posant des questions sur le programme qui ne fonctionne pas : « Qu'est-ce qui ne va pas ? » (Robin, CM1, 2022t1p4v2). Robin recourt à nouveau à ce questionnement lors de l'atteinte de la limite de blocs lors du tour 3 pour le puzzle 2022t3p4v3 pour lequel deux éléments saillants sur des cases adjacentes ne font pas partie du même motif : « Le problème c'est qu'on a huit blocs autorisés.. comment est-ce qu'on peut faire ? ». Pour ce puzzle avec cette difficulté spécifique, il faut plus de dix minutes à Robin pour parvenir à une solution valide (sur l'extrait vidéo, les passages qui ne présentent pas d'intérêt ont été coupés) (Robin, CM1, 2022t3p4v3). Ce questionnement à propos de la stratégie à adopter est aussi présent chez Alix : « qu'est-ce que je peux faire ? » (Alix, CM2, 2022t3p4v3).

Pour plusieurs sujets, dont Timéo, elle entraîne un passage aux boucles imbriquées plutôt que l'identification du motif optimal attendu (Timéo, 5ème, 2022t1p4v3).

L'effet positif de contraindre le nombre de blocs est mis en évidence dans l'expérimentation de Polledo et al. (2021). Dans cette étude, les auteurs utilisent un jeu centré spécifiquement sur l'identification de motifs visuels, hors d'un contexte de programmation. Ils ont composé des groupes expérimentaux et des groupes témoins. La contrainte à laquelle sont soumis les groupes expérimentaux est sensiblement équivalente à la limite en nombre de blocs de l'environnement Algoréa. Ces groupes expérimentaux obtiennent de meilleurs résultats que ceux pour lesquels la limitation n'a pas été activée (Polledo et al., 2021).

Lorsque le nombre de blocs n'est pas contraint, Walgenwitz et al. ont étudié quel est l'effet du nombre de motifs visuels sur le passage spontané de la séquence à la boucle pour une situation de la classe 3 similaire à celle de notre recherche (Walgenwitz et al., 2024). Ils ont trouvé que le nombre de 10 motifs incite fortement les sujets à passer à la boucle, alors qu'une situation avec 4 motifs est traitée majoritairement avec une séquence alors même que le programme solution nécessite déjà 26 blocs.

Cependant, la contrainte en nombre de blocs peut amener à des blocages et des abandons. Elle entrave la stratégie spontanée du sujet : boucle déjà utilisée en séquence dans le programme (Léa, CE1, 2021t1p4v3), solution qui résout la mission mais n'est pas validée par le système à cause d'un motif non optimal (Sven, CM2, 2021t1p4v3). Dans les cas mentionnés, le sujet n'est pas en mesure de procéder à une adaptation et quitte le puzzle.

Certains sujets verbalisent à propos de la contrainte de blocs au moment du blocage, ce qui révèle que c'est à ce niveau qu'ils situent leur difficulté : « Y'en a que huit qui sont autorisés » (Léa, CE2, 2022t3p4v3), « Expérimentatrice-Alors.. qu'est-ce qui est embêtant ? ; Inès- Euh.. j'ai voulu plus de blocs semer une graine »(Inès, CE1, 2022t1p4v2), « On peut pas faire le programme sans

inclure le premier et le dernier avec le reste.. mais.. ça prend trop de blocs » (Mickaël, 4ème, 2022t3p4v3), parfois avec une marque d'agacement « Argh j'ai pas assez de blocs » (Sam, 6ème, 2022t1p4v3). Dans certains cas, un étayage de l'expérimentatrice, qui incite à continuer le programme au-delà de la limite autorisée, a permis de surmonter la difficulté.

Nous en concluons que la limitation en nombre de blocs favorise l'évolution des schèmes du sujet. Plus précisément, elle déplace le but de la tâche de « faire faire la mission au robot » à « économiser des blocs pour pouvoir faire faire la mission au robot ». Cependant, pour favoriser le processus d'accommodation et éviter les blocages, permettre l'exécution d'un programme trop long, mais sans y associer la validation du puzzle, serait facilitateur.

13.7.2 Limitation de la nature des blocs disponibles

Pour notre recherche, la mise à disposition d'un nombre réduit de blocs de nature différente, notamment le fait que le bloc *répéter* soit le seul disponible parmi les structures de contrôle, nous a semblé pertinent dans l'objectif de concentrer nos investigations sur le concept de motif. En effet, la mise à disposition d'une palette réduite de blocs permet de limiter l'espace de recherche pour les débutants, pour éviter qu'ils se perdent.

La comparaison des confrontations de Colin avec les puzzles 2022t3p4v3 et 2022t4p1v2 qui sont équivalents donne un aperçu de l'étayage qu'apporte la limitation de la palette des blocs (sur fond gris sur la figure 13.20). Colin passe 13 minutes et 26 essais sur le puzzle 2022t4p1v2, alors qu'il a résolu le puzzle 2022t3p4v3 en 1'40 et 3 essais quelques mois auparavant. Pour le puzzle 2022t4p1v2, Colin conçoit des programmes trop complexes comme celui de la figure 13.20, dans lequel on retrouve trois des quatre blocs de structure de contrôle disponibles.

Cependant, cette limitation peut contrarier une procédure experte déjà disponible. Par exemple, lors des puzzles du parcours Algoréa, certains sujets réclament le bloc *si*, signe qu'ils placent le puzzle dans une autre classe de problèmes : « Moi avant.. [..] j'avais le bloc si faire » (Léa, CE1, 2021t2p4v3), « Déjà y'a pas le bloc si.. le truc si.. » (Lou, 6ème, 2022t3p4v3).

Limiter la palette des blocs laisse peu de possibilités au sujet pour la classe de problèmes dans laquelle placer le puzzle, ce qui constitue dans certains cas un étayage et dans d'autres cas une entrave. Au-delà de notre recherche focalisée sur le concept de motif, il serait intéressant de creuser les conséquences de limiter la nature des blocs disponibles sur la construction du champ conceptuel par le sujet. Des expérimentations pourraient être menées en variant la palette des blocs disponibles pour un même puzzle, ou en variant le moment où cette limitation est supprimée.

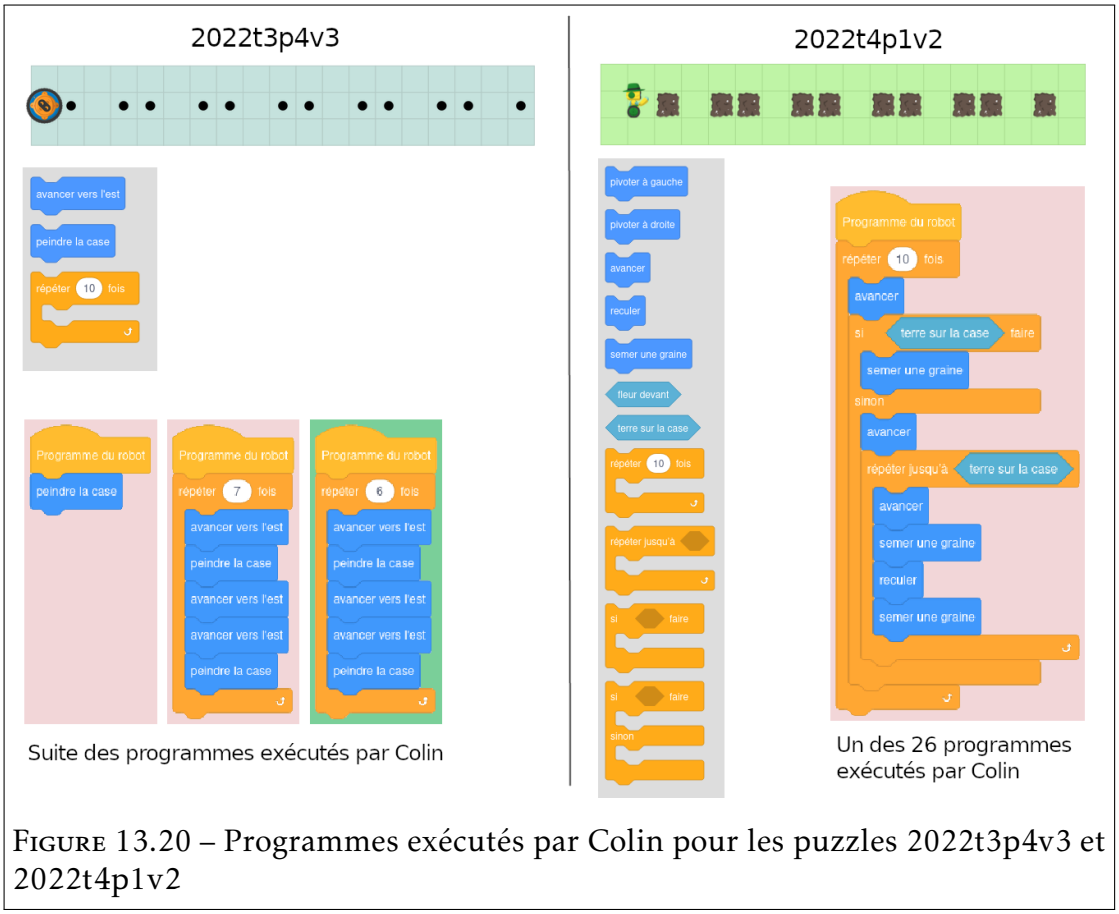


FIGURE 13.20 – Programmes exécutés par Colin pour les puzzles 2022t3p4v3 et 2022t4p1v2

13.7.3 Mode pas à pas

La dernière fonctionnalité que nous analysons est le mode pas à pas. Nous avons montré dans la section 13.6 qu'il constitue une modalité d'exécution du programme qui permet d'en contrôler le déroulé. Le mode pas à pas se veut une aide pour repérer et corriger les erreurs. Cela dit, y avoir recours n'est pas spontané. À l'échelle du sujet, nous avons largement incité à l'utiliser. Il nécessite d'être constitué comme instrument. Comprendre que le bloc surligné est le prochain qui va être exécuté n'a notamment rien d'évident pour les débutants.

Par ailleurs, nous identifions quelques limites quant à son utilisation. Une limite concerne la non disponibilité du mode pas à pas lorsque le sujet a dépassé le nombre de blocs autorisé pour éditer son programme. En effet, ce moment a été identifié précédemment comme provoquant des blocages, le sujet ne parvenant pas à synthétiser l'écriture d'un programme en séquence, ni à le corriger. Par exemple, une erreur dans le sens de pivotement empêche Alexis d'aboutir en identifiant le motif algorithmique dans son programme pour en synthétiser

l'écriture (Alexis, CM2, 2021t1p5v1) . Le mode pas à pas est indisponible à un moment où il serait particulièrement utile, poussant souvent le sujet à basculer sur une stratégie par essai-erreur. Cette limite de l'instrument entraîne une régression dans la stratégie du sujet.

Une autre limite concerne la stratégie dans le repérage de certaines erreurs. La confrontation de Sam avec le puzzle en est un exemple représentatif (Sam, 6ème, 2022t2p4v3). Sam poursuit l'exécution avec le mode pas à pas tout le temps que le robot effectue les actions attendues. La stratégie est efficace pour identifier l'erreur mais pas pour la corriger. La stratégie de débogage avec le mode pas à pas est efficace pour une séquence, mais elle est mise en échec pour certaines erreurs liées à la boucle.

13.7.4 Éléments de discussion

Ce bref aperçu montre qu'une même fonctionnalité ou un même paramétrage de l'EIAH peut constituer un étayage à certains moments de l'apprentissage, et un obstacle à d'autres. Le rôle du concepteur de puzzles ou de parcours est donc loin d'être neutre. À travers le paramétrage de l'EIAH, il contraint plus ou moins l'activité du sujet. Rabardel qualifie l'activité de « relativement requise » lorsque l'instrument impose une certaine structuration de l'action (Rabardel, 1995, p.5).

Nous pouvons aussi considérer que la conception de puzzles de programmation et les paramétrages associés constituent une contribution à l'élaboration de situations didactiques (Brousseau, 2011). Cette contribution occupe une place centrale dans le contexte du concours où l'enseignant se met en retrait. Dans un contexte classique d'enseignement, à la fois le concepteur des puzzles, ou ingénieur pédagogique, et à la fois l'enseignant participent à la conception de la situation didactique, l'enseignant ayant un rôle de décision dans les modalités d'usage de l'EIAH dans le contexte de sa classe. Si pour l'élève, l'EIAH est un environnement d'apprentissage, il est nécessaire que l'enseignant construise cet EIAH comme un instrument pour enseigner, avec des questions telles que : quelle finesse de compréhension de sa part des ressorts de chaque puzzle, de la mise en parcours, de l'activation de certains paramétrages sur l'activité du sujet ? Que délègue l'enseignant à l'EIAH, et donc à l'ingénieur pédagogique ? Quelle est sa marge de manœuvre dans le paramétrage de celui-ci ? Le rôle de l'ingénieur pédagogique, qui agit à travers l'EIAH sur la situation didactique, n'est pas envisagé à notre connaissance dans les travaux de Brousseau, en particulier dans l'étude de la responsabilité didactique (Brousseau, 1997). Selon nous, l'étude de la répartition entre ingénieur pédagogique et enseignant dans l'élaboration des situations didactiques constitue une perspective de recherche intéressante.

13.8 Synthèse

Ce chapitre a permis d'affiner le système hiérarchique de schèmes ébauché dans le chapitre précédent à propos de la programmation d'une boucle bornée (Figure 13.13). Dans ce système hiérarchique, le concept de motif tel que nous l'avons défini occupe une place centrale. Nous avons montré, par l'étude des mouvements de souris, l'importance des phases de prise d'information sur la grille dans les composantes des schèmes identifiés.

La décomposition du schème a permis de caractériser les ruptures de situation entre les classes de situations 1, 2 et 3 repérées par l'analyse des fréquences de réussite à l'échelle nationale. Cette analyse à une granularité plus fine a permis de rectifier la variable de situation impliquée dans la rupture de traitement entre les classes de situations 2 et 3. Ainsi, le passage de la classe 1 à la classe 2 est marqué d'une part par l'élaboration du motif algorithmique, qui implique soit une mise en correspondance avec le motif visuel, soit une identification de la régularité dans le programme déjà édité, et d'autre part par le changement dans l'unité à considérer pour le dénombrement des motifs. Le passage de la classe 2 à la classe 3 est pour sa part caractérisé par la nécessité de repositionner le robot pour aborder l'itération suivante, alors qu'il se trouve déjà en position adéquate après la dernière action dans les situations de la classe 2. Dans notre contexte, cette différence dans le traitement prend la forme d'un schème qui vise à ajuster ou vérifier la jonction entre deux itérations successives.

Nous retrouvons les sous-tâches relatives à la programmation d'une boucle, déjà identifiées par plusieurs auteurs. Rogalski et Samurçay caractérisent la planification répétitive du traitement, l'identification du statut fonctionnel du contrôle d'arrêt et la détermination de l'état initial dans lequel se trouvent ou doivent se trouver les variables qui sont transformées dans l'invariant de boucle (Rogalski & Samurçay, 1986). Notre apport concerne l'identification de ces sous-tâches pour les situations de programmation d'un robot virtuel sur une grille. Dans notre contexte, la planification répétitive du traitement correspond à l'identification du motif algorithmique à partir du motif visuel. Le dénombrement des motifs est un cas particulier de l'identification du statut fonctionnel du contrôle d'arrêt (cas de la boucle bornée). La détermination de l'état initial correspond au repérage de la position du robot juste avant l'exécution de la boucle. Nous avons montré que, pour ces situations proposées à des débutants en programmation de 7 à 15 ans, l'ordre des sous-tâches le plus souvent observé diffère de celui rapporté par Rogalski et Samurçay (1986).

Si nous confrontons aux travaux de Jolivet et al., 2023, nous remarquons que la détermination de l'état initial n'apparaît pas dans le référentiel proposé par ces auteurs. Pour notre part, nous considérons que la phase de positionnement

du robot à l'entrée de la boucle, et surtout la prise d'information associée à cet état initial, fait intégralement partie de la programmation de celle-ci.

Nous avons ouvert la poursuite de cette étude vers d'autres notions pour lesquelles le schème d'élaboration de motif algorithmique que nous avons caractérisé est remobilisé par le sujet, en articulation avec d'autres qui restent à investiguer. Cette ébauche permet d'entrevoir une première extension du domaine de validité des schèmes construits lors de ces situations.

Nous avons enfin montré la place importante de la simulation de l'exécution avec le curseur de souris et des phases de prise d'information sur la grille. En conséquence, une question qui peut être posée est : comment favoriser ces phases d'analyse et de vérification par rapport à une délégation de l'évaluation de la validité du programme au feedback du système ? Une expérimentation a été menée dans ce sens par Chevalier et al. dans le contexte de la robotique pédagogique (Chevalier et al., 2020). Les auteurs ont montré que bloquer la possibilité d'exécuter le programme pendant une certaine durée amènent des élèves de 9-10 ans à investir de manière plus équilibrée les différentes phases du processus cognitif qu'ils ont identifiées : compréhension du problème, génération d'idées, formulation du comportement attendu du robot, évaluation de la solution exécutée. Une perspective serait de mener une expérimentation similaire pour la programmation d'un robot virtuel. Nous serions en mesure de paramétrer l'EIAH dans ce sens et d'investiguer avec notre protocole de collecte et d'analyse de traces si cette modification génère plus de phases de prises d'information et de simulation de l'exécution, et de comparer les profils de confrontation dans les deux groupes.

Conclusion générale

Sommaire du présent chapitre

La présente thèse constitue une étape importante dans un processus de recherche débuté depuis sept ans avec les études de cas du dispositif Chticode (Léonard et al., 2019). La recherche doctorale en elle-même a donné lieu à quatre publications (Léonard, 2024 ; Léonard et al., 2023 ; Léonard, Peter, Secq & Fluckiger, 2022 ; Léonard, Secq et al., 2022). Nous avons adopté une approche didactique dans le sens où nous avons donné une place prépondérante au contenu, et investigué la manière dont ce contenu, dans sa spécificité, est appréhendé par des sujets (Fluckiger, 2019). Or une spécificité de la programmation en tant que contenu est d'être abordée à travers l'utilisation de la technologie informatique, en l'occurrence un environnement de programmation. Nos analyses abordent cette spécificité à travers une approche instrumentale qui rend compte de la constitution de cet objet technologique comme instrument.

Dans une première section, nous rappelons les principaux résultats obtenus. Du point de vue méthodologique, nous avons proposé un dispositif qui articule mobilisation de concepts didactiques et analyse de traces. Nous en mettons en évidence les principales contributions dans une deuxième section. Enfin, cette thèse constitue un point d'appui pour des travaux à venir. Elle nous a amené à collecter un volume de données conséquent, dont seulement une partie a pu être exploitée dans le temps imparti. Elle a aussi permis de construire un outil de traitement et de visualisation de données qui prend la forme d'une bibliothèque Python, et qui est largement remobilisable. Nous présentons les perspectives de recherche que cette thèse ouvre à court et plus long termes.

Synthèse des résultats obtenus

Pour débuter notre conclusion, nous rappelons la problématique de cette thèse. Elle porte sur les genèses conceptuelles en lien avec le concept de motif lors de l'initiation à la programmation dans un environnement de programmation par blocs, en particulier lors de la résolution de puzzles qui requièrent l'utilisation d'une boucle bornée.

Nous passons en revue les questions de recherche que nous avons traitées et y associons les réponses que nous proposons.

Approche instrumentale

Nous avons commencé par aborder l'aspect instrumental de l'activité de programmation. Notre question de recherche sur cet aspect portait sur les schèmes d'usage attachés à l'environnement de programmation par blocs, schèmes d'usage qu'il est nécessaire de construire afin d'être en mesure de résoudre les puzzles proposés.

Nous avons réalisé une étude instrumentale de l'EIAH que nous avons utilisé. Nous avons montré que les composantes artéfactuelles de l'EIAH sont imbriquées les unes dans les autres. Lors de la conception d'un programme, nous avons établi une structure hiérarchique de schèmes avec quatre niveaux. Nous avons montré que les schèmes des niveaux hiérarchiques supérieurs, schèmes d'usage du langage de programmation et schèmes d'action instrumentée des boutons de contrôle de l'exécution sont intimement liés à la construction des premières conceptualisations-en-acte des notions de base en programmation. Notamment, les premiers apprentissages de la boucle sont intimement liés au schème d'usage du bloc *répéter*.

Construction d'indicateurs

Concernant l'approche didactique, notre première question de recherche avait pour objectif d'explorer la possibilité de définir des indicateurs qui nous permettent, automatiquement à partir des traces d'interaction, de rendre compte de l'expertise du sujet dans la résolution des puzzles de programmation en langage par blocs. Nous avons choisi de fonder ces indicateurs sur des éléments de la théorie des champs conceptuels, afin de pouvoir ancrer nos analyses dans ce cadre théorique établi pour l'étude de l'apprentissage des compétences complexes, dont la programmation informatique fait partie.

Nous avons effectivement défini une combinaison d'indicateurs qui caractérisent la confrontation d'un sujet avec un puzzle de programmation en langage par blocs. Cette combinaison est constituée de la validité du programme exécuté, du nombre d'essais et d'un indicateur de rapidité normalisée indépendant de la longueur de la solution de référence et de la dextérité du sujet dans le maniement de l'outil.

En nous appuyant sur cette combinaison d'indicateurs, nous avons défini cinq profils qui décrivent le traitement de la situation de programmation par le sujet : résolution *experte*, réussite après *ajustement*, réussite après *accommodation*, confrontation qui se termine par un *échec*, puzzle ouvert sans être traité (profil *passé*). Un sixième profil, désigné par *indéfini*, comprend les résolutions qui ne sont pas capturées par la modélisation.

Ces profils se sont avérés efficaces pour la conduite de notre recherche. Ils présentent cependant quelques limites que nous avons repérées. En effet, la non distinction entre erreurs et programmes partiels fait que la modélisation ne capture pas l'instrumentalisation de la fonctionnalité d'exécution. En conséquence, le profil ajustement recouvre des ajustements de plusieurs natures sans que nous soyons en mesure de les distinguer : rectification d'une erreur de mise en œuvre, mais aussi ajustement de la stratégie à la longueur de la solution attendue par une décomposition du problème, ajustement à la capacité à simuler

mentalement l'exécution par le lancement d'un programme partiel.

Même avec les limites identifiées, qui nous amènent à considérer le profil ajustement avec plus de vigilance, ce calcul automatique de profils constitue une première modélisation du comportement de l'apprenant, qui, au-delà de cette recherche, pourrait être utilisée pour affiner l'évaluation de compétences en programmation par rapport à une évaluation binaire (programme valide/invalide). La modélisation se généralise à des tâches pour lesquelles il est possible de définir une action élémentaire, dont la durée est quantifiable, comme le placement d'un bloc dans l'éditeur en programmation par blocs.

Classes de situations attachées au concept de motif

Nous avons réalisé un travail théorique sur le concept de motif qui nous a amené à proposer la définition du terme de motif dans notre contexte, comme une « entité repérable au sein d'un ensemble car répétée à l'identique ou avec des variations prédictibles ». Nous avons ensuite distingué deux occurrences de motif dans les situations de programmation d'un robot sur une grille : le *motif visuel* sur la grille et le *motif algorithmique* dont le programme constitue une représentation.

En nous appuyant sur cette proposition théorique, nous avons traité notre deuxième question de recherche qui consistait à rechercher des classes de situations attachées au concept de motif ainsi défini. Nous avons réalisé une analyse a priori qui nous a amené à étudier les caractéristiques des motifs en jeu et a abouti à la caractérisation de quatre classes de situations à partir des fréquences de réussite à l'échelle nationale. Ces classes de situations constituent une gradation dans la difficulté de la tâche de programmation d'un robot virtuel sur une grille. Nous avons approfondi l'étude pour les trois premières d'entre elles, et avons rectifié la variable de situation qui distingue les classes 2 et 3 au terme de l'analyse des données expérimentales à l'échelle des classes et à l'échelle du sujet. Au final, la classe 1 comprend les situations pour lesquelles la case délimite le motif visuel. Pour les classes 2 à 4, le motif s'étend sur plusieurs cases. La classe 2 comprend les situations pour lesquelles le robot est bien positionné pour aborder le motif suivant à l'issue de la dernière action sur la case d'un élément saillant, la classe 3 comprend celles pour lesquelles un repositionnement du robot est nécessaire. Quant à la classe 4, elle regroupe les situations pour lesquelles la lisibilité du motif visuel est perturbée. Dans la mesure où notre priorité était de traiter les premiers apprentissages en algorithmique et non la capacité à faire abstraction de distracteurs sur la grille, nous n'avons pas abordé cette dernière classe dans la thèse. Nous avons cependant montré au passage que la fonction des éléments présents sur la grille est loin d'être neutre, que la difficulté des situations est corrélée au degré d'aide ou de perturbation pour la lisibilité du

motif visuel, que les grilles épurées favorisent la réussite.

Caractérisation de la conceptualisation-en-acte

Notre dernière question de recherche consistait à étudier la construction des connaissances-en-acte du sujet à propos de la notion de boucle bornée. Nous avons cherché à identifier les conceptualisations-en-acte et les erreurs récurrentes qui révèlent une conceptualisation inappropriée.

Notre réponse à cette question prend la forme d'un système hiérarchique de schèmes, qui est construit par le sujet au cours de ses confrontations avec des situations de programmation de complexité croissante, situations qui impliquent d'utiliser une boucle bornée pour être résolues.

À travers ce système de schèmes, nous retrouvons les trois sous-tâches identifiées pour la programmation d'une boucle, mais ces sous-tâches prennent une forme spécifique dans le contexte de la programmation d'un robot virtuel sur une grille en langage Scratch :

- La planification du traitement revient à l'identification de motifs, à la mise en correspondance du motif visuel sur la grille et de la représentation du motif algorithmique dans le bloc *répéter*.
- La détermination de la condition d'arrêt correspond au dénombrement des motifs visuels sur la grille.
- La détermination de l'état initial des variables au début de la boucle revient à programmer le robot pour le positionner sur la grille au début de la suite de motifs visuels.

Chacune de ces sous-tâches comprend dans ce contexte une difficulté spécifique. La difficulté pour la planification du traitement est de vérifier et ajuster la jonction entre les itérations. Cette opération prend la forme d'un repositionnement du robot à l'identique relativement au motif visuel. Elle s'apparente à la détermination de l'invariant de boucle. Le dénombrement des motifs sur la grille implique que le motif soit constitué comme unité pour le dénombrement, ce qui est loin d'être évident pour les débutants. Un certain nombre d'entre eux n'établit pas de lien entre ce qu'ils placent dans le bloc *répéter* (une représentation du motif algorithmique dans le langage de programmation) et ce qu'ils dénombrent (parfois les cases, parfois les éléments saillants).

L'étude de la construction de ce système hiérarchique de schèmes nous amène aux conclusions suivantes :

La programmation d'une boucle se fait à partir du schème de dénombrement lorsque le corps de la boucle ne comprend qu'une seule instruction. Nous

sommes dans le cas d'une nouvelle connaissance construite à partir d'un champ conceptuel déjà approprié (Rogalski, 1987), à savoir celui de nombre dans son aspect cardinal. De notre point de vue, cela rend le concept de boucle accessible dès que le dénombrement de quantités discrètes et son expression avec une écriture chiffrée sont acquis pour un domaine numérique de l'ordre de la dizaine, soit environ vers 6 ans. Ce résultat rejoint et étaie les observations des expérimentations Motif.. Motif.. en grande section de maternelle et CP (Léonard et al., 2020 ; Léonard et al., 2021).

Mais peut-on vraiment parler dans ce cas de structure itérative dans la mesure où l'aspect cyclique du passage dans la boucle n'est pas ou peu présent ?

Une des spécificités du langage Scratch est d'introduire une structure itérative indépendamment de la notion de variable, via le bloc *répéter* qui masque la variable du compteur de boucle. Le nombre d'itérations est connu a priori, et doit être renseigné dans le champ dédié. Ce qui induit un champ conceptuel structuré différemment par rapport à celui qu'ont identifié Rogalski et Vergnaud (1987), dans lequel le concept de variable occupe une place centrale pour la structure itérative.

Dans le contexte de la programmation d'un robot virtuel sur une grille, ce sont des repères visuels sur la grille, repères en rapport avec le robot virtuel (sa position, son orientation), et l'état d'éléments sur celle-ci (objets présents/absents sur certaines cases, zones de dépôt vides/pleines...) qui sont explicites pour le sujet. En arrière-plan, tout cela est traduit en termes de valeurs de variables pour générer un état du système à un moment donné, et rétablir les variables comme fonctions de l'exécution. Mais pour le sujet, rien n'est visible. Là où l'expert voit une succession d'états statiques lors de l'exécution du programme, le sujet novice voit une succession d'actions du robot virtuel sur la grille. Nous avons une contextualisation de la dialectique entre représentation statique et représentation dynamique de l'exécution mise en évidence par Rogalski (1987). Rogalski mentionne que les élèves de seconde ne rencontre que plus tard le point de vue statique. Dans la mesure où les sujets auxquels nous nous sommes adressés sont encore plus jeunes, ils n'ont pas non plus encore été confrontés à cette dialectique.

Cette situation de programmation de robot sur une grille constitue une première confrontation entre les deux points de vue, statique et dynamique. Elle donne lieu à un savoir construit dans l'action, qui se traduit par un repositionnement du robot relativement au motif. Nous pouvons nous demander si ce savoir construit dans l'action porte une conceptualisation en mesure d'être généralisée. Nous nous demandons aussi comment se passe du point de vue cognitif le passage à une représentation plus experte. Les études réalisées, par exemple celle de Branthôme (2022), portent sur le passage du langage par blocs

au langage Python, et investiguent la notion de variable lors de cette transition. Cependant, l'introduction de la syntaxe de la boucle *for* au lieu du bloc *répéter* est simultanée avec l'introduction de la syntaxe du langage Python. Afin de distinguer les sources de difficulté lors de l'introduction de la variable du compteur de boucle, une expérimentation qui utiliserait le bloc *for* disponible pour les langages Blockly ou Snap ! nous semble une perspective intéressante dans l'étude du passage du niveau conceptuel que nous avons étudié au niveau supérieur.

Validité et limites des résultats obtenus

Nous avons établi nos résultats dans un contexte particulier : la résolution de puzzles avec un langage par blocs dans l'environnement du concours Algoréa. Nous nous posons la question de leur stabilité dans ce contexte, mais aussi de leur validité au-delà de celui-ci, dans une optique de généralisation. Cette démarche consiste en d'autres termes à « analyser précisément pour les notions et opérations en jeu dans les langages et les environnements actuels ce qui s'intègre dans des invariants de niveau supérieur, et d'autre part analyser les spécificités locales (d'un langage, d'un environnement, d'un dispositif informatique) qui devront être dépassés. » (Rogalski, 1987).

Validité des résultats pour le contexte étudié

Les résultats établis au cours de ce travail de thèse présentent plusieurs points forts. Ils s'appuient sur de larges échantillons à l'échelle nationale, ce qui leur confère une robustesse statistique pour la plupart. Ils sont étayés par des données plus fines et de natures différentes sur de plus petits échantillons. L'articulation entre ces différents types de données renforce la cohérence globale de l'interprétation.

À l'échelle du sujet et dans une moindre mesure à l'échelle des classes, nous avons montré que les premières connaissances construites sur la boucle au cours des sessions sont stables dans le temps. Lorsqu'une connaissance-en-acte est observée lors d'une session, nous la retrouvons lors des sessions suivantes, jusqu'à dix-huit mois plus tard pour certains sujets. Dans le même contexte, ces connaissances sont aussi remobilisées dans des situations pour lesquelles l'utilisation d'une boucle n'est pas contrainte, alors qu'une régression vers la séquence d'instructions serait possible.

Validité de nos résultats dans l'environnement du logiciel Scratch

L'élément de langage qui code la répétition, le bloc *répéter*, est exactement

identique à celui du logiciel Scratch original. Nous en déduisons que les résultats qui concernent la maîtrise instrumentale de ce bloc restent valides dans l'environnement du logiciel Scratch.

En revanche, la scène du logiciel Scratch n'est pas explicitement une grille , même si, en arrière-plan il s'agit d'un système de coordonnées. Nous ne retrouvons pas de motif visuel dans le cas général, sauf pour le cas de dessin de figures avec le module *stylo*. Ce module est une transposition directe du micromonde de la tortue Logo développé par Papert (1980). Dans ce cas, le motif visuel n'est pas explicitement délimité par les cases d'une grille. Il répond cependant à notre définition d'une entité repérable car se répétant à l'identique dans un ensemble plus grand. Contrairement aux puzzles, ce motif visuel n'est généralement pas présent sur la scène avant l'exécution du programme. Il constitue le but à atteindre. Dans les autres cas, seul le motif algorithmique attaché aux actions du lutin ou à l'évolution temporelle de la scène est présent. Dans le contexte du logiciel Scratch, il serait intéressant d'étudier comment le sujet procède pour ajuster la jonction entre deux itérations successives.

Par ailleurs, le bloc *avancer* est paramétré dans le logiciel Scratch, donc nous ne retrouvons pas la programmation d'une succession de déplacements identiques en utilisant le bloc *répéter*. Les premières confrontations avec la boucle se font directement avec une structure plus complexe, c'est-à-dire un motif algorithmique de longueur strictement supérieure à 1.

Validité de nos résultats pour des puzzles dans d'autres environnements de programmation par blocs

Pour le concours Algoréa, trois langages sont disponibles, dont le langage Blockly, qui est aussi très répandu (notamment utilisé par `code.org`). Nous n'avons pas mené d'investigations aux échelles des classes et du sujet, mais nous disposons des résultats à l'échelle nationale pour cet autre langage par blocs. Les taux de réussite aux puzzles sont comparables à ceux obtenus avec le langage Scratch. Nous en déduisons que nos résultats restent valides pour les langages de programmation par blocs qui sont similaires au langage Scratch. Notamment, nous considérons que nos résultats pourraient être reproduits avec des puzzles issus de `code.org`, qui ont sensiblement la même structure que ceux des parcours du concours Algoréa.

Validité de nos résultats au-delà des environnements de programmation par blocs

Nous avons vu que nous retrouvons, avec quelques nuances, la décomposition

de la programmation d'une boucle en sous-tâches repérée par d'autres auteurs pour les langages textuels (Jolivet et al., 2023 ; Rogalski & Samurçay, 1986). Nos résultats représentent une contextualisation de résultats plus généraux.

Il serait d'autre part possible de mener cette étude concernant les puzzles de programmation de robot sur une grille en langage Python en adaptant certains des outils. En effet, le langage Python est disponible dans l'environnement Algoréa, la fonction de validation des programmes ne dépend pas du langage, mais de l'état intermédiaire ou final de la grille. Il serait intéressant de comparer l'activité des sujets en langage par blocs et en langage Python, les autres variables de la situation étant identiques. Par exemple, il serait probablement possible de reproduire nos résultats relatifs à l'activité du sujet dans l'environnement de programmation textuelle du logiciel Pyrates développé par Branthôme, le logiciel Pyrates étant également constitué de puzzles de déplacement d'un personnage virtuel sur une grille en langage python (Branthôme, 2022).

Portée de ces résultats au-delà de la recherche

Mise à disposition des praticiens

Cette thèse identifie un certain nombre de paliers, de repères, d'erreurs interprétées en termes de difficultés. Celles-ci concernent soit la prise en main de l'environnement de programmation, soit constituent des difficultés conceptuelles. Nous pensons que cette compréhension des erreurs et de la conceptualisation-en-acte des élèves est de nature à outiller les enseignants qui initient leurs élèves à l'informatique. Les praticiens peuvent se saisir de ces résultats pour mieux comprendre l'activité de leurs élèves, et ajuster leurs scénarios pédagogiques. Les résultats obtenus à l'issue de nos recherches pourraient conduire à formuler des recommandations aux enseignants afin de leur permettre de mieux appréhender à la fois les enjeux de savoirs et les difficultés fréquentes pour lesquelles une vigilance particulière s'impose.

À l'échelle nationale et à l'échelle des classes, les résultats ont été obtenus dans le cadre d'un concours en ligne, qui suppose la mise en retrait de l'enseignant. Des recherches complémentaires, ou l'articulation avec d'autres recherches seraient nécessaires pour replacer l'analyse de l'activité du sujet au cœur du processus d'enseignement (Haspekian & Gélis, 2021 ; Robert & Rogalski, 2002). Dans ces conditions, nos résultats pourraient alimenter la réflexion des personnes qui décident du curriculum informatique au niveau de l'école obligatoire.

Amélioration des environnements de programmation par blocs

Au cours de nos analyses, nous avons pointé quelques points de vigilance concernant l'environnement de programmation Algoréa, parmi lesquels la représentation de l'exécution de certains blocs qui est non intuitive, la distorsion de la perspective en orientation relative, la non disponibilité du mode pas à pas lorsque la limite de blocs est atteinte. Ces observations pourront mener à des propositions d'amélioration. La progression du parcours pourra aussi être ajustée au regard des résultats obtenus.

Plus généralement, l'analyse détaillée de l'activité du sujet, notamment des mouvements du curseur de souris sur l'interface, a permis de montrer l'importance des gestes lors de la programmation dans un environnement par blocs. Du point de vue de l'ergonomie, cette étude peut apporter des éléments dans le but d'améliorer la prise en main des environnements par blocs.

Contributions du point de vue méthodologique

Nous avons mené une recherche pluridisciplinaire, caractérisée par la variété des cadres théoriques mobilisés et leur croisement. Nous avons ainsi convoqué des concepts issus de la didactique des mathématiques, de l'ergonomie, de la psychologie cognitive, de la modélisation informatique. Au final, nous proposons une thèse en didactique de l'informatique, qui est centrée sur un contenu qui relève de l'apprentissage / enseignement de l'informatique, tout en s'appuyant sur des concepts didactiques plus généraux et sur une méthode mixte dont une large part relève de traitements informatisés. La démarche constitue en soi une proposition méthodologique pour les communautés de recherche en didactique de l'informatique et en EIAH.

À un premier niveau, nous avons mobilisé la théorie des champs conceptuels de manière concomitante à des méthodes d'analyse de traces pour ce que cette démarche produit comme résultats sur la connaissance des processus cognitifs des élèves. Dans cette section, nous montrons en quoi la mobilisation conjointe de ces méthodes d'analyse de traces et de cadres théoriques issus de la didactique et de l'ergonomie constitue en soi des propositions méthodologiques de nature à construire des connaissances scientifiques robustes, et à prendre conscience de points plus faibles dans cette construction, ce qui nous semble tout aussi important.

Importance des gestes pour supporter le raisonnement

Notre recherche systématique des schèmes a fait émerger la place impor-

tante des gestes lors de la résolution de puzzles de programmation dans un environnement par blocs. Le schème de pointage accompagne le raisonnement du sujet tout au long de son activité de programmation : lecture de l'énoncé et du tutoriel, pointage constitutif du schème de dénombrement, simulation de l'exécution par pointage des cases de la grille, le curseur faisant fonction de robot pion, pointage de cases de la grille lors de phases de prise d'information. Dans le cas d'un environnement de programmation par blocs, l'apprentissage de l'aspect syntaxique du langage est pour une part remplacé par l'apprentissage du maniement des blocs, qui est d'ordre gestuel.

Notre méthodologie nous a permis mettre en évidence le rôle majeur des gestes sur l'interface, instrumentés par la souris. Ils nous révèlent les points d'attention du sujet dans leur composante dynamique. Nous sommes en mesure de capturer cette activité automatiquement, et d'en produire une reconstitution visuelle afin de pouvoir l'étudier.

Le résultat concernant l'importance des gestes pour supporter le raisonnement allié à la possibilité d'en produire une visualisation humainement interprétable en termes d'activité du sujet contribue à l'exploration des natures de données mobilisables dans le champ de la recherche en EIAH.

Articulation entre un cadre didactique et l'analyse de traces

Dans cette thèse, nous avons montré comment nous combinons les apports respectifs de la théorie des champs conceptuels et de l'analyse de traces afin d'étudier la résolution de puzzles de programmation avec un langage par blocs. Notre contribution du point de vue méthodologique porte sur la mise en relation de concepts didactiques avec des méthodes d'analyse de traces collectées automatiquement.

D'un côté, nous avons construit des indicateurs qui réfèrent explicitement au cadre didactique. La théorie des champs conceptuels nous fournit ainsi des clés pour structurer l'analyse de nos données et les interpréter. Réciproquement, les méthodes d'analyse de traces nous offrent la possibilité d'articuler d'une part des traitements automatisés des données collectées, des traitements statistiques sur de larges échantillons avec d'autre part la précision d'observation d'une approche qualitative. Cette double approche est de nature à apporter une robustesse aux résultats établis tout en assurant un lien étroit avec l'explicabilité de ces résultats, qui est portée par le cadre d'analyse.

Le croisement de ces cadres théoriques nous amènent à mettre en regard les axes préconisés par Vergnaud pour l'étude de la conceptualisation et les composantes de la modélisation informatique (Pelánek, 2017). De notre point de vue, nous pouvons mettre en correspondance l'axe des situations de la théo-

rie des champs conceptuels avec la modélisation du domaine du point de vue informatique. De la même manière, nous pouvons rapprocher l'axe des schèmes et la modélisation de l'apprenant. Par cette mise en relation d'un cadre didactique et des composantes de la modélisation informatique, nous apportons une contribution à la communauté de recherche qui s'intéresse à l'analyse des traces numériques de l'apprentissage (*learning analytics*). En effet, chercher à modéliser des concepts de cadres didactiques éprouvés constitue une proposition méthodologique qui renouvelle l'appui sur ces cadres didactiques pour analyser les processus d'apprentissage, en y intégrant l'apport des technologies actuelles.

Approfondissement des analyses permises par la méthodologie adoptée

Nous avons été en mesure de collecter des données pour les mêmes situations à plusieurs échelles. Pour chacune de ces échelles, nous avons adapté la granularité des données collectées pour obtenir un compromis opérationnel entre volume de données collectées et capacité à les traiter.

Ce protocole de collecte et de traitement nous a conduit à mener des analyses dont l'image du zoom peut donner une bonne idée. Dans l'objectif du zoom, la cible est la situation de programmation, que l'on va regarder avec de plus en plus de détails : d'abord du point de vue du taux de réussite à l'échelle nationale, puis successivement du point de vue de la répartition des profils de confrontation, de la nature et de la fréquence des programmes exécutés, avant de consulter des extraits vidéos de sessions pour les endroits repérés comme intéressants à transcrire et analyser à la main. Cette approche permet de gérer un volume important de données qualitatives, sans les explorer à la main de manière exhaustive.

Différenciation entre les difficultés d'ordre conceptuel et les difficultés d'ordre instrumental

Une autre contribution de notre travail concerne la différenciation dans l'analyse de l'activité du sujet entre les difficultés d'ordre conceptuel et les difficultés d'ordre instrumental. Cette distinction a été rendue possible par la précision des analyses que permet la collecte de données de plusieurs natures et à plusieurs échelles, adossée à un appui sur des concepts didactiques. Cette distinction entre difficultés conceptuelles et difficultés instrumentales imbriquées constitue l'un des apports de la thèse par rapport aux travaux existant sur la programmation par blocs (Vaníček et al., 2023).

Limites de la méthodologie

Après les points forts, nous présentons aussi une analyse des limites de notre méthodologie.

Notre méthodologie fonctionne bien pour l'analyse des premières notions algorithmiques. Malgré le volume important de données collectées, une limite est de collecter assez de données pour les notions plus avancées, car, dans un groupe aléatoire de sujets dont la très grande majorité est novice au début de l'expérimentation, un nombre moindre de sujets atteint ce niveau de compétence.

Nous collectons en outre très peu de données contextuelles. D'autres variables ont très probablement un effet sur l'activité du sujet, variables qui constituent un angle mort de l'étude.

D'un point de vue plus général, cette méthodologie nécessite un investissement initial conséquent en termes de mise en place du dispositif de collecte de données, de conception des outils de traitement et d'analyse des traces. Elle ne peut donc s'envisager que pour des études d'une certaine ampleur et s'inscrit sur un temps long, parfois difficilement compatible avec la durée allouée pour les projets de recherche.

Enfin, cette méthodologie nécessite de s'adosser à un dispositif existant à large échelle, et donc soit de rendre compatible les objectifs de l'étude avec celui du dispositif existant, soit de collaborer avec la structure qui porte le dispositif à large échelle pour y intégrer les objectifs de l'étude.

Perspectives

Malgré la durée et l'ampleur conséquente de ce travail, le degré de précision des analyses fait qu'il reste inachevé, nous laissant ainsi de nombreuses perspectives pour le futur.

Perspectives d'étude de la catégorisation des situations

Lors de l'expérimentation complémentaire en juillet 2022, nous avons proposé une tâche de catégorisation des grilles des puzzles (8.4.2). Nous avons collecté un volume de données conséquent : 18 sujets qui ont chacun catégorisé les grilles d'une dizaine de planches, ce qui représente donc 168 catégorisations. Nous avons transcrit les verbatims enregistrés lors des sessions par visioconférence. Nous obtenons un document de 180 pages qui contient les transcriptions. Une première identification des critères est commencée. Cependant, le temps nous a manqué pour finaliser cette partie de l'étude. Nous en donnons un aperçu en annexe 13. Un début d'analyse sommaire laisse entrevoir un palier dans la conceptualisation au moment où les sujets ont une première maîtrise des deux

structures de contrôle, répétition et structure conditionnelle. Nous observons à ce moment le passage d'une catégorisation des grilles selon des critères de surface à une catégorisation selon des critères algorithmiques. Une perspective à court terme est de finaliser cette analyse des critères de catégorisation des grilles de programmation, qui nous donne des éléments sur la conceptualisation du sujet, à distance de l'activité de programmation elle-même.

Perspectives de reprise de la même méthodologie pour l'investigation d'autres notions algorithmiques

Comme nous avons adossé notre protocole au concours Algoréa, nous avons fait passer des sessions à l'échelle des classes et à l'échelle du sujet pour les catégories blanche et jaune du concours. Nous disposons donc de données beaucoup plus larges que pour la seule notion de boucle bornée. Les données déjà collectées permettent d'ébaucher l'étude des autres notions algorithmiques de base : structure conditionnelle, variable et procédure. L'avantage est que ces données ont été collectées sur les mêmes sujets et les résultats pourront donc être articulés avec ceux obtenus dans le cadre de ce travail doctoral.

Cependant, le volume de données est moins conséquent pour ces autres notions, et il faudrait probablement procéder à une nouvelle collecte pour assurer une robustesse statistique du même ordre que pour la présente étude.

Perspectives de passage à l'échelle du point de vue de la collecte de données

Deux perspectives d'extension de la collecte de données sont envisagées, avec pour objectif de changer d'échelle pour certains traitements. D'une part, le module de collecte des mouvements de souris, qui a été activé lors de l'expérimentation complémentaire est prêt à être utilisé à l'échelle des classes. D'autre part, la collecte de tous les programmes exécutés pour un échantillon randomisé de participants à l'échelle nationale est à l'étude.

Perspectives d'adjonction de modèles de machine learning

Dans la mesure où nous considérons que les indicateurs que nous avons construits constituent des éléments de modélisation du domaine et de modélisation de l'apprenant, l'adjonction de modèles de *machine learning* à ces modélisations ouvre des perspectives de recherche pour ajouter une composante prédictive à nos résultats. Nous avons ébauché le travail dans cette direction en proposant un modèle de régression linéaire dans le chapitre 11. D'autre part,

la détermination automatique de profils a donné lieu à une collaboration pour un essai de prédiction avec un modèle d'apprentissage basé sur le mécanisme d'attention (Bouton et al., 2022).

La précision de l'étiquetage des situations de programmation, la possibilité de passer à l'échelle le volume de données collectées, les indicateurs construits et le calcul automatique des profils, sont autant d'éléments sur lesquels nous pourrons nous appuyer pour poursuivre dans cette direction. Cet axe de recherche porte des enjeux relatifs à de la rétroaction adaptative et à l'adaptation automatique des parcours.

Ainsi, nous espérons, par la continuation de ce travail de thèse, contribuer à la recherche en didactique de l'informatique, en particulier à la connaissance de la manière dont les jeunes élèves construisent leurs compétences lorsqu'ils abordent ce domaine à travers la programmation en langage par blocs.

Bibliographie

Abdullah, A. H., Othman, M. A., Ismail, N., Abd Rahman, S. N. S., Mokhtar, M., & Zaid, N. M. (2019). Development of mobile application for the concept of pattern recognition in computational thinking for mathematics subject. *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*, 1-9. https://doi.org/10.1109/TALE48000.2019.9225910 (cf. p. 56)

AbdulSamad, U., & Romli, R. (2022). A comparison of block-based programming platforms for learning programming and creating simple application. In F. Saeed, F. Mohammed & F. Ghaleb (Éd.), *Advances on intelligent informatics and computing* (p. 630-640). Springer International Publishing. (Cf. p. 42).

Aho, A. V. (2012). Computation and computational thinking. *The computer journal*, *55*(7), 832-835. https://doi.org/10.1093/comjnl/bxs074 (cf. p. 18, 22)

Aivaloglou, E., & Hermans, F. (2016). How kids code and how we know : An exploratory study on the Scratch repository. *Proceedings of the 2016 ACM conference on international computing education research*, 53-61. https://doi.org/10.1145/2960310.2960325 (cf. p. 43, 47)

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework : Implications for teacher knowledge. *Journal of Educational Technology & Society*, *19*(3), 47-57 (cf. p. 58).

Archambault, J.-P., Dowek, G., & Cimelli, C. (2012). *Informatique et sciences du numérique : spécialité ISN en terminale S, avec des exercices corrigés et idées de projets*. Editions Eyrolles. (Cf. p. 11).

Arsac, J. (1987). Informatique et enseignement général. *L'éducation et l'informatisation de la société, annexe 1*, 152-165 (cf. p. 12, 15, 16).

Artigue, M. (1988). Ingénierie didactique. *Recherches en didactique des mathématiques, 9(3)*, 281-308 (cf. p. 117).

Baron, G.-L. (2017). Réflexions sur la didactique de l'informatique : le cas de la France. *L'informatique et le numérique dans la classe : Qui, quoi, comment ?* (Cf. p. 12).

Baron, G.-L., & Bruillard, É. (2001). Une didactique de l'informatique ? *Revue française de Pédagogie*, 163-172. https://doi.org/10.3406/rfp.2001.2813 (cf. p. 10, 12, 13, 15, 26, 27, 45)

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* (J. Lebeaume, A. Hasni, & I. Harlé, p. 79-90). De Boeck. (Cf. p. 10).

Baron, G.-L., & Drot-Delange, B. (2016a). L'éducation à l'informatique à l'école primaire. *1024 : Bulletin de la Société Informatique de France*, *8*, 73-79 (cf. p. 64).

Baron, G.-L., & Drot-Delange, B. (2016b). L'informatique comme objet d'enseignement à l'école primaire française ? Mise en perspective historique. *Revue française de pédagogie. Recherches en éducation*, *195*, 51-62. https://doi.org/10.4000/rfp.5032 (cf. p. 10)

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. (Cf. p. 2, 17).

Barrón-Estrada, M. L., Zatarain-Cabada, R., Romero-Polo, J. A., & Monroy, J. N. (2021). Patrony : A mobile application for pattern recognition learning. *Education and Information Technologies*, *24*, 1237-1260. https://doi.org/10.1007/s10639-021-10636-7 (cf. p. 56-58)

Bationo Tillon, A., & Rabardel, P. (2015). L'approche instrumentale : conceptualiser et concevoir pour le développement. In *L'ergonomie orientée enfants. Concevoir pour le développement* (Françoise Decortis). https://hal.science/hal-04455129. (Cf. p. 90, 91, 129)

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming : blocks and beyond. *Communications of the ACM*, *60*(6), 72-80. https://doi.org/10.1145/3015455 (cf. p. 39-41)

Begosso, L. C., Begosso, L. R., & Christ, N. A. (2020). An analysis of block-based programming environments for CS1. *2020 IEEE Frontiers in Education Conference (FIE)*, 1-5. https://doi.org/10.1109/FIE44824.2020.9273982 (cf. p. 42)

Bell, T., Witten, I. H., & Fellows, M. (2002). *Computer Science Unplugged . . . off-line activities and games for all ages*. (Cf. p. 27).

Bellegarde, K., Boyaval, J., & Alvarez, J. (2019). S'initier à la robotique/informatique en classe de grande section de maternelle. Une expérimentation autour de l'utilisation du robot Blue Bot comme jeux sérieux. *Review of*

Science, Mathematics and ICT Education, *13*(1), 51-72. https://doi.org/10.26220/rev.3105 (cf. p. 26)

Berry, G. (2014). *La pensée informatique, cœur du monde numérique*. Dailymotion. https://www.dailymotion.com/video/x1zvso5. (Cf. p. 13)

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* (cf. p. 1, 2, 112).

Berry, G. (2017). *L' Hyperpuissance de l'informatique : Algorithmes, données, machines, réseaux*. Odile Jacob. (Cf. p. 12-15, 17, 35, 215).

Bers, M. U. (2018). Coding and computational thinking in early childhood : The impact of ScratchJr in Europe. *European Journal of STEM Education*, *3*(3), 8. https://doi.org/10.20897/ejsteme/3868 (cf. p. 44)

Bers, M. U. (2019). Coding as another language : a pedagogical approach for teaching computer science in early childhood. *Journal of Computers in Education*, *6*(4), 499-528. https://doi.org/10.1007/s40692-019-00147-3 (cf. p. 39, 44, 46, 47, 53, 60)

Bers, M. U., & Resnick, M. (2015). *The official ScratchJr book : Help your kids learn to code*. No Starch Press. (Cf. p. 44).

Bouton, M., Peter, Y., Léonard, M., & El Mawas, N. (2022). Tentative de prédiction de la réussite à un exercice de programmation (cf. p. 185, 361).

Brackmann, C. P., Román-González, M., Robles, G., Moreno-León, J., Casali, A., & Barone, D. (2017). Development of Computational Thinking Skills through Unplugged Activities in Primary School. *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, 65-72. https://doi.org/10.1145/3137065.3137069 (cf. p. 59)

Branthôme, M. (2022). Conception et évaluation du jeu sérieux Pyrates : agencement du milieu didactique pour la transition Scratch-Python. *Didapro 9 – DidaSTIC 9ème colloque francophone de didactique de l'informatique*. https://hal.science/hal-03674705/ (cf. p. 352, 355)

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 annual meeting of the American educational research association*, *1*, 25 (cf. p. 22, 33).

Brousseau, G. (1997). La théorie des situations didactiques. Cours donné lors de l'attribution à Guy Brousseau du titre de Docteur Honoris Causa de l'Université de Montréal. *Interactions didactiques* (cf. p. 344).

Brousseau, G. (1986). Fondements et méthodes de la didactique des mathématiques. *Recherches en didactique des mathématiques*, *7*(2), 33-115 (cf. p. 121, 122).

Brousseau, G. (1998). *Théorie des situations didactiques : Didactique des mathématiques 1970-1990*. La Pensée Sauvage. (Cf. p. 116, 118-120, 122).

Brousseau, G. (2002). Glossaire de quelques concepts de la théorie des situations didactiques en mathématiques. http://guy-brousseau.com/wp-content/uploads/2010/09/Glossaire_V5.pdf (cf. p. 120, 121)

Brousseau, G. (2011). La théorie des situations didactiques en mathématiques. *Éducation et didactique*, *5*, 101-104. https://doi.org/10.4000/educationdidactique.1005 (cf. p. 143, 344)

Brousseau, G. (2012). Des dispositifs piagétiens… aux situations didactiques. *Éducation et didactique*, *6*(2), 103-129. https://doi.org/10.4000/educationdidactique.1475 (cf. p. 120)

Bruner, J. (1983). *Le développement de l'enfant. Savoir faire, savoir dire*. PUF. (Cf. p. 90).

Burgoyne, K., Witteveen, K., Tolan, A., Malone, S., & Hulme, C. (2017). Pattern understanding : relationships with arithmetic and reading development. *Child Development Perspectives*, *11*(4), 239-244. https://doi.org/10.1111/cdep.12240 (cf. p. 54)

Busana, G., Denis, B., Duflot-Kremer, M., Higuet, S., Kataja, L., Kreis, Y., Laduron, C., Meyers, C., Parmentier, Y., & Reuter, R. (2019). PIAF : développer la Pensée Informatique et Algorithmique dans l'enseignement Fondamental. https://hal.science/hal-02424418/. (Cf. p. 29)

Çakiroğlu, Ü., & Mumcu, S. (2020). Focus-fight-finalize (3F) : problem-solving steps extracted from behavioral patterns in block based programming. *Journal of Educational Computing Research*, *58*(7), 1279-1310. https://doi.org/10.1177/0735633120930673 (cf. p. 179, 212)

Chevalier, M., Giang, C., Piatti, A., & Mondada, F. (2020). Fostering computational thinking through educational robotics : a model for creative computational problem solving. *International Journal of STEM Education*, *7*(1), 1-18. https://doi.org/10.1186/s40594-020-00238-z (cf. p. 160, 346)

Ching, Y.-H., Hsu, Y.-C., & Baldwin, S. (2018). Developing computational thinking with educational technologies for young learners. *TechTrends*, *62*(6), 563-573. https://doi.org/10.1007/s11528-018-0292-7 (cf. p. 25, 26)

Ciccione, L., & Dehaene, S. (2023). *Les motifs, source d'éveil aux mathématiques en maternelle et au primaire* (N° 10). Conseil Scientifique de l'Éducation Nationale. (Cf. p. 65, 66, 68, 231, 255).

Collins, M. A., & Laski, E. V. (2015). Preschoolers' strategies for solving visual pattern tasks. *Early Childhood Research Quarterly*, *32*, 204-214. https://doi.org/10.1016/j.ecresq.2015.04.004 (cf. p. 53, 61, 63, 64)

Crahay, M. (2006). Dangers, incertitudes et incomplétude de la logique de la compétence en éducation. *Revue française de pédagogie. Recherches en éducation*, *154*, 97-110. https://doi.org/10.4000/rfp.143 (cf. p. 157)

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational thinking. A guide for teachers. *Computing at school* (cf. p. 20, 57, 58).

Da Silva Junior, B. A., Da Silva, J. V., Da Costa Cavalheiro, S. A., & Foss, L. (2022). Pattern Recognition in Computing Education : A Systematic Review. *Anais do XXXIII Simpósio Brasileiro de Informática na Educação*, 232-243. https://doi.org/10.5753/sbie.2022.225128 (cf. p. 56)

Dagienė, V., Futschek, G., & Stupurienė, G. (2019). Creativity in Solving Short Tasks for Learning Computational Thinking. *Constructivist Foundations*, *14*(3), 413-415. https://constructivist.info/14/3/413.dagiene (cf. p. 58)

Dasgupta, A., & Purzer, S. (2016). No patterns in pattern recognition : A systematic literature review. *2016 IEEE Frontiers in Education Conference (FIE)*, 1-3. https://doi.org/10.1109/FIE.2016.7757676 (cf. p. 56)

Declercq, C., & Tort, F. (2018). Organiser l'apprentissage de la programmation au cycle 3 avec des activités guidées et/ou créatives. *RJC EIAH 2018*. https://hal.science/hal-01765408/ (cf. p. 32)

Declercq, C., & Zeyringer, M. (2018). Analyse de l'activité des élèves dans l'environnement PixelArt pour l'apprentissage de la séquence et de la répétition au cycle 3. *ETIC3*. https://doi.org/https://hal.science/hal-01826065 (cf. p. 32, 159, 160, 305)

Delmas-Rigoutsos, Y. (2018). Proposition de structuration historique des concepts de la pensée informatique fondamentale. *Didapro 7–DidaSTIC. De 0 à 1 ou l'heure de l'informatique à l'école*. https://doi.org/https://hal.science/hal-01752797/ (cf. p. 12, 14)

Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33-39. https://doi.org/10.1145/2998438 (cf. p. 17, 18, 21, 28, 81)

Develay, M. (1993). Pour une épistémologie des savoirs scolaires. *Pédagogie collégiale*, *7*(1), 35-40 (cf. p. 11).

Douady, R. (1986). Jeux de cadres et dialectique outil-objet. *Recherches en didactique des mathématiques*, *38*(1), 1-15. https://revue-rdm.com/1986/jeux-de-cadres-et-dialectique/ (cf. p. 4)

Dowek, G. (2011). Les quatre concepts de l'informatique. *Sciences et technologies de l'information et de la communication en milieu éducatif : Analyse de pratiques et enjeux didactiques.*, 21-29. https://edutice.hal.science/edutice-00676169/ (cf. p. 11-14, 16, 18, 35, 105, 215)

Drot-Delange, B., Pellet, J.-P., Delmas-Rigoutsos, Y., & Bruillard, É. (2019). Pensée informatique : points de vue contrastés. *Sticef*, *26*, 39-61. https://doi.org/1023709/sticef.26.11 (cf. p. 16, 17, 21, 35)

El Rouadi, N. (1999). *Programmation informatique et conceptualisation entre 13 et 15 ans* (thèse de doct.). Paris 5. (Cf. p. 80, 96, 334).

Ericsson, K. A., & Simon, H. A. (1993). *Protocol Analysis : Verbal Reports as Data*. The MIT Press. (Cf. p. 133).

européenne, C. (2022). Eurydice Report : Informatics education at school in Europe. *Publications Office of the European Union : Luxembourg* (cf. p. 1).

Fagerlund, J., Häkkinen, P., Vesisenaho, M., & Viiri, J. (2021). Computational thinking in programming with scratch in primary schools : A systematic review. *Computer Applications in Engineering Education*, *29*(1), 12-28. https://doi.org/10.1002/cae.22255 (cf. p. 22, 42, 58)

Fernandez, C., Freitas, J. A., Lopes, R. d. D., & Blikstein, P. (2022). Using video analysis and learning analytics to understand programming trajectories in data science activities with Scratch. *Interaction Design and Children*, 253-260. https://doi.org/10.1145/3501712.3529742 (cf. p. 131, 160)

Flick, U. (2011). Triangulation. In G. Oelerich & H.-U. Otto (Éd.), *Empirische Forschung und Soziale Arbeit* (p. 323-328). VS Verlag für Sozialwissenschaften. https://doi.org/10.1007/978-3-531-92708-4_23. (Cf. p. 148)

Fluckiger, C. (2019). *Une didactique de l'informatique scolaire*. Presses universitaires de Rennes. https://hal.univ-lille.fr/hal-02143687. (Cf. p. 1, 23, 348)

Forget, M.-H. (2013). Le développement des méthodes de verbalisation de l'action : un apport certain à la recherche qualitative. *Recherches qualitatives*, *32*(1), 57-80. 10.7202/1084612ar (cf. p. 132)

Fraser, N. (2015). Ten things we've learned from Blockly. *2015 IEEE blocks and beyond workshop (blocks and beyond)*, 49-50. https://doi.org/10.1109/BLOCKS.2015.7369000 (cf. p. 34, 41, 42)

Furber, S. (2012). *Shut down or restart : the way forward for computing in UK schools*. The Royal Society. https://royalsociety.org/topics-policy/projects/computing-in-schools/report/. (Cf. p. 19)

Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns : elements of reusable object-oriented software*. Pearson Deutschland GmbH. (Cf. p. 53).

Gao, G., Marwan, S., & Price, T. W. (2021). Early performance prediction using interpretable patterns in programming process data. *Proceedings of the 52nd ACM technical symposium on computer science education*, 342-348. https://doi.org/10.48550/arXiv.2102.05765 (cf. p. 129, 170)

Gaubert-Macon, C., Chesneaux, J.-M., Desprez, J.-M., Picaronny, C., & Montreuil, V. (2022). *Pratique de l'informatique aux cycles 3 et 4*. Inspection générale de l'éducation, du sport et de la recherche. https://www.education.gouv.fr/pratique-de-l-informatique-aux-cycles-3-et-4-344254. (Cf. p. 2, 22, 23, 25-28, 35, 36, 64)

Gelman, R., & Gallistel, C. R. (1978). *The child's understanding of number*. Harvard University Press. (Cf. p. 77, 293).

González, M. R. (2015). Computational thinking test : Design guidelines and content validation. *Proceedings of EDULEARN15 conference*, 2436-2444 (cf. p. 21).

Gouédard, C., & Bationo-Tillon, A. (2022). La conception du pouvoir d'agir selon Pierre Rabardel : quelles filiations conceptuelles avec Gérard Vergnaud ? *Bulletin de psychologie*, *75*(4), 347-356. https://doi.org/10.3917/bupsy.578.0347 (cf. p. 84)

Gouws, L., Bradshaw, K., & Wentworth, P. (2013a). Computational thinking in educational activities : an evaluation of the educational game light-bot. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 10-15. https://doi.org/10.1145/2462476.2466518 (cf. p. 59)

Gouws, L., Bradshaw, K., & Wentworth, P. (2013b). First year student performance in a test for computational thinking. *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, 271-277. https://doi.org/10.1145/2513456.2513484 (cf. p. 19)

Grover, S. (2020). Designing an assessment for introductory programming concepts in middle school computer science. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 678-684. https://doi.org/10.1145/3328778.3366896 (cf. p. 21)

Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming : examining misconceptions of loops, variables, and boolean logic. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 267-272. https://doi.org/10.1145/3017680.3017723 (cf. p. 34, 35, 287)

Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., & Stamper, J. (2017). A framework for using hypothesis-driven approaches to support data-driven learning analytics in measuring computational thinking in block-based programming environments. *ACM Transactions on Computing Education (TOCE)*, *17*(3), 1-25. https://doi.org/10.1145/3105910 (cf. p. 178, 212)

Grover, S., & Pea, R. (2013). Computational thinking in K–12 : A review of the state of the field. *Educational researcher*, *42*(1), 38-43. https://doi.org/10.3102/0013189X12463051 (cf. p. 19, 22)

Grover, S., & Pea, R. (2018). Computational thinking : A competency whose time has come. *Computer science education : Perspectives on teaching and learning in school*, *19*(1), 19-38 (cf. p. 17, 19, 57, 59).

Harvey, B., Garcia, D. D., Barnes, T., Titterton, N., Armendariz, D., Segars, L., Lemon, E., Morris, S., & Paley, J. (2013). SNAP ! (build your own blocks). *Proceeding of the 44th ACM technical symposium on Computer science education*, 759-759. https://doi.org/10.1145/2445196.2445507 (cf. p. 44)

Haspekian, M., & Gélis, J.-M. (2021). Informatique, Scratch et robots : de nouvelles pratiques enseignantes en mathématiques ? *STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation)*, *28*(1), 13-49. https://doi.org/10.23709/sticef.28.1.1 (cf. p. 43, 355)

Hoc, J.-M. (1979). Le problème de la planification dans la construction d'un programme informatique. *Le Travail humain*, *42*(2), 245-260 (cf. p. 28).

Hofstadter, D., & Sander, E. (2013). *L'analogie, cœur de la pensée*. Odile Jacob. https://hal.science/hal-02113546/. (Cf. p. 182)

Hsu, T.-C., Chang, S.-C., & Hung, Y.-T. (2018). How to learn and how to teach computational thinking : suggestions based on a review of the literature. *Computers & Education*, *126*, 296-310. https://doi.org/10.1016/j.compedu.2018.07.004 (cf. p. 56-58, 60)

Hu, Y., Chen, C.-H., & Su, C.-Y. (2021). Exploring the effectiveness and moderators of block-based visual programming on student learning : a meta-analysis. *Journal of Educational Computing Research*, *58*(8), 1467-1493. https://doi.org/10.1177/0735633120945935 (cf. p. 39)

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Börstler, J., Edwards, S. H., Isohanni, E., Korhonen, A., Petersen, A., & Rivers, K. (2015). Educational data mining and learning analytics in programming : Literature review and case studies. *Proceedings of the 2015 ITiCSE on Working Group Reports*, 41-63. https://doi.org/10.1145/2858796.2858798 (cf. p. 130, 139)

Inhelder, B., & De Caprona, D. (1992). Vers le constructivisme psychologique : Structures ? Procédures ? Les deux indissociables. *Le cheminement des découvertes de l'enfant*, 19-50 (cf. p. 90).

ISTE & CSTA. (2011). Operational definition of computational thinking for K-12 education. *National Science Foundation* (cf. p. 20).

Jääskeläinen, R. (2010). Think-aloud protocol. *Handbook of translation studies*, *1*, 371-374 (cf. p. 133).

Jiang, B., Zhao, W., Zhang, N., & Qiu, F. (2022). Programming trajectories analytics in block-based programming language learning. *Interactive Learning Environments*, *30*(1), 113-126. https://doi.org/10.1080/10494820.2019.1643741 (cf. p. 160)

João, P., Nuno, D., Fábio, S. F., & Ana, P. (2019). A cross-analysis of block-based and visual programming apps with computer science student-teachers [Publisher : MDPI]. *Education Sciences*, *9*(3), 181. https://doi.org/10.3390/educsci9030181 (cf. p. 42)

Jolivet, S., Dechaux, E., Gobard, A.-C., & Wang, P. (2023). Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation. *Actes de la onzième Conférence sur les Environnements Informatiques pour l'Apprentissage Humain*. https://orfee.hepl.ch/handle/20.500.12162/6759 (cf. p. 29, 32, 33, 345, 355)

Kalas, I., Blaho, A., & Moravcik, M. (2018). Exploring control in early computing education. In S. N. Pozdniakov & V. Dagienė (Éd.), *Informatics in schools. fundamentals of computer science and software engineering* (p. 3-16). Springer International Publishing. https://doi.org/10.1007/978-3-030-02750-6_1. (Cf. p. 191, 225)

Kalelioglu, F., Gulbahar, Y., & Kukul, V. (2016). A framework for computational thinking based on a systematic research review. *Baltic Journal of Modern Computing*, *4*, 583-596 (cf. p. 55-57).

Kalelioğlu, F. (2015). A new way of teaching programming skills to K-12 students : Code. org. *Computers in Human Behavior*, *52*, 200-210. https://doi.org/10.1016/j.chb.2015.05.047 (cf. p. 44)

Kellman, P. J., & Garrigan, P. (2009). Perceptual learning and human expertise. *Physics of life reviews*, *6*(2), 53-84. https://doi.org/10.1016/j.plrev.2008.12.001 (cf. p. 56)

Khan, M., & Khan, S. S. (2011). Data and information visualization methods, and interactive mechanisms : A survey. *International Journal of Computer Applications*, *34*(1), 1-14 (cf. p. 160).

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly*, *92*(3), 170-181. https://doi.org/10.1080/00029890.1985.11971572 (cf. p. 28)

Knuth, D. E., & Cégielski, P. (2011). *Eléments pour une histoire de l'informatique* (CSLI Publications-SMF Paris). (Cf. p. 14).

Komis, V., & Misirli, A. (2013). Étude des processus de construction d'algorithmes et de programmes par les petits enfants à l'aide de jouets programmables. *Sciences et technologies de l'information et de la communication*

(STIC) en milieu éducatif. https://edutice.hal.science/edutice-00875628/ (cf. p. 211)

Krieter, P. (2020). *Looking inside-mobile screen recordings as a privacy friendly long-term data source to analyze user behavior* (thèse de doct.). Universität Bremen. https://media.suub.uni-bremen.de/handle/elib/4318. (Cf. p. 132, 178, 180)

Kushniruk, A. W., & Patel, V. L. (2004). Cognitive and usability engineering methods for the evaluation of clinical information systems. *Journal of biomedical informatics*, *37*(1), 56-76. https://doi.org/10.1016/j.jbi.2004.01.003 (cf. p. 132)

Lagrange, J.-B., & Rogalski, J. (2017). Savoirs, concepts et situations dans les premiers apprentissages en programmation et en algorithmique. *Annales de Didactiques et de Sciences Cognitives*. https://doi.org/10.4000/adsc.723 (cf. p. 10, 11, 15, 29, 36, 96, 217)

Lazar, J., Feng, J. H., & Hochheiser, H. (2017). *Research methods in human-computer interaction*. Wiley Publishing. (Cf. p. 132).

Lee, K., Ng, S. F., Bull, R., Pe, M. L., & Ho, R. H. M. (2011). Are patterns important ? An investigation of the relationships between proficiencies in patterns, computation, executive functioning, and algebraic word problems. *Journal of Educational Psychology*, *103*(2), 269-281. https://doi.org/10.1037/a0023068 (cf. p. 54)

Leleu-Merviel, S., & Useille, P. (2008). Quelques révisions du concept d'information. In *Problématiques émergentes dans les sciences de l'information* (Hermès, p. 25-56). Lavoisier. https://hal.science/hal-00695777. (Cf. p. 12)

Lenay, C., Stewart, J., & Gapenne, O. (2002). Espace d'action technique et geste perceptif. *Le geste technique : réflexions méthodologiques et anthropologiques*, *2*(14), 215-230 (cf. p. 193).

Léonard, M. (2020). Score et chronomètre sur l'interface : Quels effets sur les résultats et la perception d'un concours en ligne. *Actes des huitièmes rencontres jeunes chercheur-es en EIAH*, 56-64. https://hal.science/hal-03208548v1/file/Actes_RJC_EIAH_2020.pdf#page=57 (cf. p. 120)

Léonard, M. (2024). Zoom sur quelques erreurs récurrentes lors des premiers apprentissages en algorithmique. *Enseigner, apprendre, former à l'informatique à l'école : regards croisés*, 135-154. https://hal.science/hal-04687657/ (cf. p. 348)

Léonard, M., Bouton, M., & Peter, Y. (2023). Détermination de profils relatifs à la mobilisation de schème lors de la résolution de puzzles de programmation. *Actes de la onzième Conférence sur les Environnements Informatiques*

pour l'Apprentissage Humain, 133-144. https://hal.science/hal-04354452 (cf. p. 185, 348)

Léonard, M., Peter, Y., & Secq, Y. (2019). Patterns and loops : early computational thinking. *European Conference on Technology Enhanced Learning*, 280-293. https://doi.org/10.1007/978-3-030-29736-7_21 (cf. p. 2, 16, 348)

Léonard, M., Peter, Y., & Secq, Y. (2020). Reconnaissance et synthèse de motifs redondants avec des élèves de 6-7 ans MOTIFS.MOTIFS.MOTIFS. <=> 3 x MOTIFS. 3 x MOTIFS. *Colloque DIDAPRO 8 -DIDASTIC - L'informatique, objets d'enseignements enjeux épistémologiques, didactiques et de formation*. https://hal.science/hal-02971775/ (cf. p. 3, 352)

Léonard, M., Peter, Y., Secq, Y., Alvarez, J., & Fluckiger, C. (2021). MOTIF.. MOTIF.. : initier à la notion de répétition en maternelle sans mobiliser de repérage spatial. *Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation*, *28*(3). https://www.persee.fr/doc/stice_1764-7223_2021_num_28_3_1824 (cf. p. 3, 69, 224, 227, 352)

Léonard, M., Peter, Y., Secq, Y., & Fluckiger, C. (2022). Computational Thinking : Focus on Pattern Identification. *European Conference on Technology Enhanced Learning*, 187-200. https://doi.org/10.1007/978-3-031-16290-9_14 (cf. p. 67, 68, 237, 348)

Léonard, M., Secq, Y., Peter, Y., & Fluckiger, C. (2022). Pensée informatique : approche didactique de l'identification de motifs. *Colloque Didapro 9 : L'informatique, objets d'enseignement et d'apprentissage. Quelles nouvelles perspectives pour la recherche ?*, 113-125. https://hal.science/hal-03697950 (cf. p. 67, 68, 185, 237, 348)

Liljedahl, P. (2004). Repeating pattern or number pattern : The distinction is blurred. *Focus on learning problems in mathematics*, *26*(3), 24-42 (cf. p. 53, 54, 61, 230).

Linehan, C., Bellord, G., Kirman, B., Morford, Z. H., & Roche, B. (2014). Learning curves : analysing pace and challenge in four successful puzzle games. *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, 181-190. https://doi.org/10.1145/2658537.2658695 (cf. p. 48, 49, 118, 123)

Liu, Z., Zhi, R., Hicks, A., & Barnes, T. (2017). Understanding problem solving behavior of 6–8 graders in a debugging game. *Computer Science Education*, *27*(1), 1-29. https://doi.org/10.1080/08993408.2017.1308651 (cf. p. 35)

McGill, M. M., & Decker, A. (2020). Construction of a taxonomy for tools, languages, and environments across computing education. *Proceedings of the 2020 ACM Conference on International Computing Education Research*, 124-135. https://doi.org/10.1145/3372782.3406258 (cf. p. 42)

McMillan, D., McGregor, M., & Brown, B. (2015). From in the wild to in vivo : video analysis of mobile device use. *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 494-503. https://doi.org/10.1145/2785830.2785883 (cf. p. 178)

Mervis, C. B., & Rosch, E. (1981). Categorization of natural objects. *Annual Review of Psychology*, *32*(1), 89-115. https://doi.org/10.1146/annurev.ps.32.020181.000513 (cf. p. 136)

Miller, G. A. (1956). The magical number seven, plus or minus two : Some limits on our capacity for processing information. *Psychological review*, *63*(2), 81. https://doi.org/10.1037/h0043158 (cf. p. 40)

Miller, J. (2019). STEM education in the primary years to support mathematical thinking : using coding to identify mathematical structures and patterns. *ZDM*, *51*(6), 915-927. https://doi.org/10.1007/s11858-019-01096-y (cf. p. 56)

Mirabail, M. (1990). La culture informatique. *Aster : Recherches en didactique des sciences expérimentales*, *11*(1), 11-28. https://doi.org/10.4267/2042/9117 (cf. p. 1)

Modeste, S. (2012). La pensée algorithmique : apports d'un point de vue extérieur aux mathématiques. *Enseignement Des Mathématiques et Contrat Social : Enjeux et Défis pour le 21e Siècle – Actes du Colloque EMF 2012*, 467-479 (cf. p. 14, 28).

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch : Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46), 1-23 (cf. p. 21, 47, 159).

Nogry, S. (2020). Des objets pour apprendre (mémoire HDR). (Cf. p. 181).

Palts, T., & Pedaste, M. (2020). A model for developing computational thinking skills. *Informatics in Education*, *19*(1), 113-128. https://doi.org/10.15388/infedu.2020.06 (cf. p. 19, 58)

Papadopoulos, Y., & Tegos, S. (2012). Using microworlds to introduce programming to novices. *2012 16th Panhellenic Conference on Informatics*, 180-185. https://doi.org/10.1109/PCi.2012.18 (cf. p. 45)

Papert, S. (1980). *Mindstorms : children, computers, and powerful ideas*. Basic Books, Inc. (Cf. p. 11, 17, 31, 45, 46, 354).

Papert, S. (1981). *Jaillissement de l'esprit : ordinateurs et apprentissage* (Flammarion). (Cf. p. 26).

Papic, M. (2007). Promoting Repeating Patterns with Young Children - More Than Just Alternating Colours ! *Australian Primary Mathematics Classroom*, *12*(3), 8. https://doi.org/10.3316/informit.135529953955178 (cf. p. 53, 54, 62-64, 230, 231)

Pattis, R. E. (1981). *Karel the robot : a gentle introduction to the art of programming* (1st edition). John Wiley & Sons, Inc. (Cf. p. 46).

Payne, S. J. (1991). Interface problems and interface resources. *Designing interaction : Psychology at the human-computer interface*, 128-153. https://doi.org/10.5555/120352.120360 (cf. p. 91)

Pelánek, R. (2017). Bayesian knowledge tracing, logistic models, and beyond : an overview of learner modeling techniques. *User Modeling and User-Adapted Interaction*, *27*(3), 313-350. https://doi.org/10.1007/s11257-017-9193-2 (cf. p. 137, 138, 159, 357)

Pelánek, R. (2018). Exploring the utility of response times and wrong answers for adaptive learning. *Proceedings of the fifth annual ACM conference on learning at scale*, 1-4. https://doi.org/10.1145/3231644.3231675 (cf. p. 159, 167, 169, 170, 173)

Pelánek, R., & Effenberger, T. (2020). Design and analysis of microworlds and puzzles for block-based programming. *Computer Science Education*, *32*(1), 1-39. https://doi.org/10.1080/08993408.2020.1832813 (cf. p. 45, 46, 48, 49, 104, 105, 107, 108, 110, 112, 114, 118, 119, 128-131, 139)

Peter, Y., Léonard, M., & Secq, Y. (2019). Reconnaissance de Motifs et Répétitions : Introduction à la Pensée Informatique. *Environnements Informatiques pour l'Apprentissage Humain*. https://hal.science/hal-02151035/ (cf. p. 16)

Peter, Y., Secq, Y., & Léonard, M. (2020). Reconnaissance de motifs redondants et répétitions : introduction à la Pensée Informatique. *STICEF (Sciences et Technologies de l'Information et de la Communication pour l'Éducation et la Formation)*, *27*(2). https://doi.org/10.23709/sticef.27.2.3 (cf. p. 2)

Piaget, J. (1935). *La naissance de l'intelligence chez l'enfant*. Delachaux et Niestlé Neuchatel-Paris. (Cf. p. 74, 76, 77, 84, 88).

Piaget, J. (1974a). *La prise de conscience.* Presses universitaires de France. (Cf. p. 133).

Piaget, J. (1974b). *Réussir et comprendre*. Presses universitaires de France. (Cf. p. 88, 133).

Planas, E., & Cabot, J. (2020). How are UML class diagrams built in practice? A usability study of two UML tools : Magicdraw and Papyrus. *Computer Standards & Interfaces*, *67*, 103363. https://doi.org/10.1016/j.csi.2019.103363 (cf. p. 132)

Polledo, E., Garaizar, P., & Guenaga, M. (2021). Lempel : Developing the pattern recognition skill in computational thinking through an online educational game. *Learning Analytics Summer Institute Spain 2021*, 28-37. https://ceur-ws.org/Vol-3029/ (cf. p. 56, 341)

Poulakis, E., & Politis, P. (2021). Computational thinking assessment : literature review. In S. Demetriadis, A. Mikropoulos, V. Dagdilelis & T. Tsiatsos (Éd.), *Research on e-learning and ICT in education* (p. 111-128). Springer International Publishing. https://doi.org/10.1007/978-3-030-64363-8_7. (Cf. p. 21)

Rabardel, P. (1995). *Les hommes et les technologies ; approche cognitive des instruments contemporains*. Armand Colin. https://hal.science/hal-01017462. (Cf. p. iii, iv, 5, 6, 50, 71, 84-91, 94, 97, 190, 344)

Rabardel, P. (2005). Instrument, activité et développement du pouvoir d'agir. *Entre connaissance et organisation : l'activité collective*, 251-265 (cf. p. 85, 87, 88).

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., & Silverman, B. (2009). Scratch : programming for all. *Communications of the ACM*, *52*(11), 60-67. https://doi.org/10.1145/1592761.1592779 (cf. p. 42)

Resnick, M., & Siegel, D. (2015). A different approach to coding. *International Journal of People-Oriented Programming*, *4*(1), 1-4 (cf. p. 47).

Reuter, Y., Cohen-Azria, C., Daunay, B., Delcambre, I., & Lahanier-Reuter, D. (2013). *Dictionnaire des concepts fondamentaux des didactiques* (T. 3e éd.). De Boeck Supérieur. https://doi.org/10.3917/dbu.reute.2013.01. (Cf. p. 10, 23)

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017). K-8 Learning Trajectories Derived from Research Literature : Sequence, Repetition, Conditionals. *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 182-190. https://doi.org/10.1145/3105726.3106166 (cf. p. 34, 59, 60, 293)

Rieber, L. P. (1996). Seriously considering play : Designing interactive learning environments based on the blending of microworlds, simulations, and games. *Educational technology research and development*, *44*(2), 43-58. https://doi.org/10.1007/BF02300540 (cf. p. 45)

Rittle-Johnson, B., Zippert, E. L., & Boice, K. L. (2019). The roles of patterning and spatial skills in early mathematics development. *Early Childhood Research Quarterly*, *46*, 166-178. https://doi.org/10.1016/j.ecresq.2018.03.006 (cf. p. 54)

Rix-Lièvre, G., & Lièvre, P. (2012). La dimension «tacite» des connaissances expérientielles individuelles : une mise en perspective théorique et méthodologique. *Management international/International Management/Gestiòn Internacional*, *16*, 21-28. https://doi.org/10.7202/1012390ar (cf. p. 132)

Robert, A., & Rogalski, J. (2002). Le système complexe et cohérent des pratiques des enseignants de mathématiques : une double approche. *Canadian*

Journal of Math, Science & Technology Education, *2*(4), 505-528. https://doi.org/10.1080/14926150209556538 (cf. p. 355)

Roche, M., Higuera, C. d. l., & Michaut, C. (2018). Enseigner la programmation informatique : comment réagissent les professeurs des écoles ? *Notes du CREN*. https://hal.science/halshs-01713729/ (cf. p. 2)

Rogalski, J. (1988a). Acquisition de structures conditionnelles. *Annales de didactique et de Sciences cognitives*, *1*, 131-152 (cf. p. 82).

Rogalski, J. (1988b). Les représentations mentales du dispositif informatique dans l'alphabétisation. In *Actes du premier colloque Franco-Allemand de didactique des mathématiques et de l'informatique*, *La Pensée Sauvage* (C. Laborde). (Cf. p. 16, 89, 92, 198, 339).

Rogalski, J. (2015). Psychologie de la programmation, didactique de l'informatique, déja une histoire... In *Informatique en éducation : perspectives curriculaires et didactiques* (B. Drot-Delange, G-L. Baron, E. Bruillard, p. 280-305). Presses Universitaires Blaise-Pascal. (Cf. p. 10, 28, 29).

Rogalski, J. (1985). Alphabétisation informatique - problèmes conceptuels et didactique. *Bulletin de l'APMEP*, *347*, 61-74 (cf. p. 335).

Rogalski, J. (1987). Acquisition de savoirs et de savoir-faire en informatique. *Cahiers de Didactique des Mathématiques*, *43* (cf. p. 14, 80, 82, 313, 314, 339, 352, 353).

Rogalski, J., & Samurçay, R. (1986). Les Problèmes cognitifs rencontrés par des élèves de l'enseignement secondaire dans l'apprentissage de l'informatique. *European Journal of Psychology of Education*, *1*(2), 97-110 (cf. p. 31, 32, 218, 280, 282, 298, 345, 355).

Rogalski, J., & Vergnaud, G. (1987). Didactique de l'informatique et acquisitions cognitives en programmation. *Psychologie française*, *32.4* (cf. p. 12, 14, 29, 74, 78, 79, 94, 96, 315, 334, 352).

Romero, M., Duflot-Kremer, M., & Viéville, T. (2018). Le jeu du robot : analyse d'une activité d'informatique débranchée sous la perspective de la cognition incarnée. *Review of science, mathematics and ICT education*. https://doi.org/10.26220/rev.3089 (cf. p. 27)

Romero, M., Lille, B., Viéville, T., Duflot-Kremer, M., Smet, C. d., & Belhassein, D. (2018). Analyse comparative d'une activité d'apprentissage de la programmation en mode branché et débranché. https://hal.science/hal-01861732/ (cf. p. 43)

Rouchier, A. (1990). Objets de savoir de nature informatique dans l'enseignement secondaire. *Aster : Recherches en didactique des sciences expérimentales*, *11*(1), 29-44. https://doi.org/10.4267/2042/9118 (cf. p. 31, 32, 269)

Samurçay, R., & Rouchier, A. (1985). De «faire» à «faire faire» : planification d'actions dans la situation de programmation. *Enfance*, *38*(2), 241-254. https://doi.org/10.3406/enfan.1985.2883 (cf. p. 31)

Saxena, A., Lo, C. K., Hew, K. F., & Wong, G. K. W. (2020). Designing unplugged and plugged activities to cultivate computational thinking : an exploratory study in early childhood education. *The Asia-Pacific Education Researcher*, *29*(1), 55-66. https://doi.org/10.1007/s40299-019-00478-w (cf. p. 56, 57, 225)

Selby, C., & Woollard, J. (2013). *Computational thinking : the developing definition*. University of Southampton. (Cf. p. 20).

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158. https://doi.org/10.1016/j.edurev.2017.09.003 (cf. p. 57-59)

Spach, M. (2017). *Activités robotiques à l'école primaire et apprentissage de concepts informatiques : quelle place du scénario pédagogique? Les limites du co-apprentissage* (thèse de doct.). Université Sorbonne Paris Cité. (Cf. p. 26, 81, 211).

Tang, J. C., Liu, S. B., Muller, M., Lin, J., & Drews, C. (2006). Unobtrusive but invasive : using screen recording to collect field data on computer-mediated interaction. *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, 479-482. https://doi.org/10.1145/1180875.1180948 (cf. p. 132)

Tanimoto, S. L. (1998). Connecting middle school mathematics to computer vision and pattern recognition. *International journal of pattern recognition and artificial intelligence*, *12*(8), 1053-1070. https://doi.org/10.1142/S0218001498000592 (cf. p. 56)

Tchounikine, P. (2009). Précis de recherche en ingénierie des EIAH. https://hal.science/hal-00413694/ (cf. p. 45, 116, 117)

Tchounikine, P. (2017). Initier les élèves à la pensée informatique et à la programmation avec Scratch. *Laboratoire d'informatique de Grenoble* (cf. p. 46).

Theureau, J. (2010). Les entretiens d'autoconfrontation et de remise en situation par les traces matérielles et le programme de recherche «cours d'action». *Revue d'anthropologie des connaissances*, *4*(2), 287-322. https://doi.org/10.3917/rac.010.0287 (cf. p. 133-135, 142, 143)

Tijus, C.-A., & Cordier, F. (2003). Psychologie de la connaissance des objets. Catégories et propriétés, tâches et domaines d'investigation. *L'année psychologique*, *103*(2), 223-256. https://doi.org/10.3406/psy.2003.29635 (cf. p. 135, 136, 147, 181)

Toll, D., & Wingkvist, A. (2018). Visualizing Programming Session Timelines. *Proceedings of the 11th International Symposium on Visual Information*

Communication and Interaction, 106-107. https://doi.org/10.1145/3231622.3232506 (cf. p. 160)

Touloupaki, S., & Baron, G.-L. (2019). Apprendre à programmer à l'école primaire ? Une approche exploratoire en cycle 2. In *Le numérique à l'école primaire. Pratiques de classe et supervision pédagogique dans les pays francophones* (S. Nogry, L. Boulc'h, F. Villemonteix). Presses Univ. Septentrion. https://hal.science/hal-04135648. (Cf. p. 294)

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 (cf. p. 48)

Vaníček, J., Dobiáš, V., & Šimandl, V. (2023). Understanding loops : What are the misconceptions of lower-secondary pupils ? *Informatics in Education*, *22*(3), 525-554 (cf. p. 35, 269, 277, 358).

Vergnaud, G. (2003). Pourquoi parler autant de conceptualisation. *Actes du colloque Conceptualisation et surdité Synthèse des travaux, Suresnes* (cf. p. 157).

Vergnaud, G. (1991). La théorie des champs conceptuels. In *Recherches en didactique des mathématiques*. La Pensée Sauvage. (Cf. p. iii, iv, 3, 4, 6, 36, 71, 74-76, 78-81, 84, 85, 89, 91, 94, 96, 97, 120-123, 135, 153, 154, 157-161, 167-169, 181, 210, 214, 215, 219, 221, 223, 230, 236, 254, 258, 259, 293, 302, 318, 357).

Vergnaud, G. (1996). Au fond de l'action, la conceptualisation. In *Savoirs théoriques et savoirs d'action* (J.M. Barbier, p. 275-292). Presses Universitaires de France. (Cf. p. 75, 77, 79, 157).

Vergnaud, G. (2007). Représentation et activité : deux concepts étroitement associés. *Recherches en éducation*, *4*, 9-22. https://doi.org/10.4000/ree.3889 (cf. p. 76, 293)

Vergnaud, G. (2010). La forme opératoire et la forme prédicative de la connaissance (cf. p. 77).

Vergnaud, G. (2012). Forme opératoire et forme prédicative de la connaissance. *Investigações em Ensino de Ciências*, *17*(2), 287-304 (cf. p. 74-76, 157, 293).

Vergnaud, G. (2013a). Pourquoi la théorie des champs conceptuels ? *Journal for the Study of Education and Development*, *36*(2), 131-161. https://doi.org/10.1174/021037013806196283 (cf. p. 177)

Vergnaud, G. (2013b). Qu'est-ce que la pensée ? *La nouvelle revue de l'adaptation et de la scolarisation*, *N° 63*(3), 277-299. https://doi.org/10.3917/nras.063.0277 (cf. p. 76, 78, 150, 168, 327, 329)

Vergnaud, G., & Durand, C. (1976). Structures additives et complexité psychogénétique. *Revue française de pédagogie*, *36*, 28-43. https://doi.org/10.3406/rfp.1976.1622 (cf. p. 153-156, 238)

Vermersch, P. (2019). *L'entretien d'explicitation*. ESF Sciences humaines. (Cf. p. 133-135, 143).

Vidal-Gomel, C., Bourmaud, G., & Munoz, G. (2022). Gérard Vergnaud et Pierre Rabardel, entre filiation et débats, une contribution à la didactique professionnelle. *Bulletin de psychologie*, *75*(4), 333-346. https://doi.org/10.3917/bupsy.578.0333 (cf. p. 84)

Vygotsky, L. (1985). Pensée et langage (trad. F. Sève). *Paris, Sociales* (cf. p. 84).

Walgenwitz, G., Croset, M.-C., & Beffara, E. (2024). Modélisation des connaissances mobilisées dans une situation de généralisation de motif. *Colloque Didapro 10 sur la Didactique de l'informatique et des STIC*, 159-159. https://hal.science/hal-04485409/ (cf. p. 341)

Warren, E. (2005). Patterns supporting the development of early algebraic thinking. *Building connections : Research, theory and practice*, *2*, 759-766 (cf. p. 54).

Warren, E., & Cooper, T. (2006). Using Repeating Patterns to Explore Functional Thinking. *Australian Primary Mathematics Classroom*, *11*(1), 9. https://doi.org/10.3316/informit.170932599345758 (cf. p. 53, 54, 61, 62)

Warren, E., & Cooper, T. (2007). Repeating Patterns and Multiplicative Thinking : Analysis of Classroom Interactions with 9 -Year- Old Students that Support the Transition from the Known to the Novel. *The Journal of Classroom Interaction*, *41/42*(2), 7-17 (cf. p. 54).

Warren, E., & Miller, J. (2010). Exploring Four Year Old Indigenous Students' Ability to Pattern. *International Research in Early Childhood Education*, *1*(2), 42-56. https://doi.org/10.4225/03/5821092e32c6d (cf. p. 61-64)

Warren, E., Miller, J., & Cooper, T. (2012). Repeating patterns : strategies to assist young students to generalise the mathematical structure. *Australasian Journal of Early Childhood*, *37*(3), 111-120. https://doi.org/10.1177/183693911203700315 (cf. p. 53, 61, 64, 69, 231)

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, *62*(8), 22-25. https://doi.org/10.1145/3341221 (cf. p. 39)

Weintrop, D., & Wilensky, U. (2018). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education*, *18*(1), 1-25. https://doi.org/10.1145/3089799 (cf. p. 39)

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215 (cf. p. 1, 17, 19, 21, 22)

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A : Mathematical, Physical*

and Engineering Sciences, *366*(1881), 3717-3725. https://doi.org/10.1098/rsta.2008.0118 (cf. p. 19)

Xu, W., Wu, Y., & Ouyang, F. (2023). Multimodal learning analytics of collaborative patterns during pair programming in higher education. *International Journal of Educational Technology in Higher Education*, *20*(1), 8. https://doi.org/10.1186/s41239-022-00377-z (cf. p. 179, 212)

Zapata-Cáceres, M., Martín-Barroso, E., & Román-González, M. (2020). Computational Thinking Test for Beginners : Design and Content Validation. *2020 IEEE Global Engineering Education Conference (EDUCON)*, 1905-1914. https://doi.org/10.1109/EDUCON45650.2020.9125368 (cf. p. 21)

Zhang, L., & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, *141*, 103607. https://doi.org/10.1016/j.compedu.2019.103607 (cf. p. 22, 33, 42)

Annexes

Étant donnés la nature et le nombre de pages que représentent les annexes à cette recherche, nous avons fait le choix de les mettre à disposition uniquement au format numérique. Les documents annexes sont mentionnés et accessibles au fil du texte. Le présent chapitre regroupe tous les liens vers le dépôt sur Nextcloud de l'université de Lille. Vous trouverez aussi dans ce chapitre une brève description de chacun de ces documents.

Annexe 1 : description de la base de données bebras-chticode

Cette annexe contient la description des champs de la table extraite de la base de données bebras-chticode, ainsi qu'une visualisation d'un court extrait de cette table.

Accéder à l'annexe

Annexe 2 : autorisation parentale scolaire

Le lecteur trouvera dans cette annexe le document qui a été transmis aux enseignants qui ont accepté de faire participer leur classe à l'étude. Ce document a été transmis aux parents via l'ENT de l'établissement.

Accéder à l'annexe

Annexe 3 : autorisation parentale individuelle

Les parents des participants à l'échelle du sujet ont rempli et m'ont retourné l'autorisation parentale de participation à l'étude disponible dans cette annexe.

Accéder à l'annexe

Annexe 4 : traitements de données pour l'échelle nationale

Cette annexe est constituée d'un export au format pdf du notebook jupyter qui a été utilisé pour le traitement des données de l'échelle nationale.

Accéder à l'annexe

Annexe 5 : étude de la robustesse des données pour l'échelle nationale

Cette annexe contient quelques simulations, afin de déterminer la robustesse des données collectées à l'échelle nationale.

Accéder à l'annexe

Annexe 6 : préparation des données pour l'échelle des classes

Cette annexe est constituée d'un export au format pdf d'un notebook jupyter. Celui-ci permet d'effectuer automatiquement toutes les étapes de préprocessing à partir d'un fichier de données brutes, en lançant une série de fonctions python.

Accéder à l'annexe

Annexe 7 : traitements relatifs à la détermination des profils pour l'échelle des classes

Dans cette annexe se trouve le détail des traitements réalisés pour le calcul de l'indicateur de rapidité normalisée et la détermination des profils. Le lecteur y trouvera notamment les essais réalisés en amont du choix du seuil de la valeur de cet indicateur, seuil qui détermine la distinction entre certains profils.

Accéder à l'annexe

Annexe 8 : étude des programmes exécutés pour l'échelle des classes

Nous avons regroupé dans cette annexe tous les traitements quantitatifs qui concernent l'étude des programmes exécutés à l'échelle des classes : catégorisation sommaire automatique, visualisations des programmes exécutés, processus d'appui à la catégorisation manuelle des erreurs.

Accéder à l'annexe

Annexe 9 : analyses préalables des extraits vidéos pour l'échelle du sujet

Dans cette annexe sont regroupés le travail d'analyse préalable ainsi que les liens vers les extraits vidéo mentionnés dans le document de thèse.

Accéder à l'annexe

Annexe 10 : traitements relatifs aux premiers apprentissages de la boucle

Cette annexe contient les traitements quantitatifs réalisés pour étudier les premiers apprentissages de la boucle dans le chapitre 12.

Accéder à l'annexe

Annexe 11 : traitements relatifs à l'étude des résolutions expertes

Dans cette annexe se trouvent les traitements effectués après le codage manuel des résolutions expertes à partir des enregistrements vidéo.

Accéder à l'annexe

Annexe 12 : analyse a priori des planches pour la catégorisation de grilles

Dans cette annexe, le lecteur trouvera l'ensemble des planches utilisées pour la tâche de catégorisation de grilles. Chaque planche est accompagnée d'une

courte analyse *a priori*.

Accéder à l'annexe

Annexe 13 : transcription des entretiens relatifs à la catégorisation des grilles

Cette annexe contient un aperçu du travail ébauché pour la tâche de catégorisation des situations, dont la finalisation constitue une perspective à court terme.

Accéder à l'annexe

Table des figures

Liste des tableaux