



HAL
open science

Generation of Adapted Training Game Activities : a Model-Driven Engineering Design and Implementation Framework

Bérénice Lemoine

► **To cite this version:**

Bérénice Lemoine. Generation of Adapted Training Game Activities : a Model-Driven Engineering Design and Implementation Framework. Technology for Human Learning. Le Mans Université, 2024. English. NNT : 2024LEMA1013 . tel-04763449

HAL Id: tel-04763449

<https://theses.hal.science/tel-04763449v1>

Submitted on 2 Nov 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT

DE
LE MANS UNIVERSITÉ

SOUS LE SEAU DE
LA COMUE ANGERS – LE MANS

ÉCOLE DOCTORALE N° 641
*Mathématiques et Sciences et Technologies
de l'Information et de la Communication*
Spécialité : *Informatique*

Par

Bérénice LEMOINE

**Generation of Adapted Training Game Activities: a Model-Driven
Engineering Design and Implementation Framework**

Thèse présentée et soutenue à LAVAL, le 27/09/2024

Unité de recherche : Laboratoire d'Informatique de l'Université du Mans (LIUM)

Thèse N° : 2024LEMA1013

Rapporteurs avant soutenance :

Sophie DUPUY-CHESSA Professeure, Université Grenoble-Alpes
Karim SEHABA Maître de conférence et HDR, Université Lumière Lyon 2

Composition du Jury :

Président :	Marianne HUCHARD	Professeure, Université de Montpellier
Examineurs :	Sophie DUPUY-CHESSA	Professeure, Université Grenoble-Alpes
	Karim SEHABA	Maître de conférences et HDR, Université Lumière Lyon 2
	Amel YESSAD	Maitresse de conférences, Sorbonne Université
	Marianne HUCHARD	Professeure, Université de Montpellier
Dir. de thèse :	Sébastien GEORGE	Professeur, Le Mans Université
Encadrant :	Pierre LAFORCADE	Maître de conférences, Le Mans Université

ACKNOWLEDGEMENT

Je tiens à exprimer ma profonde reconnaissance à toutes les personnes ayant contribué, de près ou de loin, à la réalisation de mes travaux de recherche. Votre soutien, votre bonne humeur, votre aide et votre intérêt pour ma recherche ont décuplé ma motivation et ont créé un environnement propice à la finalisation de cette thèse.

Tout d'abord, je souhaite exprimer ma gratitude envers les personnes qui m'ont encouragé à entreprendre cette thèse. Parmi ces personnes, je tiens particulièrement à remercier Professeur Violaine Prince. Sa gentillesse, son soutien, ses conseils et la confiance qu'elle m'a accordée, même lorsque je doutais de moi, m'ont permis d'aller aussi loin. Je souhaite également remercier Professeur Marianne Huchard, qui a été mon enseignante, mon encadrante de stage, et qui est aujourd'hui l'examinatrice de mes travaux. Sa bienveillance et ses conseils m'ont encouragé à poursuivre cette thèse. La confiance et les encouragements de ma famille m'ont également guidé vers cette réalisation, mais leurs remerciements viendront plus tard ! 😊

J'adresse également mes remerciements à mon comité de suivi, Amel Yessad et Christophe Desprès, pour leurs précieux retours tout au long de cette thèse, qui ont permis d'améliorer et d'approfondir mes travaux. Un grand merci à l'ensemble des membres de mon jury, plus précisément aux deux rapporteurs, Sophie Dupuy-Chessa et Karim Sehaba, et aux deux examinatrices, Amel Yessad et Marianne Huchard.

J'ai eu la chance de bénéficier d'un encadrement sans faille tout au long de cette thèse. Merci Sébastien pour tes conseils avisés, ta bienveillance et tes remarques toujours constructives. Je suis certain que tous tes doctorants s'accorderaient pour dire que tu es un excellent directeur de thèse. Pierre, je ne suis pas sûre d'avoir les mots pour exprimer toute ma gratitude à ton égard. Merci pour tout ! Merci de m'avoir guidée dans l'approfondissement de la recherche et dans sa conduite. Merci de m'avoir soutenue tout au long de cette thèse, malgré mon caractère bien trempé. Ta gentillesse, ton humour (quelque peu particulier 😊), et nos discussions ont été essentielles pour mieux me comprendre et ont grandement contribué à mon développement personnel.

Un grand merci à l'ensemble de mes collègues au CERIU² dont la bonne humeur a créé une ambiance agréable tout au long de la thèse : Ibtissem, Sebastian, Hamza, Albane,

Vincent, Nail, Valériane, Mamoudou, Manith, Jordi, Dalal, Jean, Mohamed, Wassim, Amine. Je n'oublie pas non plus nos stagiaires de passage, qu'ils soient informaticiens ou biologistes : Maysa, Billy, Clément, Théo, Valentin, Martin, Alkhali, Rova, et Lucie. Plus particulièrement, un grand merci à Ibtissem, Sebastian et Hamza. Votre présence et nos rires durant la première année de thèse ont rendu cette expérience inoubliable. Sebastian, mon Sebinou, on a commencé nos thèses ensemble et on les aura terminées ensemble (à quelques jours/semaines près). Un binôme parfait ! Je retiendrai avec vous tous nos blagues, nos chants, nos danses, nos sorties et nos déplacements. Merci d'avoir partagé ces moments mémorables qui ont enrichi cette expérience.

Je tiens à remercier mes amis du sud : Anaïs, Sélène, Anthony, Thomas, Lisa, Marine, Fanny. Vous m'avez toujours soutenue et encouragée. Sans nos voyages, rires, jeux, appels vidéos, mon quotidien aurait été bien moins intéressant. Plus particulièrement, merci à Antho, Sésé, Thomas, Fanny pour notre petit voyage à Châteauroux l'année dernière, qui m'a permis d'être avec vous. Docteur Anaïs, un grand merci d'être là au quotidien et pour tous les moments passés à Caen pendant ta thèse, qui ont rendu la distance plus facile à gérer.

Enfin, ces remerciements ne sauraient être complets sans une pensée pour ma famille et mes proches, qui sont là au quotidien pour me soutenir, m'encourager et croire en moi. Merci à mes grand-parents, mon oncle et ma tante, mes cousins (Tristan et Tanguy), ma maman, mon beau-père et ses enfants (Paula et Maxime). Paula, un grand merci pour tout le temps que tu as passé avec moi, nos appels ont rendu la distance beaucoup moins pesante. Maman, je ne te remercierais jamais assez pour ton soutien, tes encouragements, ton amour, tes relectures de mes manuscrits et articles en français. Merci pour tout ! Je vous aime ♥

Un grand merci à toutes les personnes dont les noms n'apparaissent pas dans ces quelques lignes, et qui m'ont soutenu sur le plan social, émotionnel et à bien d'autres égards.

Enfin, ce travail n'aurait pas été possible sans l'aide financière des collectivités locales mayennaise qui m'a permis de me consacrer sereinement à ces travaux de thèse, et pour laquelle je suis très reconnaissante.

TABLE OF CONTENTS

List of Figures	9
List of Tables	13
Acronyms	15
1 Introduction	17
1.1 Research Context	17
1.1.1 Research Laboratory	17
1.1.2 AdapTABLES Project	18
1.2 Research Problem	19
1.3 Thesis Structure	22
I Research Background Towards Adaptation and Games	25
2 Adaptation in TEL	27
2.1 Definitions	28
2.2 Characterisation of Adaptation	30
2.3 Existing Work	32
2.3.1 Adaptation in TEL	32
2.3.2 Adaptation in Educational Games	33
2.3.3 Approaches Guiding Adaptation	34
2.4 Synthesis & Discussion	34
3 Games & Content Generation	37
3.1 Definition of Procedural Content Generation	38
3.2 Games & Serious Games Design	39
3.3 Existing Work	40
3.3.1 Content Generation in Video Games	40
3.3.2 Content Generation in TEL	42
3.4 Synthesis & Discussion	46
4 Research Issue	49
4.1 Research Questions	50
4.2 Positioning	52
4.2.1 Roguelite Game Genre	52
4.2.2 Adaptations & Variety	54

4.2.3	Model-Driven Engineering	55
4.3	Research Method & Evaluation	56
II	Design and Implementation Framework of Generators	59
5	Design Framework of Activity Generators	61
5.1	General Overview	61
5.2	Definition: Game Activity for DK Training	63
5.3	Declarative Knowledge Training Elements	65
5.3.1	Training Path	66
5.3.2	Training Task Types	67
5.3.3	Training Tasks Parameters	67
5.4	Roguelite-oriented Game Elements	68
5.4.1	Analysis method for Roguelite Design	69
5.4.2	Design Choices for Activity Generation	70
5.4.3	Gameplay Categories	72
5.5	Synthesis	74
6	Mapping Game and Educational Elements	77
6.1	Existing Work	78
6.1.1	Relations Between Dimensions	78
6.1.2	Methods to Define Relations Between Dimensions	79
6.1.3	Synthesis	80
6.2	Mapping Approach Development	80
6.2.1	Identification of the <i>Pivot</i>	81
6.2.2	Mapping Task Types onto Gameplay Categories	84
6.3	A Systematic Mapping Approach	88
6.3.1	Proposed Mapping Approach	88
6.3.2	Relations Between Task Types and Gameplay Categories	90
6.3.3	Evaluation of the Relations	90
6.4	Synthesis	92
7	Conceptual Design Approach	93
7.1	Conceptual Models for Activity Generation	94
7.1.1	Domain Model: Training and Knowledge	95
7.1.2	Game Model	97
7.1.3	Activity Model	99
7.1.4	Learner-Player Model	100
7.1.5	Relation Model	102
7.1.6	Synthesis & Discussion	103
7.2	Mapping Questioned Facts with Game Elements	104
7.2.1	Generic Modelling of Questions about Facts	105

7.2.2	Modelling Gameplays Descriptions	107
7.2.3	Generic Generation of Varied Task-oriented Gameplays	108
7.3	Synthesis	111
8	Software Infrastructure	113
8.1	Model-Driven Engineering Foundations	113
8.1.1	Conceptual Models to Computerised Metamodels	114
8.1.2	Models as Inputs and Outputs of Generation	118
8.2	Activity Generation Algorithm	119
8.2.1	Algorithm for Generating Training Game Activities	119
8.2.2	Algorithm for Generating Questions about Facts	122
8.3	Extension Rules	123
8.4	Synthesis	125
III	Application & Evaluation	127
9	Extensions of the Framework	129
9.1	Generator for Multiplication Tables Training	130
9.2	Generator for History-Geography Facts Training	134
9.3	Generator for Judo Facts Training	137
9.4	Generator for Solar System Facts Training	141
9.5	Discussion	144
10	Tests and Validation of the Framework	147
10.1	Framework Properties Evaluation through Tests	148
10.1.1	Learner Adaptation of the Generated Activities	149
10.1.2	Player Adaptation of the Generated Activities	152
10.1.3	Variety of the Generated Activities	153
10.2	Validation of Static Properties of Models	156
10.3	Framework Evaluation with an Engineer	157
10.4	Use of a Generator in Ecological Conditions	160
10.5	Synthesis	161
11	Conclusion	163
11.1	Synthesis	163
11.1.1	Contributions to TEL Research Domain	165
11.1.2	Limitations	166
11.2	Perspectives	166
	Bibliography	171
A	Analysis of Existing Games for Multiplication Tables Training	191

TABLE OF CONTENTS

B	Gameplay Mock-Ups Evaluation Questionnaire	194
C	Algorithm for Generic Generation of Task-oriented Gameplays	237
D	XMI to XML Code Transformation in ETL	239
E	Guidelines for Extending the Framework	244
F	JUnit Test Method Example	266
G	Model Validation Source Code (EVL)	268
H	Framework Usability Evaluation Questionnaire	272

LIST OF FIGURES

1.1	Usage view in AdapTABLES.	18
1.2	Research problem illustration	21
1.3	Outline of the manuscript	23
2.1	Spectrum of adaptation in computer systems of Wilson and Scott (2017)	29
2.2	Tripartite structure of adaptive instruction of Vandewaetere et al. (2011)	30
2.3	High level illustration of the adaptation process of a system	31
3.1	General illustration of the content generation process	38
3.2	MDA framework order of influence (Hunicke et al. 2004)	39
3.3	Scenario generator architecture (GOALS) of Sehaba and Hussaan (2013)	44
3.4	The 3x3 metamodel-based architecture of Laforcade and Laghouaouta (2018)	45
4.1	General position of our research problem	50
4.2	Examples of dungeons maps and rooms from existing commercial <i>Roguelites</i>	53
4.3	MDE levels of abstraction (models, metamodels, meta-metamodels) (Brambilla et al. 2012)	56
4.4	Our research method	57
5.1	Coarse-grained components of a training game activity generator	62
5.2	Design Framework Overview	63
5.3	Levels of activity, as defined in activity theory (Carvalho et al. 2015)	64
5.4	Description of the training structure	66
5.5	AdapTABLES game flow	70
5.6	Example of mock-ups by gameplay categories	73
5.7	Illustration of the levels of proposed activities from activity theory	74
5.8	Overview of exchanges made with experts to specify training and gameplays elements (in orange : exchanges described in Chapter 6, in blue : exchanges described in Chapter 9)	75
6.1	Illustration of the educational-game dimensions <i>mapping</i> research question	78
6.2	General idea to map task types onto gameplay categories	81
6.3	Mapping between task types and exercises illustration	85
6.4	Division of gameplay categories illustration ((S) = Single, (M) = Multiple)	86
6.5	Mapping between task types and gameplay categories illustration	88
6.6	Proposed Mapping Approach	89
6.7	Conditional relations between task types and gameplay categories	90
6.8	Examples of possible solutions	91

LIST OF FIGURES

7.1 Interconnected conceptual models involved in activity generation 94

7.2 Conceptual domain model describing knowledge and training path 95

7.3 Conceptual game model 97

7.4 Conceptual activity/dungeon model 100

7.5 Conceptual learner-player model 101

7.6 Transformation process of raw facts into questioned facts 101

7.7 Conceptual relation (between task types and gameplay categories) model . 102

7.8 Illustration of questioned facts to game elements mapping problem 103

7.9 General idea behind the concept of generic questioned facts 104

7.10 Conceptual modelling of generic questioned facts 105

7.11 Examples of questions about facts in generic form 106

7.12 Focus and detail of the game conceptual model 107

7.13 Gameplays with structures (orange dashed borders) per fact (up-left) /
per proposals (up-right) / per statement (bottom-left) / per visualisation
(bottom-right) 108

7.14 Example of generated positioned elements from a questioned fact, game
elements and a gameplay description 109

7.15 Examples of generated gameplays, based on the same questioned facts but
different gameplay descriptions 110

8.1 Interconnected models conform to metamodels involved in activity generation 114

8.2 Knowledge metamodel 115

8.3 Training metamodel 115

8.4 Game metamodel 116

8.5 Learner-player metamodel 116

8.6 Relation metamodel 117

8.7 Activity metamodel (to be generated) 117

8.8 Tree-based EMF model view of models with properties on the selected node 118

8.9 Principle of extension for questions on facts generations 119

8.10 Activity generation algorithm steps 120

8.11 Step-by-step example of the generation algorithm. Puzzle pieces colours cor-
respond to Figure 8.1 and puzzle pieces with borders present data modified
or created by the algorithm 121

8.12 Structure of the template method design pattern ([Shvets 2018](#)) 122

8.13 Design framework and generators components overview 125

9.1 Overview of the evaluation of the framework through proof-of-concept . . . 129

9.2 Extension (in blue) of the metamodels for multiplication tables 130

9.3 Tree-based EMF views of mathematic models 131

9.4 Examples of gameplays for multiplication tables training tasks interpreted
by the *game engine* 133

9.5 Extension (in green) of the metamodels for history-geography facts 135

9.6 Tree-based EMF views of history-geography models 136

9.7	Examples of gameplays for history-geography facts training tasks interpreted by the <i>game engine</i>	137
9.8	Extension (in cyan) of the metamodels for judo facts	138
9.9	Tree-based EMF views of judo models	139
9.10	Examples of gameplays for judo facts training tasks interpreted by the <i>game engine</i>	141
9.11	Extension (in violet) of the metamodels for solar system facts	141
9.12	Tree-based EMF views of solar system models	142
9.13	Examples of gameplays for solar system facts training tasks interpreted by the <i>game engine</i>	143
10.1	Overview of the overall evaluation of the framework	148
10.2	Training path and learner's progress used for testing the objective/level pair selection	149
10.3	Training paths and learner-player's progress used for testing the allocation of the tasks	151
10.4	Player's game preferences for testing based on the equipments unlocking abilities used in our <i>game engine</i>	153
10.5	Maps of four generated dungeons for the same objective/level pair and learner-player	154
10.6	Example of two variants of the same gameplay	155
10.7	An example of model validation rule written in EVL that verifies whether the facts associated to an objective belong to the knowledge model associated to the path	156
10.8	Experimentation of the framework with an engineer	158
10.9	Spanish metamodel created by an engineer familiar with the framework	159
10.10	Spanish knowledge model created by an engineer familiar with the framework	159
10.11	Use of the AdapTABLES in ecological conditions	160

LIST OF TABLES

2.1	Key concepts for characterising the adaptation of a TEL system	32
2.2	First characterisation of the adaptation of our system	35
3.1	Comparative table of different generation works in TEL (🎓 = educational dimension, 🎮 = game dimension)	47
4.1	Final characterisation of the adaptation of our system	55
5.1	Parameters for multiplication tables	68
5.2	Grid for the Design Needs Analysis of educational Roguelite games	69
5.3	Design choices for AdapTABLES	71
6.1	Exercises by quiz format (✓ present; ✗ absent; present but incomplete) .	83
6.2	Characterisation of the exercises	84
6.3	Characterisation of the task types	85
6.4	Characterisation of the gameplay categories ((S) = Single, (M) = Multiple)	87
8.1	Summary of the creation and use of the models required for activity generation (✗ = not modified, ✓ = modified, — = can be modified, but should not be)	124
A.1	Analysis of existing games for Multiplication Table training	191

ACRONYMS

AHS Adaptive Hypermedia Systems.
.....

DK Declarative Knowledge.

DNB Diplôme National du Brevet des Collèges.
.....

EMF Eclipse Modeling Framework.

ETL Epsilon Transformation Language.

EVL Epsilon Validation Language.
.....

HCI Human Computer Interaction.
.....

IEIAH Ingénierie des Environnements Informatiques pour l'Apprentissage Humain.

ITS Intelligent Tutoring Systems.
.....

LIUM Laboratoire d'Informatique de l'Université du Mans.

LST Language and Speech Technology.
.....

MDE Model-Driven Engineering.
.....

PCG Procedural Content Generation.
.....

TEL Technology-Enhanced Learning.
.....

UML Unified Modeling Language.
.....

XMI XML Metadata Interchange.

XML Extensible Markup Language.

INTRODUCTION

Contents

1.1	Research Context	17
1.1.1	Research Laboratory	17
1.1.2	AdapTABLES Project	18
1.2	Research Problem	19
1.3	Thesis Structure	22

This PhD thesis is a contribution to Technology-Enhanced Learning by means of Model-Driven Engineering. This chapter presents our research context, outlines the research problem tackled over the last three years, and details this manuscript structure.

1.1 Research Context

The following section presents our research context by describing the research laboratory, where our research has been carried out, and the AdapTABLES project which served as a case study and an experimental ground for this thesis.

1.1.1 Research Laboratory

This research has been conducted within the IEIAH team (*Ingénierie des Environnements Informatiques pour l'Apprentissage Humain*) of the LIUM laboratory (*Laboratoire d'Informatique de l'Université du Mans*). The LIUM¹ is a computer science laboratory with two major research themes, which are Technology-Enhanced Learning (TEL) and Language and Speech Technology (LST). The IEIAH team main objective concerns the elaboration of a scientific basis for the development and engineering of TEL systems. This objective structures the team's actions, which are organised around three axes:

- **Design, Operationalisation, and Adaptation of Pedagogical Situations:** a main focus is on the design of TEL systems that integrates teachers and educators in the process.
- **Modelling Observation and Analysing Traces:** this axis focuses on analysing learning situations based on users' traces (i.e., teachers or learners) in TEL systems.
- **Advanced and Collaborative Interactions for Learning:** this axis addresses advanced interactions for learning using technologies such as interactive tablets, mixed reality or tangible interfaces.

1. <https://lium.univ-lemans.fr/>

This PhD falls within the first axis as it addresses the design of a framework aimed at the generation of adapted training activities. This work is connected to the **AdapTABLES** project presented below.

1.1.2 AdapTABLES Project

The **AdapTABLES** project² is a research project led by Pierre Laforcade. The main educational objective is to design and develop an educational game for the long-term acquisition of multiplication tables. The practice of multiplication tables is intended to complement classroom learning (from the teacher’s viewpoint): learning the tables, applying them to problem-solving and generalising them are beyond the scope of the educational game. The second objective concerns the design of a web-app (i.e., authoring tool) enabling teachers to specify needs and monitor each learner’s progress.

In this project, mathematics experts (i.e., 8 teachers from 2nd to 6th grade and a didactician) as well as two game designers are involved as stakeholders in the design process of the educational game. These experts participate in the definition of needs and the validation of proposals. The project is based on an iterative prototyping approach.

This project provides a research context for the creation of adapted game activities (i.e., precise and detailed descriptions of the task that the learner-player has to perform, in terms of objectives, instructions, resources and criteria for success). This project served as a case study (i.e., ground for expressing and developing needs) and an experimental ground (i.e., validating proposals) for this thesis. Figure 1.1 presents the usage view in **AdapTABLES** (i.e., learners interact with the educational game while teachers interact with the web-app).

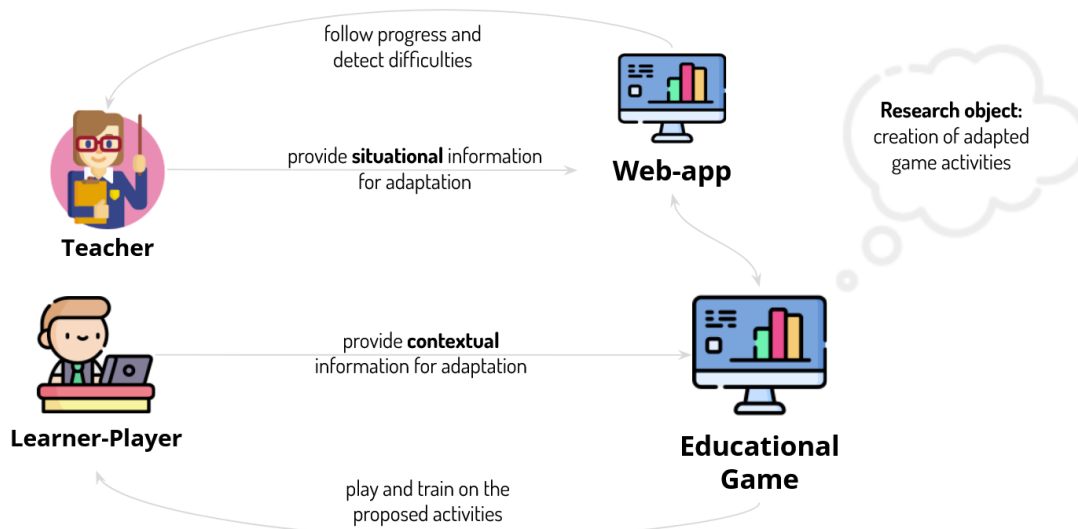


Figure 1.1 – Usage view in AdapTABLES.

2. <https://projets-lium.univ-lemans.fr/adaptables/>

1.2 Research Problem

Declarative Knowledge (DK) is one of the required knowledge to perform a task. Anderson and Lebiere (2014) defined it as the knowledge of “things we are aware we know and can usually describe to others”. DK consists of factual information such as multiplication tables, historical dates, or geographical data. Repetition is necessary to encourage the retention³, and generalisation of DK (Kim *et al.* 2013).

In cognitive psychology, Test-Enhanced Learning represents the idea that the process of remembering concepts or facts (i.e., retrieving them from memory) increases their long-term retention. Retrieval Practice is a strategy of Test-Enhanced Learning, consisting of repeated recalls of what has been learned, usually involving low-stakes and no-stakes writing prompts, quizzes, flashcards, etc. This strategy has been proven to significantly improve long-term retention (Brame and Biel 2015; Roediger and Pyc 2012). Furthermore, evidence suggests that various test formats enhance learning (i.e., the benefits are not linked to a specific strategy) (Brame and Biel 2015). Therefore, DK training can be seen as a form of retrieval practice that entails repeatedly asking learners various questions about facts to foster their long-term retention.

Repetition can easily become boring for learners (Smith 1981). Lately, the design and use of *educational games* has become a common practice to make this kind of activity more attractive (Codish and Ravid 2015). Serious game can be defined as “(digital) games used for purposes other than mere entertainment” (Susi *et al.* 2007). Educational games⁴ are serious games with an educational purpose (i.e., learning, training/practice). More precisely, Li *et al.* (2024) defined *digital educational games* as “interactive activities, facilitated by electronic devices, designed with an educational purpose”. Additionally, these digital games have been proven to improve learners’ motivation and engagement in comparison to traditional learning settings (*ibid.*). Consequently, the use of digital games for DK training seems relevant.

However, research has shown that learner-players can quickly feel bored by educational games that offer repetitive activities with challenges that are not tailored to their skills and knowledge (Streicher and Smeddinck 2016). Such feelings could potentially lead to the drop-out of the task, thereby negatively impacting learning. Therefore, to reduce boredom, repetitive game activities must be adapted to learner-players. Moreover, most digital educational games are not perceived as real video games, mainly because of their lack of gameplay (i.e., game elements are masked by educational aspects) (Kiili 2005; Prensky 2005). Gameplay can be defined as “the fun things that the player gets to decided, control, and do” (Prensky 2005). Therefore, digital educational games must provide varied activities, in terms of gameplays, to be seen as real video games.

Many existing online education games are designed to train DK (e.g., multiplication table, geographical data). Within the AdapTABLES project, several educational games (i.e.,

3. The ability to store or retain information in the memory over a certain period of time. Whereas memorisation refers to the process by which information is intentionally stored in memory.

4. In the literature, the term learning games can also be found to describe games having a learning purpose.

computer/online and phone/tablet applications) for training multiplication tables have been found on *Google* and *Play Store*. Twenty-two games have been tested to assess their gameplay and possible educational and gaming adaptations. Appendix A presents a table describing the analysis of these twenty-two games.

These games are often designed to merely present questions for learner-players to answer, accompanied by game mechanics such as time pressure, rewards, scores and currencies. A few exceptions introduce more advanced gameplay elements and interactions, such as platform game mechanics where players control an avatar that must jump to make choices. Educational choices are often limited to selecting tables to work on or selecting a difficulty level (i.e., easy, medium, difficult). The difficulty levels⁵ mainly have an impact on the response time allowed or the tables to be worked on. Additionally, game choices concerns elements that do not impact the training activities.

Multiplication training games analysis observations

1. Educational settings are mostly reduced to the choice of table(s) ($\frac{11}{22}$ only have this setting, $\frac{16}{22}$ have at least this setting);
2. Few games ($\frac{7}{22}$) offer to choose between 2 or 3 difficulty levels, which can impact the response time allowed, or the tables worked on in the activity. However, these levels may have different meanings that are not clearly explained.
3. Only a few games propose to change some game settings ($\frac{10}{22}$). Some settings are only cosmetic features that do not impact the core activity (e.g., changing the avatar), others ($\frac{5}{22}$) such as the difficulty level or the number of players do impact the core activity.
4. Few games ($\frac{7}{22}$) offer data persistence. The data stored differs between games. Some games store badges, money earned or items purchased. Others, divided into levels, save the current level. But none of them give access to a detailed summary of the progress (e.g., past mistakes, seen tables) or the learners' knowledge.

Although the analysed existing digital games lack these characteristics, educational games aimed at DK training should provide *activities*⁶ that are: 1) varied in terms of questions and gameplays, and 2) adapted to learner-players to reduce the feeling of boredom. However, creating game activities requires game design skills (i.e., designing game

5. Adapting the game using difficulty levels can be seen as a means of asking players to classify themselves within a predefined low-resolution stereotype which may not be appealing to players who do not how to choose or do not identify with any (Lopes and Bidarra 2011a).

6. Broadly defined here as game situations targeting a training objective.

situations) that teachers may not possess. Additionally, designing a variety of activities involves building several versions of every activity for each learner. This is a time-consuming and demanding task that teachers can not realise manually.

Definition: DK Training through Digital Games

Repeatedly providing learner-players with **varied** and **adapted** game activities which are **questioning facts**.

Generation or procedural generation is a technique for automatic content creation (e.g., game level, story, dialogue) using structured data and rules by means of algorithms based on pseudo-randomness. This technique is widely used in role-playing games, particularly to design varied game levels. Well-known games such as *Minecraft*, *Diablo*, *Rogue Legacy*, *Hades*, and *The Binding of Isaac* are based on this principle. Generating content describing a formalised training game activity is a possible solution to offer varied and adapted activities. However, generation has rarely been addressed in TEL (Bezza *et al.* 2013).

Our work focuses on *research in engineering* for the design and development of activity generators for DK training. It consists of an exploratory research aiming at better characterising these generators (research object) and proposing models, tools and techniques to facilitate their design and implementation. Designing such generators is a complex task that cannot be reduced to a mere computer engineering problem (Tchounikine *et al.* 2009), since their specification and implementation require several experts: educational game developers (i.e., design choices, use of technologies), didactic domain experts (i.e., facts to be worked on, how to work on them, adaptations choices) and video game experts (i.e., game knowledge, game design...). More precisely, our general question is: **How to guide the design and implementation of activity generators for Declarative Knowledge training?** Figure 1.2 illustrates our research problem.

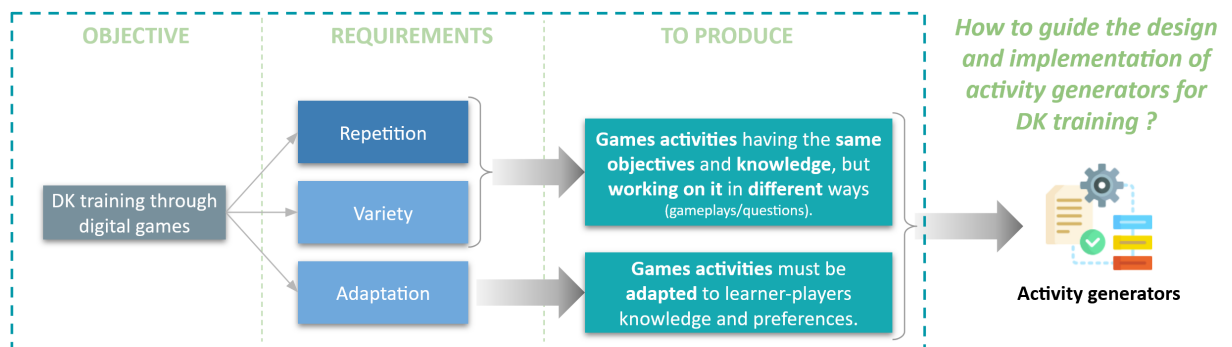


Figure 1.2 – Research problem illustration

In order to address our question, it is necessary to examine the design of game and training activities, the alignment required to build training game activities and the specification of requirements for automating this design. Accordingly, a state-of-the-art on

game and serious game design, on the adaptation of a system in TEL, and on content generation in video games and in TEL has been realised.

Our resulting proposal is an extensible framework (i.e., software infrastructure) based on Model-Driven Engineering principles (Brambilla *et al.* 2012) that guides the design and implementation of domain specific (i.e., didactic domain) game activity generators. Model-Driven Engineering (MDE) is a research domain supporting an active use of models during the entire software development process, enabling the automated generation of the final application. The use of MDE to support the specification of the elements required to drive the generation of learning game scenarios adapted to learners has already been successfully demonstrated (Laforcade and Laghouaouta 2018). Additionally, MDE has also been used efficiently in the field of Human Computer Interaction (HCI) for the generation of user interfaces (Sottet *et al.* 2007).

1.3 Thesis Structure

As depicted in Figure 1.3, this manuscript is structured in three parts (excluding Introduction and Conclusion): Part I – Research Background Towards Adaptation and Games, Part II – Design and Implementation Framework of Generators, as well as Part III – Application & Evaluation.

Part I is structured in three chapters. This part describes the research background. Chapter 2 presents and characterises adaptation in TEL. Chapter 3 presents a state-of-the-art on game content generation in TEL. Chapter 4 clearly defines our research issue and positions our work in regard to the observations made from the literature.

Part II is structured in four chapters. This part presents our contribution: an extensible framework, based on a Model-Driven Engineering approach, for the design and implementation of generators of Roguelite-oriented and adapted game activities for DK training. Chapter 5 introduces the framework and its components. Chapter 6 presents the issue of mapping games and educational elements that has arisen during the design of activities, as well as our proposed solution. Chapter 7 describes the conceptual aspects of the framework (i.e., conceptual models, algorithm). Chapter 8 presents the computerised aspects of the framework (i.e., computerised models, technical aspects of the algorithms).

Part III is structured in two chapters. This part presents the usage and evaluation of our framework. Chapter 9 describes a proof-of-concept of our framework by presenting several didactic domain extensions (i.e., creation of activity generators for several didactic domains using the framework). Chapter 10 presents tests and experiments realised to evaluate the framework.

In conclusion, the work carried out, and the contributions made are synthesised, and several perspectives for future work are proposed.

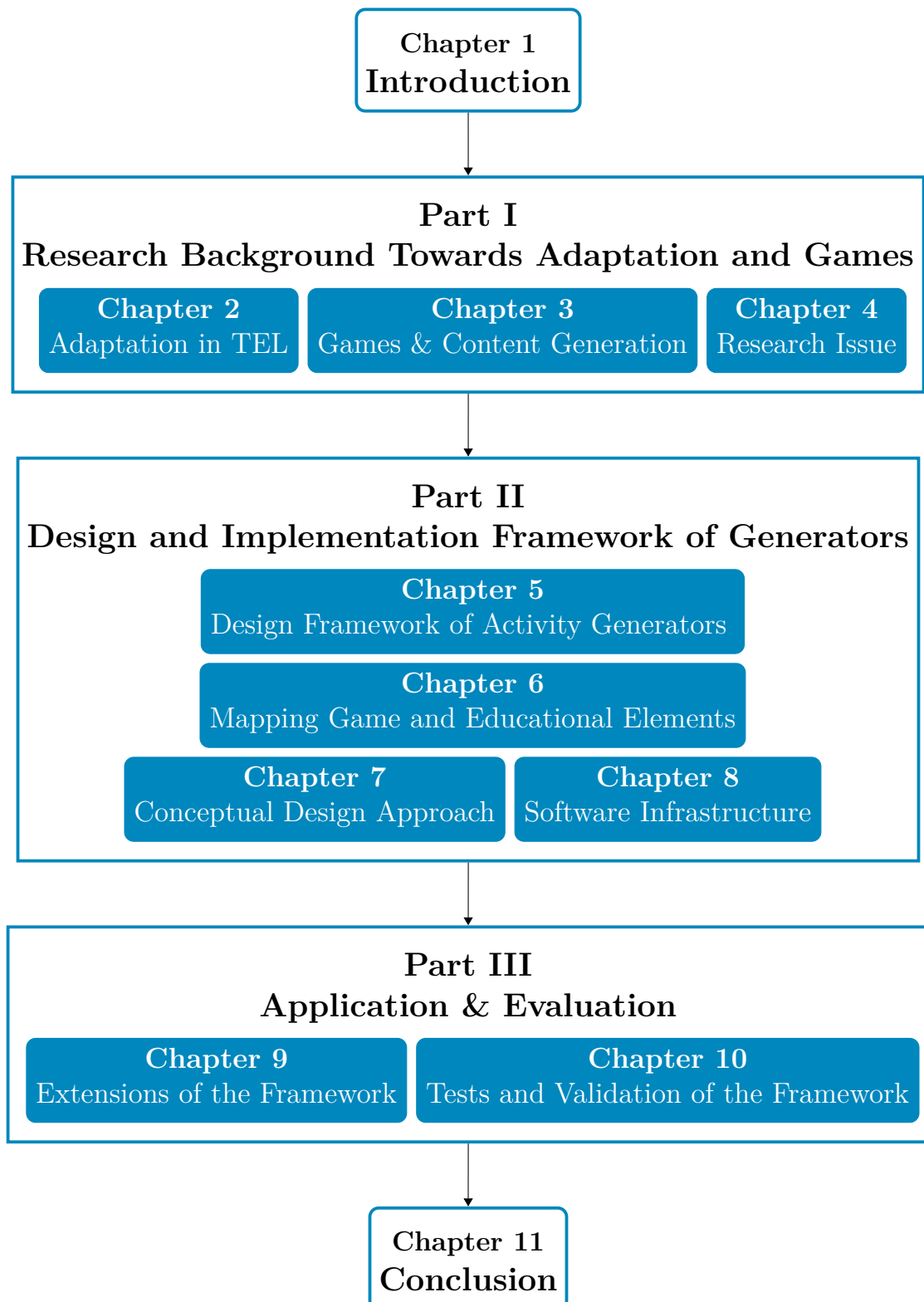


Figure 1.3 – Outline of the manuscript

PART I

Research Background Towards Adaptation and Games

ADAPTATION IN TEL

Contents

2.1	Definitions	28
2.2	Characterisation of Adaptation	30
2.3	Existing Work	32
2.3.1	Adaptation in TEL	32
2.3.2	Adaptation in Educational Games	33
2.3.3	Approaches Guiding Adaptation	34
2.4	Synthesis & Discussion	34

Everybody is a genius. But if you judge a fish by its ability to climb a tree,
it will live its whole life believing that it is stupid.

— ANONYMOUS ^a

^a. Famous quote that perfectly illustrates the importance to consider individual needs as everyone is different.

Long before the rise of digital learning, teachers and researchers paid a great interest in tailoring learning activities to learners' needs. A key reason is that providing each learner with an experience that meets their needs at all times was considered the ultimate goal (Plass and Pawar 2020). Such individualised instruction have been demonstrated as beneficial compared to the *one-size-fits-all* approaches (Vandewaetere *et al.* 2011).

Recently, our digital era, with powerful computers and the World Wide Web, has given rise to specific types of adaptive learning systems such as Intelligent Tutoring Systems (ITS) or Adaptive Hypermedia Systems (AHS) (Wilson and Scott 2017). ITS are computer-based systems that imitate human tutors (i.e., “intelligent” tutor) and dynamically provide tailored instructions or feedbacks to learners. AHS are web-based environments that can provide user-adapted elements (Brusilovsky 1998). When correctly designed, these systems have been proven to be highly effective for learning (Fletcher 1999; Wilson and Scott 2017).

For many years, video games and educational games have become increasingly popular in the research community. Therefore, researchers are expressing great interest in adapting educational games to learner-players (Göbel and Wendel 2016; Sajjadi *et al.* 2022). Numerous works suggest that educational games are effective learning tools (De Freitas 2018). In addition, educational games have been proven to improve motivation (Li *et al.* 2024). However, some researchers argue that a lack of adaptation could lead to a loss of motivation, predictability, or non-replayability (Lopes and Bidarra 2011b).

This chapter aims at characterising adaptation in TEL and to present an initial positioning of our work in terms of adaptation. First, it defines adaptation in Technology-Enhanced Learning. Then, this chapter suggests a way for characterising the adaptation of TEL systems. Next, it discusses existing work on adaptation (i.e., How? What? On what basis?) and how to guide adaptation. Finally, it summarises the requirements for system adaptation and presents a first positioning of the adaptation of our system.

2.1 Definitions

Broadly, adapting a system is the action that consists of tailoring it, in part or in full, to one or more users. Adaptation is a research issue addressed in various fields such as educational sciences, computer science, HCI or TEL. In the TEL literature, adaptation can be found under a wide range of terms such as adaptability, adaptivity, personalisation, customisation, individualisation, and so on. This variety of terms can be an obstacle to the progress of research on adaptation¹, as there is no consensus on the definitions (Shemshack and Spector 2020).

Streicher and Smeddinck (2016) define adaptability as “the fact that a system is not fixed, but can be changed (to the needs of users, to changing environmental contexts, etc.; changes are usually understood to be performed manually)”. Whereas Plass and Pawar (2020) state that adaptable systems must “respond to the diagnosis of specific learner variables and corresponding needs of a learner by providing the learner with individualized options and choices for how to proceed, putting the control of the learning in the individual’s hands”.

Guettat *et al.* (2010) state that customising Interactive Learning Environments (ILE) “requires consideration of several specific items related to the learning process. They must also include the characteristics of users (learners or tutors), the tasks or the problems of the ILE”. While Streicher and Smeddinck (2016) define customisation as “the act of changing a system to the needs of a user group or individual user (manually or automatically; may can be done by the group itself or by the user him- or herself, but may also be done by third parties; often related to the appearance or content of the given system)”.

Bakkes *et al.* (2012) define personalised games as “a game that utilises player models for the purpose of tailoring the game experience to the individual player”. While Streicher and Smeddinck (2016) define personalisation as “the act of changing a system to the needs of a specific individual user (often automatic but does not have to be, i.e., can be understood as a specific form of customization with a focus on individuality; personalization is also often related to appearance or content)”. Moreover, Ismail and Belkhouche (2018) define personalised systems as “systems that tailor learning resources accesses within the software environment to a user model”.

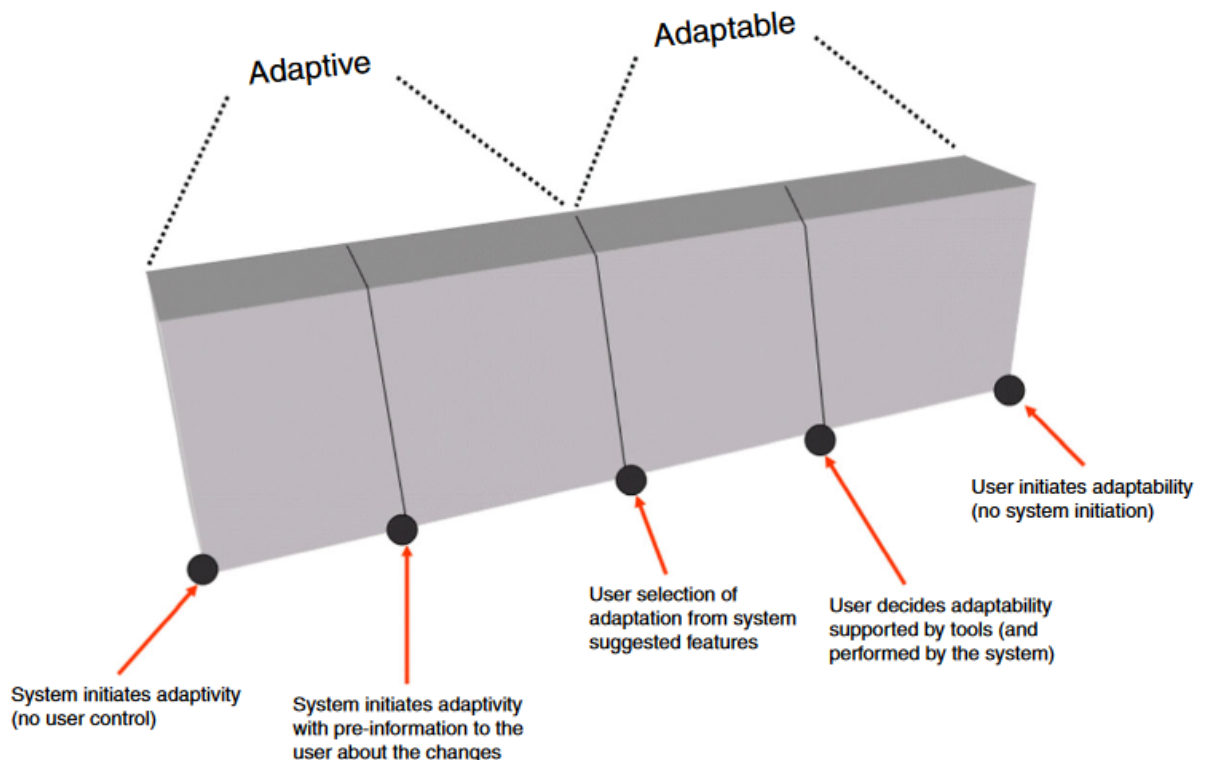
Sajjadi *et al.* (2022) characterise individualisation as “tailoring the learning game to

1. In the field of HCI, adaptation also called plasticity can be implemented through adaptable, adaptive or mixed approaches. However, there is a lack of definition on how to implement adaptation (i.e., components required) (Miraz *et al.* 2021).

the individual's needs, state, abilities, and preferences". While [Ćurčić et al. \(2018\)](#) state that individualisation through computer software "provides the pupil with the opportunity to become aware of learning goals, to master the strategies of studying, recognize the achieved goals and estimate the potentials for new accomplishments".

[Plass and Pawar \(2020\)](#) define adaptivity as "the ability of a learning system to diagnose a range of learner variables, and to accommodate a learners' specific needs by making appropriate adjustments to the learner's experience with the goal of enhancing learning outcomes". Whereas [Streicher and Smeddinck \(2016\)](#) define adaptivity as "the fact that a system is not fixed, but dynamically changes over time (to adjust to the needs of users or an individual user, or to adjust to changing environmental contexts, etc.; typically happens automatically; often related to settings and parameters present in the given system)".

Observably, all the definitions are quite similar. Sometimes, terms are even defined on the basis of others, for example, the U.S. Department of Education defined personalisation as encompassing individualisation ([Education 2010](#)). When clearly specified, the differences among these definitions concerns: how adaptation is performed (e.g., manually, automatically), what data is used to adapt (e.g., users' characteristics, tasks' characteristics), and what content is to be adapted (e.g., appearance, content, environment).



Source: Based on [Oppermann et al. \(1997\)](#)

Figure 2.1 – Spectrum of adaptation in computer systems of [Wilson and Scott \(2017\)](#)

An interesting vision is provided by [Oppermann and Rashev \(1997\)](#), who considers adaptation as a spectrum ranging from adaptability (i.e., “systems that allow the user to change certain system parameters and adapt their behaviour accordingly”) to adaptivity (i.e., “systems that adapt to the users automatically based on the system’s assumptions about user needs”). [Wilson and Scott \(2017\)](#) built on [Oppermann and Rashev \(1997\)](#)’s work by softening the opposition between adaptability and adaptivity, viewing each term as two areas of the spectrum rather than two extremes (see Figure 2.1). Their vision seems more consistent with existing work and the nuances found in the definitions of the TEL literature².

2.2 Characterisation of Adaptation

Adaptation can be implemented in several ways. Additionally, it can be aimed at one or more targets (e.g., game preferences, learning content, difficulty). Therefore, adaptive systems (i.e., systems that adapt to the users) are often characterised, see Figure 2.2, by three concepts ([Vandewaetere et al. 2011](#)):

- the target, i.e., *what is adapted?*
- the source, i.e., *what does it adapt to?*
- the pathways, i.e., *how to translate source into target?*

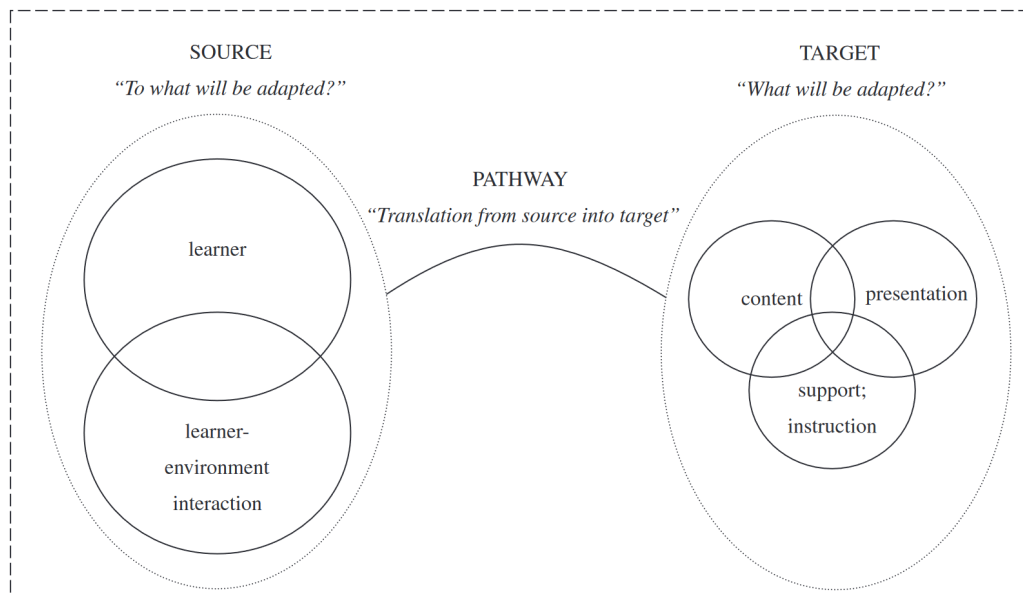


Figure 2.2 – Tripartite structure of adaptive instruction of [Vandewaetere et al. \(2011\)](#)

2. Note that although the HCI literature distinguishes three categories of adaptation, namely adaptability, adaptivity and mixed approaches ([Miraz et al. 2021](#)), given that a mixed approach can be implemented in many ways, it is a mean to consider adaptation as a spectrum.

The sources of adaptation are modelled data such as player and learner profiles, preferences, pedagogical strategies, user traces, etc. Sources may be directly collected from experts or users by filling in questionnaires, providing a “profile” estimated to be close to the player (e.g., Hexad (Tondello *et al.* 2016), BrainHex (Nacke *et al.* 2014)) or the learner (e.g., Big Five Factor Model (Goldberg 1992)), or they can be extracted from user traces of the system. Source information can rely on one or several implicit dimensions such as didactic, pedagogy, game, cognition, etc. Accordingly, the system must take many source information into account. Each dimension adds a different adaptation objective: improving learning, motivating the player, motivating the learner, and so on. Several elements can be the targets of adaptation (i.e., objects targeted by adaptation), such as the content (e.g., activities, pedagogical resources), the presentation³ (e.g., feedbacks provided, HCI, UI⁴, HUD⁵, sounds), the navigation (e.g., ordering resources = scenarios). Finally, various methods exist to adapt sources to targets such as generation, selection, recommendation, parameterisation, assembling⁶. Figure 2.3 describes the main elements characterising the adaptation process of a TEL system.

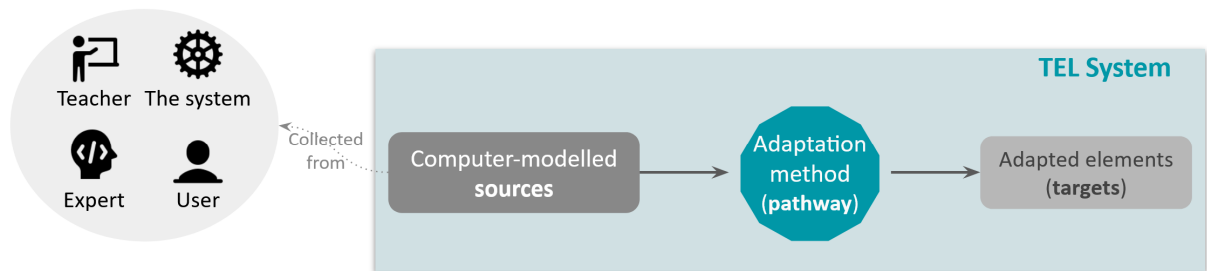


Figure 2.3 – High level illustration of the adaptation process of a system

In addition to these three concepts, adaptation should be characterised by its *automation level* (e.g., manual, automated): Is the target adaptation process performed strictly by humans? Do some parts of the process depend on human intervention, while others are computerised? Is the process completely automated (i.e., carried out entirely by a computer system)? A final key element to characterise a system’s adaptation is the moment when the adaptation is performed/required. Are the targets previously adapted to the sources (e.g., recommendation of activity sequences based on learner’s characteristics)? Are the targets adapted “at-runtime” (i.e., targets are adapting while the users interact with the system)? Are the targets previously adapted to the sources but “adaptive” in-between sessions (e.g., learning game activities are provided to learner-players each time considering previous results)? Table 2.1 presents an overview of the required concepts for characterising the adaptation of a TEL system. Using several concepts to characterise adaptation is not foreign to TEL, and has already been applied by [Sehaba](#)

3. The presentation is often the target in the field of HCI.

4. User Interface

5. Head Up Display

6. These methods can be implemented at different levels: strategic, tactical, operational.

(2014) who characterises his work on adaptation through multiple criteria: why adapt? What to adapt? Who adapts? When to adapt? What to adapt to? How to adapt?

	EXAMPLES
SOURCES	User profile User behaviour Experts strategies
TARGETS	Content Instructions Feedbacks Presentation
PATHWAYS	Generation Selection Recommendation Assembling
AUTOMATION LEVEL	Manual Semi-automated Automated
MOMENT	Previously adapted Adapted during runtime

Table 2.1 – Key concepts for characterising the adaptation of a TEL system

2.3 Existing Work

Numerous works have addressed adaptation in TEL, either to offer adaptive (i.e., adaptable, personalised, individualised, etc.) systems (e.g., educational games, interfaces, gamified systems) or to assist in the adaptation process (e.g., authoring tools, methods, approaches guiding the design of adaptive systems).

2.3.1 Adaptation in TEL

Many studies propose methods or tools to recommend ordered sequences of pedagogical content adapted to learners. [Klašnja-Milićević et al. \(2011\)](#) (*pedagogical* adaptation) proposes a recommendation module (*pathway*) which, based on a learner model including their knowledge and learning style (*sources*), automatically (*automation level*) provides an ordered sequence of activities (*target*) to the learners once they are logged in (*moment*). The learner can modify the activities order but not the activities themselves. [Lefevre et al. \(2012\)](#) (*educational*⁷ adaptation) propose the PERSUA2 model used to recommend (*pathway*) sequences of activities (*target*) adapted to learners on the basis of the learner’s profile, the teacher’s strategies and the context of use (*sources*). At each request (*moment*), a fixed sequence of activities is automatically created from the input data (*automation level*). [Sablayrolles et al. \(2022\)](#) (*pedagogical* adaptation) propose a recommendation approach (*pathway*) of ordered sequence of relevant resources (*target*) based on a learner’s competency profile, a mastery level, a learning objective, a competency framework modelled by experts and a set of constraints (*sources*). As with PERSUA2, at each request (*moment*) a fixed sequence of activities is automatically created from the input data (*automation level*). [Pham et al. \(2016\)](#) (*pedagogical & visual* adaptations) propose PACARD (Personalise Adaptive CARD-based interface) a card-based system that selects (*pathway*) the intervals between cards reviews (card sequencing) and their presentation

7. Adaptation considering pedagogical and didactic elements to adapt.

(*target*, i.e., interface objects containing educational content and with which users interact) on the basis of learner performance (i.e., recorded in databases), learning strategies (i.e., spaced repetition and forgetting curve) and interface preferences (e.g., how many cards, which type per session) (*sources*). Every session (*moment*), a new sequence of card is automatically provided (*automation level*).

Further work focuses on the adaptation of gamification systems⁸ in order to improve motivation. Gamification can be defined as the integration of game elements into learning environments to encourage learners' motivation (Deterding *et al.* 2011). As an example, Monterrat *et al.* (2014) (*gaming* adaptation) present an adaptive gamification system that can be plugged into learning environments. This system recommends (*pathway*) game elements (*target*, e.g., edges, chat, rewards) using a player model (*source*) including information on the interactions, the environment (i.e., school/ work/personal, device used, group size), and the user (e.g., age, gender). Each time the learning environment is used (*moment*), game elements are automatically recommended to the learner and the player model is updated (*automation level*).

2.3.2 Adaptation in Educational Games

Some studies have focused on designing or helping in the design of adapted game content. Marne and Labat (2014) (*gaming* adaptation) propose an authoring tool enabling teachers (*pathway*) to “manually” design (*automation level*) learning game scenarios (*target*) that adapt according to players' answers/interactions. Scenarios are built from activities whose objectives and prerequisites (i.e., input/output states) are predefined (*sources*). Once created, scenarios (*moment*) can be exported in XML format and interpreted by compatible games. Soflano *et al.* (2015) (*pedagogical* adaptation) propose an adaptive conversation system between players and Non-Player Characters (NPCs) using learner-players' learning style (*source*) to select (*pathway*) the presentation (i.e., text or image/diagram) of task instructions (*target*). Two systems have been implemented. One system where the learning style is predefined beforehand (*moment*), resulting in the instructions being set automatically (*automation level*). Another system where players can change the presentation of the instructions (i.e., learning styles) during execution (*moment, automation level*).

Further research has proposed systems to select or recommend game content tailored to users. Natkin *et al.* (2007) (*gaming* adaptation) propose a quest recommendation system (*pathway*) for MUG Systems (Multi-player Ubiquitous Game). Based on a user model (*source*), a set of quests (*target*) is proposed to the player each time a level is requested (*moment*). Based on the choices made and a trace analysis, the user model is modified, and the suggested quests are automatically refined (*automation level*). Bontchev *et al.* (2021) (*educational & gaming* adaptations) present a student modelling approach applied to the

8. Educational game design entails the joint creation of game and learning content, whereas gamification involves adding a layer of game elements to existing learning content/environments. For this reason, our choice has been to consider gamification independently of educational games.

adaptation of learning game content. Their proposal have been implemented in educational mazes containing four types of mini-games, namely question, searching, arranging, and action games. At each game session (*moment*), the puzzle types and complexity, type and content of the learning material (*target*) are dynamically selected (*pathway, automation level*) according to the three-part student model (*source*, i.e., composed of player’s characteristics, learner’s characteristics, and user’s characteristics) concerned.

2.3.3 Approaches Guiding Adaptation

Some studies have focused on proposing approaches that are sufficiently generic to guide adaptation, or models that can take several dimensions into account when recommending adapted content. [Montserrat *et al.* \(2017\)](#) propose MAGAM (Multi-Aspect Generic Adaptation Model), a multi-aspect activity recommendation model (i.e., the ability to consider several dimensions) based on three entities: user-learners, pedagogical activities and properties linking users and activities (i.e., represented in the form of matrices). As a result, the system provides a recommendation matrix describing how well each activity is adapted to each learner. [Roepke *et al.* \(2021\)](#) provide a modular, component-based architecture for implementing custom pipelines for educational games in anti-phishing training. Their definition of a pipeline for adapting learning games is a three-step process: data collection, content generation, content delivery. These pipelines are intended to “precede a game and provide adaptations to gameplay or the configuration of a game”. Although the concept of the pipeline is only applied in the context of anti-phishing games, it seems broad enough to be reused in the general design of other games. [Ismail and Belkhouche \(2018\)](#) propose a reusable architecture for the design of personalised learning software systems, broken down into four units: the learner unit which stores learner data, the knowledge unit which stores learning resources, the personalisation unit which matches the learner model to the learning resources and the presentation unit which represents the software environment. These works propose generic enough guidelines to be followed in the high-level design of any adaptive learning system.

2.4 Synthesis & Discussion

This chapter presented the various terms and definitions of adaptation present in the literature. Since adaptation can be considered as a spectrum, **what is important** is not the term used, but **the way in which adaptation is characterised**. What is adapted? What sources are used to adapt? How are the targets adapted to the sources? At what moment? Is the process automatised? Etc.

Additionally, an observation based on existing work is that sources are modelled data either supplied by humans or collected from interactions between humans and systems. Accordingly, **source modelling is a requirement** to adapt a system.

Another observation is that **few works seem to adapt on several dimensions simultaneously** ([Bontchev *et al.* 2021](#); [Montserrat *et al.* 2017](#); [Pham *et al.* 2016](#)). In

the context of educational games, adaptation seems to either focus on game adaptation (i.e., narration, quests, game elements) or educational adaptation (i.e., activities, order, sequences) but rarely considers both (Bontchev *et al.* 2021; Monterrat *et al.* 2017). However, adapting activities according to learners is recognised as a way of improving learning. Furthermore, adapting to players helps motivate and make the tasks more engaging. Some research even stated that the “impact of gameplay, in terms of engagement and learning, depends on players’ individual differences (i.e., gaming proficiency, personality, preferences, and emotional state)” (Abdul Jabbar and Felicia 2015). It would therefore seem relevant to consider adaptation from both an educational and a gaming dimension.

Finally, **most researches are oriented towards recommending or selecting adapted content** to user and not building content that is adapted to the users (e.g., composing an activity like a product in a software product line, procedurally generating an activity). **Our interest lies in automatically** (*automation level*) **generating** (*pathway*) **adapted training**⁹ **game content** (*target*). To that extent, information about the learner-player and the training are (*sources*) required to generate adapted content. In our context, adaptation is realised during the generation process, which produce content that is adapted to learner-players. Table 2.2 summarises the first characterisation of our adaptation using the five criteria previously identified (i.e., sources, targets, pathways, automation level, and moment).

SOURCES	Learner-Player and Training data
TARGETS	Training game content
PATHWAYS	Procedural generation
AUTOMATION LEVEL	Generator automatically provides adapted content
MOMENT	Each time the generator is called

Table 2.2 – First characterisation of the adaptation of our system

In order to better position our work, it is necessary to examine existing works in terms of generation. Accordingly, the next chapter addresses content generation in video games and in TEL.

9. As mentioned in the introduction, training refers to the action of providing learner-players with repeated but varied and adapted game activities which question facts (i.e., declarative knowledge).

GAMES & CONTENT GENERATION

Contents

3.1	Definition of Procedural Content Generation	38
3.2	Games & Serious Games Design	39
3.3	Existing Work	40
3.3.1	Content Generation in Video Games	40
3.3.2	Content Generation in TEL	42
3.4	Synthesis & Discussion	46

Play is our brain’s favorite way of learning things.

— DIANE ACKERMAN

Nowadays, video games are increasingly present in our lives, as they entertain hundreds of millions of players around the world (Hendrikx *et al.* 2013). A video game can be defined as “an interactive application, entering into interaction with a player” (Djaouti *et al.* 2007). Designing video games is a complex task (Junior and Silva 2021). For educational games, this is especially true as it requires finding a balance between achieving player engagement and meeting learning outcomes (Hall *et al.* 2014). As a result, researchers have attempted to guide the design and analysis of games by developing dedicated methods (Junior and Silva 2021).

Often, the ability of video games to present engaging content is taken for granted. While manual content production is already expensive and unscalable, demand for new and tailored content continues to grow (Hendrikx *et al.* 2013). Therefore, any technology that could ease the burden of content creation and facilitate tailoring content to players would be warmly welcomed by game and serious game designers (Yannakakis and Togelius 2011). An existing solution, is the automatic content creation using algorithms. Several famous commercial games such as *The Binding of Issac* or *Hades* use that principle to automatically build different levels each time.

This chapter aims to position our work in terms of generation in relation to existing work. First, procedural content generation is introduced and defined in our context. Then, existing methods for game design are presented, since creating game content is an integral part of game design. Next, a state-of-the-art on existing work on generation of adapted content in games and in TEL is done. Finally, a discussion about the observations made from the literature is realised.

3.1 Definition of Procedural Content Generation

Born in the 80s, Procedural Content Generation (PCG) is a common method used in video game which “refers to the automatic creation of contents, performed using algorithms and/or heuristics” (Ripamonti *et al.* 2017). PCG can be exploited to produce different contents such as levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters (Shaker *et al.* 2016). More precisely, in games, content “refers to all aspects of a game that affect gameplay but are not nonplayer character behavior or the game engine itself” (Yannakakis and Togelius 2011).

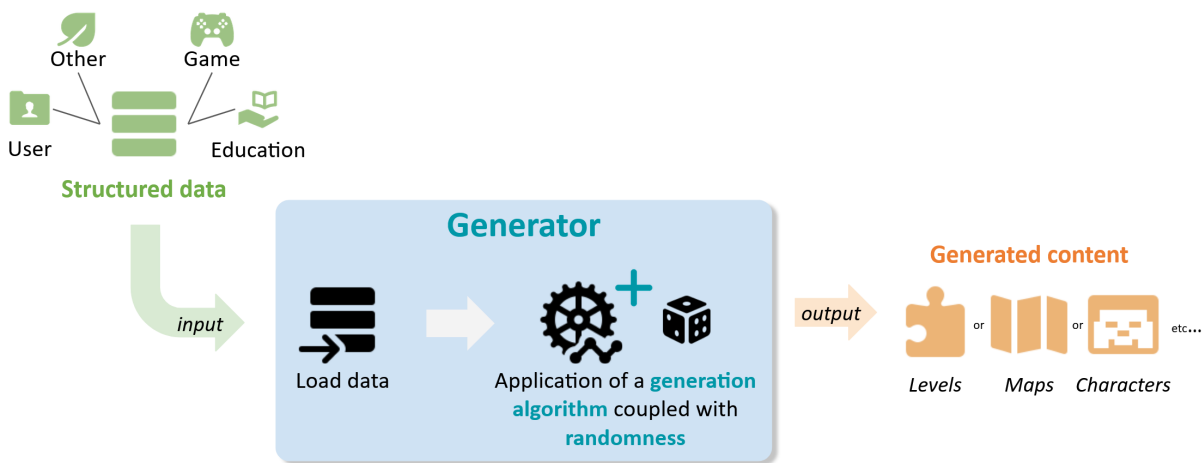


Figure 3.1 – General illustration of the content generation process

Traditional PCG methods¹ are computer procedures or algorithms often coupled with randomness² that uses structured data (i.e., machine-readable information such as game elements available) as input to create/produce content. In order to tailor the generated content to the players, data about them needs to be provided to the algorithm (Browne *et al.* 2014). Player data can be collected before the game (i.e., to generate new content for the next game), or during the game itself (i.e., to tailor content on the fly in the current game) (*ibid.*). The same principle applies for tailoring the generated content to any user: user data (e.g., personal information, knowledge, preferences, environment) must be collected before or during the activity. Figure 3.1 illustrates the generation process.

PCG has two main advantages: 1) it reduces the development time for new content, and 2) it increases randomness of content (Ripamonti *et al.* 2017), thereby increasing the variety of content provided to the users. However, designing a PCG algorithm is a complex task, especially since the generated content must be playable/achievable, e.g., completing a generated level, climbing a generated staircase, or winning a generated game should

1. PCG methods have been classified into three categories: traditional methods, search-based methods and machine learning methods. In this manuscript only the first one (i.e., traditional) is addressed.

2. In a stochastic approach, PCG uses randomness to vary the generated content compared to a deterministic approach where, for the same input data, the content created is identical (i.e., regenerated).

be possible (Shaker *et al.* 2016). In this manuscript, procedural content generation is abbreviated by the term *generation*.

3.2 Games & Serious Games Design

Although **designing generators of game content**, such as activities or levels, **does not require the actual design and development of games** (i.e., generators can be seen as independent software components, see Chapter 5), creating such content **does involve making upstream game design choices** (e.g., a game level cannot be generated without knowing what its composition should be: is it a racetrack? Is it a road with obstacles to avoid?). Therefore, it is necessary to pay attention to game design when it comes to generating game content.

Numerous models, methods, frameworks and approaches have been developed to guide the design and analysis of the possibilities and limitations offered by games and serious games (Carvalho *et al.* 2015; Junior and Silva 2021). As an example, the MDA framework, introduced by Hunicke *et al.* (2004), has been among the most influential and frequently used mainly for game analysis (Junior and Silva 2021). MDA proposes to divide games into three elements, see Figure 3.2, that influences each other: Mechanics, Dynamics, and Aesthetics. Mechanics describes the game components. Dynamics describes the game mechanics behaviour at runtime. Aesthetics describes the desired emotional reactions for players when they interact with the game. Furthermore, MDA has been redefined by Junior and Silva (*ibid.*) to make it more useful from a game designer perspective, since it is not in use in the game industry to help the game design work.

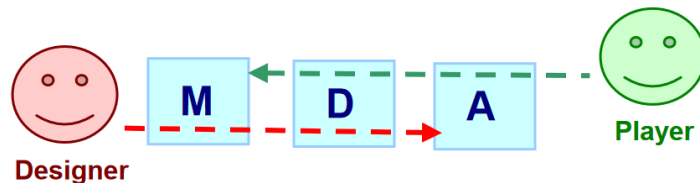


Figure 3.2 – MDA framework order of influence (Hunicke *et al.* 2004)

Another example is GOM (Game Object Model) version I and II proposed by Amory *et al.* (1999) and Amory (2007). GOM is an object-oriented model which considers that serious games are composed of objects described through concrete or abstract interfaces. Educational elements are linked to abstract interfaces, whereas games elements are linked to concrete interfaces. Several other work addresses educational games analysis and design. Kiili (2005) proposes a model based on experiential learning theory, flow theory (Nakamura and Csikszentmihalyi 2009) and game design that stresses the importance of immediate feedback, clear goals and challenges matching player’s skill levels. De Freitas and Jarvis (2006) propose a four-dimensional (i.e., context, representation, learner, pedagogy) framework. Yusoff *et al.* (2009) propose a conceptual framework to assist developers in ensuring the effectiveness of learning in a serious game. Barbosa *et al.* (2014) propose

a method for designing and developing serious games that facilitates the integration of educational content into games by using the concept of a learning mechanism that must be included in the game (i.e., in the storytelling or gameplay). Winn (2009) proposes a framework extending MDA to add the educational dimension to game design. Marne *et al.* (2012) present a conceptual framework based on six facets (i.e., pedagogical objectives, condition of use, decorum, problems and progression, interactions with the simulation, domain simulation) that aims at allowing everybody concerned of the game design process to speak the same language. Carvalho *et al.* (2015) propose a conceptual model in order to better understand the relations between game elements and educational goals of the game. An interesting aspect of research into educational game design and analysis concerns the alignment between educational and game content. Chapter 6 addresses this aspect in more detail.

However, most of these works are theoretical and therefore difficult to translate into the design process (Junior and Silva 2021), as they mainly focus on high-level aspects and requirements and do not provide an understanding of how these requirements can be satisfied in practice (Carvalho *et al.* 2015). In addition, when these studies are not aimed at high-level design, they are focused on game design analysis (Junior and Silva 2021). Moreover, despite adaptation being a major research interest and despite many video games being based on a generation principle, to the best of our knowledge, none of this work addresses the concepts of generation and adaptation.

3.3 Existing Work

Although content generation has mainly been addressed in the context of video games, some studies have focused on content generation in TEL. This section presents the different angles from which generation has been tackled in video games and in TEL.

3.3.1 Content Generation in Video Games

In video games, content generation has been tackled from two main angles: non-adapted generation (i.e., provide new algorithms for generating game content) and player adapted generation (i.e., provide new methods for generating game content tailored to players' model/characteristics/preferences/etc. to improve their experience).

Non-adapted generation. Several studies addressed game content generation through the use of genetic algorithms. Khalifa *et al.* (2016) propose General Video Game Level Generation (GVG-LG) a framework to help level generation and provide a benchmark for level generation. The framework is developed in Java and supports a Java Interface that allows different users to create their own level generators. Games are described through objects called *GameDescription* using the Video Game Description Language (VGDL). Three level generators have been implemented: Random (i.e., the algorithm places defined sprites at random empty positions, then surrounds the borders with solid tiles),

Constructive (i.e., the procedure uses the *GameDescription* object to design better levels), and Search-Based (i.e., the algorithm is based on the Feasible Infeasible 2 Population Genetic Algorithm). Soares De Lima *et al.* (2019) propose a procedural generation architecture based on the use of genetic algorithms to create quests (i.e., linear sequences of events or tasks that the player must complete, e.g., `attack(zombie, anne, johnhome)`, `go(john, johnhome, village)`). The architecture is composed of a *Domain database* which contains information about all the elements available in the game world (e.g., *john* is a character), all the properties and relationships that exist in the game world (e.g., *john* is alive, *antidote1* is an *antidote*), the set of semantic integrity constraints (e.g., *is infected*, *has*), the set of possible events (i.e., to maintain consistency with previous quests) and a parameter for managing difficulty. Pereira *et al.* (2021) propose a PCG method based on the use of constrained evolutionary algorithms that evolve a population of randomly generated dungeon maps filled with locked-door missions. Dungeons and missions are modelled in the form of trees (i.e., rooms = nodes) comprising 3 types of room: normal rooms (i.e., nothing special inside), key rooms (i.e., contain a key) and locked rooms (i.e., the access path to their parent is blocked). The method aims at matching the level and mission characteristics (i.e., number of rooms, keys, locked doors, and level linearity) expected by the game designer.

Player adapted generation. Several works propose methods for content generation adapted to players in order to improve their game experience. Lopes and Bidarra (2011b) propose a semantics-based procedural game world generation framework that adapts gameplay according to experience, player and content utility models during the game (e.g., between each room, at each street). Gameplays are modelled using semantics, i.e., “declarative modeling approach that embeds the world and its objects with all information beyond their geometry” (*ibid.*). Experience and player models capture skills (e.g., shooting proficiency), preferences (e.g., items used, actions taken) and styles (e.g., explore, achieve) while the content utility model captures the association of player/experience features with relevant content. The framework uses case-based reasoning to encode the valid combinations between the content and the gameplay experiences. Oliveira and Magalhaes (2017) propose a PCG method that can be used by game designers to steer the generation process in order to automatically propose a more personalized and context-aware experience to each player. The method has been applied to generate tailored items, using a player model (i.e., including characteristics/personality), a context model (e.g., location, device, current season) and a game model (i.e., preferences and actions of a specific player), each time the player defeats a wave of monsters.

Some studies have proposed methods or tools to guide the high-level design of content generators which take player adaptation into account. Yannakakis and Togelius (2011) propose a generic framework to link player experience with procedural content generation called Experience-Driven Procedural Content Generation (EDPCG). The framework has been defined based on existing research in the literature, and defines player experience as the “synthesis of affective patterns elicited and cognitive processes generated during

gameplay” (Yannakakis and Togelius 2011). The proposed work is composed of four main components: Content generator (i.e., “The generator searches through content space for content that optimizes the experience for the player according to the acquired model”), Content Representation (i.e., “Content is represented accordingly to maximize efficacy, performance, and robustness of the generator”), Content quality (i.e., “The quality of the generated content is assessed and linked to the modeled experience of the player”), and Player Experience Model (i.e., “Player experience is modeled as a function of game content and player (the player is characterized by her playing style, and her cognitive and affective responses to gameplay stimuli”). This framework guides the high-level design of generators (e.g., presents three types of player experience model: subjective, objective, or gameplay-based, with their advantage/disadvantages to choose from).

Other works addressed the generation of content tailored to types of players. Dormans and Bakkes (2011) proposes a framework for the design of action-adventure game levels based on the use of generative grammars, i.e., “a generative grammar typically consists of an alphabet and a set of rules”. Therefore, the alphabet consists of every element available to build game spaces and missions, whereas rules describes how to compose them. The game space and missions generated are adapted upstream according to player types (e.g., the dungeon structure is altered to add a portion specific to a player type) to personalise the game experience.

Some interesting studies investigate human behaviour to categorise content into categories³. Fujihira *et al.* (2022) investigated player’s behavioural tendencies when playing mazes, i.e., directions chosen by the players at branch points in the maze with the aim to extract the probability of choosing a path. Then, they used these results to build a prediction model simulating human players in order to defined mazes difficulty based on human perspective. Finally, they automatically generated mazes using a digging method and classified them into difficulty levels using the number of steps (i.e., predicted number of steps minus the shortest number of steps).

3.3.2 Content Generation in TEL

Despite being little discussed in TEL (Bezza *et al.* 2013), generation has been approached from three perspectives: non-adapted (i.e., provide new algorithms for generating learning or learning game content), learner adapted (i.e., provide new methods for generating game content tailored to players’ model/skills/knowledge/etc. to improve their learning), and learner-player adapted (i.e., provide new methods for generating game content tailored to learner-players’ model/characteristics/skills/knowledge/preferences/etc. to improve their motivation and learning).

Non-adapted generation. Some researches have addressed the generation of exercises (also called exercisers). Holohan *et al.* (2006) propose *OntAWARE* a system that leverages

3. Such approaches can be seen as a form of reverse engineering, wherein human behaviour is analysed and reproduced to classify the generated content. Once classified, players can benefit from tailored content.

ontologies for the generation of e-learning exercise problems on the subject domain of relational databases (i.e., the exercises consist of writing SQL queries). The user must import the relation schema ontology and populate the ontology with particular schema instance information (e.g., table and column names) then, on demand, the generator randomly creates queries that are converted into an English form (i.e., problems presented to learners).

Other studies have dealt with the generation of narrative learning scenarios. [Sina et al. \(2014\)](#) propose **ScenarioGen**, a general “fill and adjust” semi-automatic method for generating textual content about everyday activities through combining computer science techniques with the crowd. **ScenarioGen** generates coherent new scenarios by replacing content details within an original scenario. It is composed of three main components: MaxSat a maximal satisfiability solver which identifies the places where modifications are required, KAR a K-nearest neighbour Activity Replacement which selects the most appropriate activity for a new scenario, and SNACS Social Narrative Adaptation using Crowdsourcing which adjusts the activity by adding rich detail. They apply the method in a serious game called **VirtualSuspect** which allows repeated practice sessions using different types of investigation techniques for different cases of property felonies for law enforcement training.

Some studies have focused on the generation of learning paths. [Diwan et al. \(2019\)](#) propose a generic model for generating learning pathway based on open-sources resources. More precisely, they address the lack of standardisation and metadata in open learning resources (i.e., the search for genericness) by proposing various methods for calculating learning pathways (i.e., comparing learning resources and integrating resources of a given subject into a logical learning space). The model has two main components: a Greedy Generator (i.e., generates a learning path using a learning goal and a resource as a starting point) and a Validator (i.e., provides feedback to the generator and improves the pathway it produces).

Learner adapted generation. Numerous studies have tackled the generation of learning paths adapted to the learner from existing resources⁴. [Vassileva \(1995\)](#) proposes an approach for dynamic courseware generation which generates a course plan with a given objective. This work uses a set of task hierarchies, teaching methods, and a learner model to dynamically decide how to best execute the plan for the learner according to a set of teaching rules (i.e., adapted in real time based on learners’ interactions). [Melis et al. \(2001\)](#) present **ActiveMath** a web-based ITS for mathematics that dynamically generates, when explicitly asked, interactive courses adapted to a learner model. Generation is realised in three incremental steps: retrieval of the concepts and learning resources required to achieve the learning goals from a knowledge base, arrangement of the concepts and learning resources retrieved based on learner’s prior knowledge, and linearisation of the concepts and learning resources (i.e., represented as a graph). [Sehaba and Hussaan \(2013\)](#)

4. Several studies exist in Courseware generators (i.e., a course is generated in a single go before being presented to the learner) and sequencers (i.e., the best activity/resource is selected at any moment).

propose **GOALS** a generic (i.e., domain independent) approach, see Figure 3.3, for the generation of scenarios (i.e., “series of activities in the form of educational games helping the learner to achieve one or more learning objectives”) in educational games. Several models define this approach: the domain model, which models domain concepts and their relationships; the learner model, which models learners’ personal information, motivation, skills and interactions; the presentation model, which describes the structure of the scenarios; and the serious game model, which combines game resources with pedagogical resources. Adaptation knowledge is modelled using a rule-based system. The scenario generation process is broken down into several incremental steps: 1) selection of domain concepts and creation of the conceptual scenario, 2) selection of learner-adapted pedagogical resources based on the conceptual scenario, followed by creation of the pedagogical scenario according to the presentation model, 3) selection of game resources based on the needs of the pedagogical scenario for the creation of the game scenario.

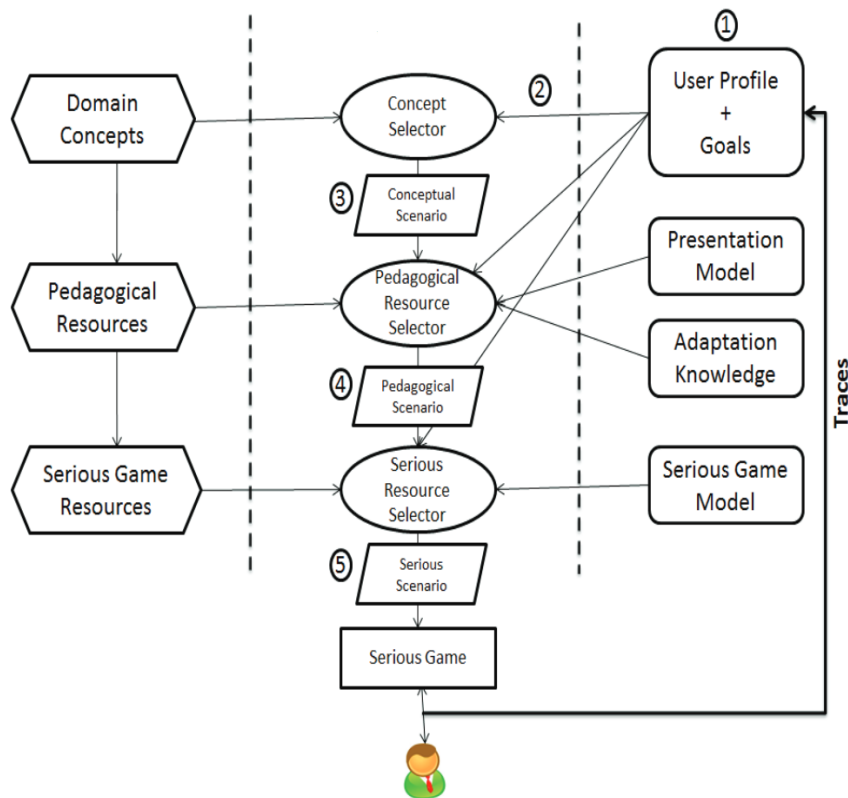


Figure 3.3 – Scenario generator architecture (GOALS) of [Sehaba and Hussaan \(2013\)](#)

[Carpentier and Lourdeaux \(2014\)](#) propose an approach based on the Zone of Proximal Development (ZPD) to dynamically generate learning situations adapted to learners’ profiles (i.e., their abilities and pedagogical needs) in virtual environments. The approach is part of a framework based on three models: the domain model (i.e., the static description of the world, its elements and their relationships), the activity model (i.e., the hierarchical

structure of the observed activity), the causality model (i.e., the expression of the relevant causal chains occurring in the environment).

Some work address the generation of learning scenarios and activities. Laforcade and Laghouaouta (2018) propose 3x3, see Figure 3.4, a generic MDE-based architecture for game scenario (i.e., “ordered sequence of scenes including precise descriptions of each scene components and locations”)⁵ generation adapted to learners’ individual needs based on the general architecture of Sehaba and Hussaan (2013). Like Sehaba and Hussaan (*ibid.*), the generation process is decomposed into three-incremental steps: objective scenario generation, structural scenario generation, and feature scenario generation. The proposed architecture, see Figure 3.4, captures the domain elements required for generation in three inter-related parts of a metamodel that must be specified in models: the profile model (input) describing the learner profile, the game description model (input) describing each element of the game, the scenario model (output) encompassing the three generated scenarios (objective, structural, feature). The proposed work is a very general specification method whose metamodels and models are produced during its application. The method has been implemented within the serious educational game *Escape It!*, which aims at developing visual skills for autistic children.

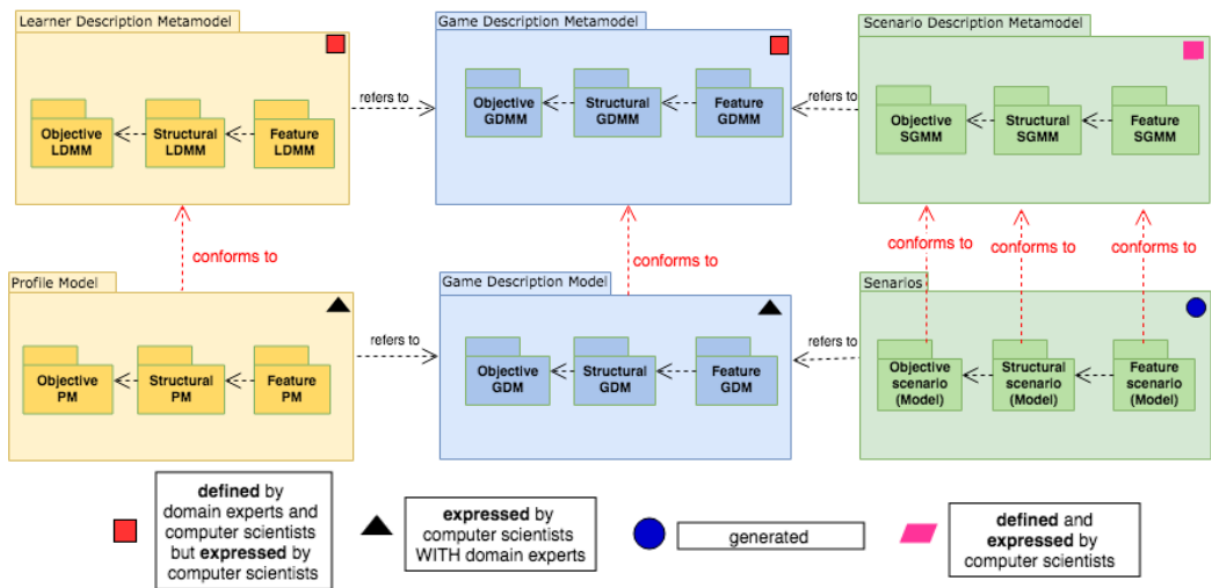


Figure 3.4 – The 3x3 metamodel-based architecture of Laforcade and Laghouaouta (2018)

Learner-Player adapted generation. One study proposed to generate learning scenarios, based on existing resources, adapted to the learner and the player. Callies (2016) proposes an architecture for real-time generation of learning scenarios (i.e., sequences of

5. Compared to Sehaba and Hussaan (2013), the scenarios in this work are composed of scenes that are built (i.e., components are chosen and positioned) by the generator instead of using predefined activities.

predefined activities) adapted to learner-players (i.e., adapted feedbacks, non-player character behaviour, and learning context) in serious simulation games which was implemented in a called **Game of Homes**. The architecture is composed of three main components: a learner-player model that allows the estimation of knowledge and skills relative to learning and skills relative to gameplay; an adaptation module that selects the sequences of activities, to achieve a goal, for a learner-player; and a control module that executes the scenario generated by the adaptation module (i.e., it modifies in real time, non player character behaviour, change the simulation parameters, etc.) and requires new scenarios when the scenario is no more adapted to the player’s current knowledge.

3.4 Synthesis & Discussion

This chapter presented procedural content generation as well as its application in video games and in TEL. Although the design of procedural generation methods is complex, it enables the automatic production of varied and adapted content (i.e., simplifying work for designers).

Accordingly, several research contributions focus on the generation of adapted or non-adapted content. Among these works, some have proposed so-called **generic approaches** which can be defined as: the ability of the generator “to be independent of a particular application domain, and therefore able to be used in several domains and several serious games” (Sehaba and Hussaan 2013). In this definition, genericness encompasses the educational dimension (i.e., independence from the didactic domain) and the game dimension (i.e., independence from a specific game genre). The existing generic works propose general design approaches for generators. However, as some of the approaches presented are outside the context of games, the genericness characteristic only refers to their independence from the didactic domain. Having such high-level approaches is beneficial for guiding the general design of generators for any didactic domain and any game genre. However, the main disadvantage of being generic on both the game dimension and the educational dimension is that it does not deliver a software infrastructure for generator design and therefore **prevents any reuse of structured input data** (i.e., generator sources) from one didactic domain to another.

Even though adapting the generated content to the player is a common practice in the context of video games, the majority of work in TEL deals with the generation of content adapted to the learner. To the best of our knowledge, **only one work deals with adapting the content generated to learner-players** (Callies 2016). Additionally, although generation of game space and missions (i.e., level with activities/tasks) is common in video games, **in TEL, generation mainly concerns learning or learning game scenarios** (i.e., sequences of pre-existing activities) and **not the design of activities** that learners or learner-players will have to carry out. As far as we know, only one work addresses the generation of learning games activities in TEL (Laforcade and Laghouaouta 2018). Table 3.1 summarises and compares the generation approaches in TEL.

	GENERIC APPROACH	LEARNER ADAPTED	PLAYER ADAPTED	GENERATED CONTENT
Holohan et al. (2006)	✗	✗	✗	e-learning exercises on relational databases
Sina et al. (2014)	🎓	✗	✗	textual game scenarios
Diwan et al. (2019)	🎓	✗	✗	learning paths based on existing open-sources resources
Vassileva (1995)	🎓	✓	✗	learning paths based on existing resources
Melis et al. (2001)	✗	✓	✗	learning paths based on existing maths resources
Sehaba and Hussaan (2013)	🎓 🎮	✓	✗	learning game scenarios based on existing resources
Carpentier and Lourdeaux (2014)	🎓	✓	✗	virtual learning situations
Laforcade and Laghouaouta (2018)	🎓 🎮	✓	✗	learning game scenarios + tasks to be carried out by users
Callies (2016)	🎓 🎮	✓	✓	learning game scenarios based on existing resources

Table 3.1 – Comparative table of different generation works in TEL (🎓 = educational dimension, 🎮 = game dimension)

Additionally, an important observation is that in a model-based approach, the structure of the content to be produced is modelled such as the presentation model for [Sehaba and Hussaan \(2013\)](#), the scenario model for [Laforcade and Laghouaouta \(2018\)](#), or the activity model for [Carpentier and Lourdeaux \(2014\)](#). Specifically, it can be observed that to generate adapted content for educational games, it is necessary to provide various sources:

- information about the game, i.e., description of the elements available and the progression of the game (e.g., [Callies \(2016\)](#), [Laforcade and Laghouaouta \(2018\)](#), and [Sehaba and Hussaan \(2013\)](#));
- information about the structure of the content to be produced, i.e., elements that

- constitute the content (e.g., [Sehaba and Hussaan \(2013\)](#) [Carpentier and Lourdeaux \(2014\)](#) and [Laforcade and Laghouaouta \(2018\)](#));
- information about the didactic domain, i.e., knowledge, how to work on it (e.g., [Carpentier and Lourdeaux \(2014\)](#), [Laforcade and Laghouaouta \(2018\)](#), [Melis *et al.* \(2001\)](#), and [Sehaba and Hussaan \(2013\)](#));
 - information about the learner, i.e., knowledge, skills, progress/results (e.g., [Callies \(2016\)](#), [Laforcade and Laghouaouta \(2018\)](#), [Melis *et al.* \(2001\)](#), and [Sehaba and Hussaan \(2013\)](#));
 - information about the player such as preferences or profile, progress (e.g., [Callies \(2016\)](#)).

Our interest lies in proposing a **generic approach**, on the **educational dimension**, for the **generation of educational game activities adapted to learner-players** (i.e., game level and task to be carried out by learner-players). The main idea is to be domain-independent and to be able to reuse data from one domain to another (i.e., propose a generic generation algorithm). Next chapter details our research issue and delimits our research perimeter.

RESEARCH ISSUE

Contents

4.1	Research Questions	50
4.2	Positioning	52
4.2.1	Roguelite Game Genre	52
4.2.2	Adaptations & Variety	54
4.2.3	Model-Driven Engineering	55
4.3	Research Method & Evaluation	56

Research is what I'm doing when I don't know what I am doing.

— WERNHER VON BRAUN

Previous chapters have highlighted several observations. First, although generating video game content often targets missions to be carried out by players, in TEL, generation is more often oriented towards organising the activities to be carried out rather than designing them.

Second, adaptation is rarely tackled simultaneously from an educational and a gaming perspective. In the context of generation, this observation is even more significant, as only one work proposes to adapt to the player and the learner.

Third, an essential observation relates to the use of structured data sources, such as models or ontologies, in research dealing with generation and adaptation. Specifically, to generate adapted content for educational games, it is necessary to provide various sources presenting information about the game, the didactic domain, the structure of the content, and the learner-player.

Lastly, though most TEL generation works propose generic approaches that can be applied to different didactic domains or video game genres, these works do not allow the direct reuse of some of the elements designed for one generator to another.

Building on these observations, our interest lies in the generation of activities, namely activities to be undertaken by learners and not a sequencing of pre-existing activities, in the context of video games for declarative knowledge training. Figure 4.1 positions our work in relation to adaptation, generation and games. More precisely, our research problem is the following:

Research problem

How to guide the design and implementation of **generators** of **learner-player adapted** and **varied game activities** for declarative knowledge training?

This is a TEL systems **engineering research** problem (Tchounikine *et al.* 2009) seeking the exploration of solutions for the generation of adapted and varied activities. This chapter aims to define our research questions, in relation to our research problem, based on the previously made observations, as well as define the scope and research method of this PhD thesis. First, this chapter presents the research questions arising from our research problem. Next, it outlines the scope of our work by presenting our proposal for tackling the research questions, and describing its perimeter (i.e., Roguelite game genre, targeted adaptations, Model-Driven Engineering). Finally, it describes the research and evaluation methods.

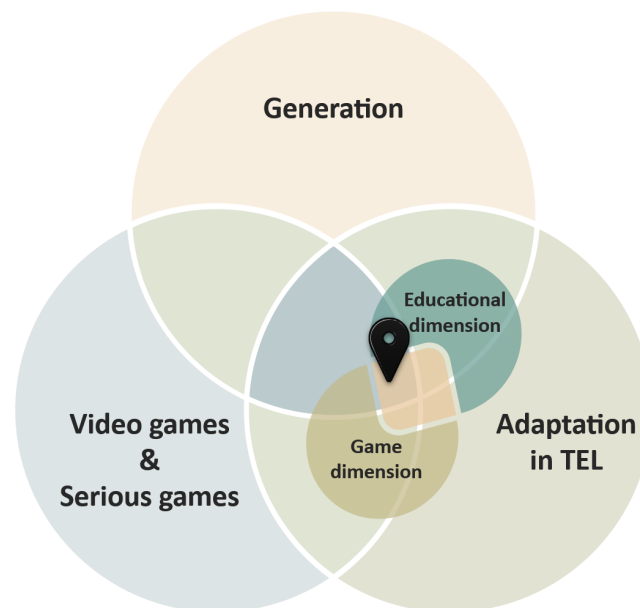


Figure 4.1 – General position of our research problem

4.1 Research Questions

A key advantage of focusing on declarative knowledge in general is the possibility of proposing models, algorithms and tools that can be reused beyond a specific didactic domain. Consequently, our first research question is as follows:

Research question 1 (RQ1)

How to propose **an approach** sufficiently **generic** to **consider declarative knowledge independently** of a specific **didactic domain**?

Furthermore, generation requires modelling the structure of the content to be produced. Therefore, it is necessary to define what is a game activity for declarative knowledge training that is varied and adapted to the learner-player. Consequently, our second question is as follows:

Research question 2 (RQ2)

What is a learner-player adapted and varied game training activity?

From this question, the following sub-questions are drawn:

Which educational and game elements constitute these activities?

How to combine the game and educational elements coherently? ^a

a. Relates to the problem of *educational and game dimensions alignment*.

As the generation requires several sources to produce the targeted activities, it is necessary to structure the sources (i.e., information about the game, the didactic domain, the learner-player) and their relationships in order to pilot the generation process. That leads us to our third question:

Research question 3 (RQ3)

How to structure the required data and their relations in order to drive the generation of coherent activities?

Finally, these structured data have then to be computerised to enable their use in the generation process. To this extent, the data required, their relationships and the structure of the activities to be produced must be specified at a granularity level understandable by computers. Accordingly, our last question is as follows:

Research question 4 (RQ4)

How to specify every information required for generation to enable computer interpretation for the development of activity generators?

4.2 Positioning

In order to address our problem and research questions, our proposal is **an extensible framework for the design and implementation of generators of adapted and varied activities for declarative knowledge training**. This framework is broken down into two parts: a conceptual framework (i.e., theoretical part describing all the generation features, e.g., conceptual models and algorithms) (Lemoine and Laforcade 2023a) and a software infrastructure (Lemoine *et al.* 2023c) that can be extended to different didactic domains (i.e., construction tool guiding the design of generators). This software infrastructure captures every element that is common to different didactic domains. Furthermore, the extension mechanism guides the insertion of specific elements for a given didactic domain. This infrastructure is addressed through a Model-Driven Engineering approach (Brambilla *et al.* 2012).

Game activities have structures that are entirely dependent on the game genre targeted. As examples, let's consider two activities, one involving solving a puzzle and another involving exploring a world. Both activities are designed differently as a puzzle is built around a problem to be solved (e.g., *Professor Layton*), whereas an exploration activity requires the design of a world (i.e., a virtual exploration zone) and an avatar (i.e., a character) who will roam and perform actions in the world (e.g., *Minecraft*).

Accordingly, a game genre must be chosen to enable the generation of activities. Moreover, to propose a software architecture extensible to different didactic domains, the chosen game genre must have different characteristics that enable them to satisfy the required conditions for declarative knowledge training, namely they must:

- encourage repetition (i.e., DK training requires repeated sessions);
- encourage variety (i.e., DK training requires several diverse and adapted activities);
- provide a sense of progression (i.e., training aims at knowledge acquisition).

Note that in order to provide a software infrastructure (i.e., low-level design) allowing the reuse of data/algorithms between different didactic domains, the genericness of the game dimension had to be discarded. Selecting different genres for different domains would require a similar approach, where elements of the game dimension are also abstracted and extended. Additionally, having both dimensions extensible would probably exponentially increase the complexity of extending the work. Therefore, our choice has been to disregard the genericness of the game dimension.

4.2.1 Roguelite Game Genre

Originating from the groundbreaking game *Rogue* (Toy *et al.* 1980), *Rogue like* and *Rogue lite* video game genres have experienced a significant surge in popularity over the last decade. *Rogue* is a turn-based dungeon crawler that pioneered procedural generation by automatically generating levels of dungeons for players to explore, fight enemies, collect items, and progress. *Rogue*'s infinite replayability has been a huge draw and has been emulated many times over, a recent example is the commercial game *Diablo (Blizzard)* (Yannakakis and Togelius 2011). In both *Rogue like* and *Rogue lite* genres, a dungeon often

represent “a set of interconnected rooms containing different challenges” (Pereira *et al.* 2021). Figure 4.2 displays examples of dungeons maps and rooms from commercial games.

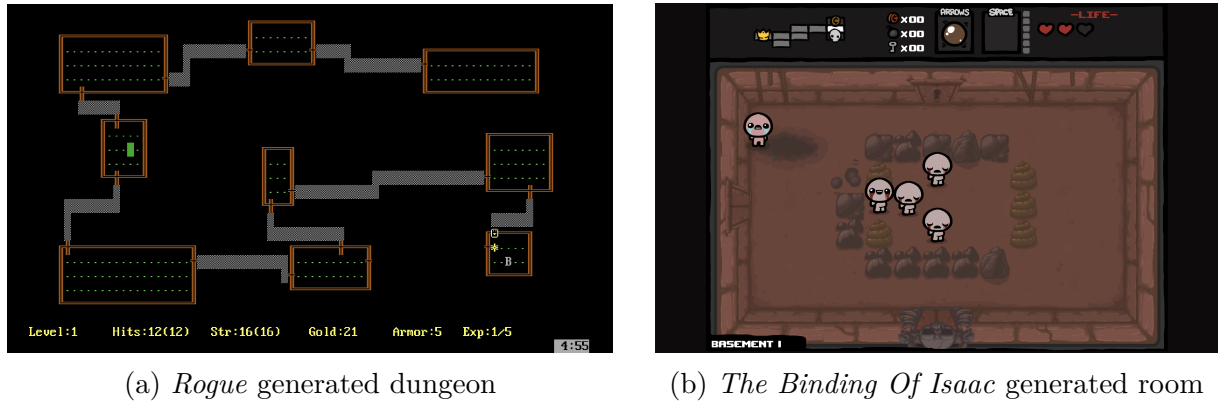


Figure 4.2 – Examples of dungeons maps and rooms from existing commercial *Roguelites*

According to the Berlin Interpretation (Harris 2020)¹, there are eight key factors that characterise *Roguelikes*, including:

- *randomised generation*: levels and their elements (e.g., foes/objects locations, environmental conditions) are usually built using procedural generation with semi-randomness to avoid unwinnable situations. Randomness offers an element of surprise and unpredictability, requiring players to adapt their strategies on the fly.
- *permanent death*: each time the avatar dies, any progress achieved is lost and players have to start all over again (i.e., no progress is carried over between runs).

Although many *Roguelike* games respect these eight key aspects, some games deviate from certain elements. Hence, *Roguelites* have emerged as a means to differentiate such games from traditional *Roguelikes*.

Roguelites introduce macro-level objectives by allowing players to keep some items or upgrades between attempts. This system of persistent progression allows players to progressively become stronger over time, increasing their chances of success in subsequent runs. Well-known commercial *Roguelites* (e.g., *Hades*, *Enter the Gungeon*, *The Binding of Isaac*, *Rogue Legacy*, *Children of Morta*, *Dead Cells*) provide diverse, gameplays, lores, features, and permanent elements (e.g., weapons, currencies, upgrades, characters) that contribute to achieve cross-run objectives. For some games, collectible resources can persist in-between deaths and be used to unlock permanent upgrades.

Failure is an integral part of *Roguelites*. Players are often confronted with new mechanics, traps, enemies and bosses that require learning skills. As a result, players often fail and die several times before completing a play through. Despite repeated failures, *Roguelites* generally offer quick restarts, allowing players to rapidly re-play. Each new game helps players understand the underlying game mechanics better, enabling them

1. It is a popular interpretation of “What a Roguelike is” that has been created at the International Roguelike Development Conference of 2008.

to progress further. Additionally, replayability is another integral aspect of *Roguelites* as each run offers a distinct experience. Due to the changing nature of the levels and encounters, *Roguelite* games have a high replay value, as no two runs are identical. Moreover, many *Roguelites* have a replay value that goes beyond just completing the game, e.g., a *new game+* mode like in *Rogue Legacy* or a scenario requiring repeated defeats of the final boss like in *Hades* or *Dead Cells*.

As *Roguelites* are mainly characterised by procedural dungeon generation with pseudo-randomness providing a variety of content, permanent death providing repetitive game sessions, and limited detention of unlockable items (e.g., characters, items, powerups) providing persistent progression, they meet the requirements of declarative knowledge training. Accordingly, our **framework aims to guide the design and development of generators of dungeon² levels for Roguelite-oriented games** for declarative knowledge training.

4.2.2 Adaptations & Variety

As aforementioned, adaptation and variety of activities are required in order to reduce the feeling of boredom caused by repetitive training. Since adaptation is rarely addressed from an educational and gaming perspective conjointly, the adaptation of levels generated aims to consider information about learners, their training, and players. More precisely, our aim is to adapt according to three viewpoints:

- the **learner**, by taking into account their level of knowledge, results, and progress;
- the **teacher**, by considering their training strategy for learners and their pedagogical/didactic choices;
- the **player**, by considering their game preferences.

According to [Plass and Pawar \(2020\)](#), games can be adapted to different types of variables. In our context, the adaptation targeted focuses on two cognitive variables (i.e., the learner’s knowledge level/progression and the teacher’s strategy/choices) as well as a motivational variable (i.e., the player’s personal interests given as game preferences). Therefore, adapting to the educational dimension involves considering various aspects such as the types of facts encountered, how they are questioned, their number, their order according to the learner’s level, their previous results and the training established by the teacher. *Roguelites* often include a purchase/activation mechanism enabling players to make choices that may or may not influence the generation of game levels. Game preferences are then expressed in the form of purchasable and activatable items, which affect the game situations presented. These preferences allow players to bypass game situations³ they dislike by deactivating items, or to simplify the game by deactivating or not purchasing them. Table 4.1 summarises the characterisation of our adaptation.

2. Note that considering *Roguelites* through dungeons (i.e., set of interconnected rooms) is a design choice, another form such as caves could have been made.

3. Importantly, some game situations must be independent of the purchasable/activatable items (i.e., default game situations) to ensure that game situations are available in all circumstances.

In order to generate varied activities, our approach consists of modelling educational and game elements, provided as input to the generator, with a certain degree of variability. The general idea is to allow the generation algorithm to pseudo-randomly select, to preserve the coherence of the activities, several training and game elements to increase variety. For example, questions on facts are modelled to ensure that bad proposals are selected at each generation. Our approach to modelling information is based on model-driven engineering.

SOURCES	Learner’s knowledge/results/progress Player’s preferences Teacher’s training strategy for learners
TARGETS	Training game activities for Roguelite-oriented games
PATHWAYS	Procedural generation
AUTOMATION LEVEL	Generator automatically provides adapted content
MOMENT	Each time the generator is called

Table 4.1 – Final characterisation of the adaptation of our system

4.2.3 Model-Driven Engineering

Model-Driven Engineering is a form of generative engineering based on the use of models to design all or part of a computer application in order to facilitate its development. Models are abstractions of objects linked to a specific business domain that are sufficiently exhaustive and explicit for understanding the domain being modelled. MDE is structured around four principles:

- *capitalisation*, i.e., models must be reusable;
- *abstraction*, i.e., models must capture the complexity of a system and exclude unnecessary details;
- *modelling*, i.e., models must be productive, in other words they must enable at least some of the final software code to be generated;
- *separation of concerns*, i.e., models must capture different concerns, aspects, viewpoints.

Model transformation is a central operation in MDE, consisting of the “production of a set of target models from a set of source models, according to a transformation definition” (Sottet *et al.* 2007). A procedural content generation process based on MDE can thus be considered as a set of model transformations taking models as input and producing models as output, until executable artefacts are obtained. An entity called a metamodel is used to enable these models to be designed and manipulated in a computerised context. A metamodel is an abstraction of the modelling language (i.e., syntax, grammar and semantics) of models, i.e., each model is considered as “conforming to” the metamodel. Additionally, to describe the structure of metamodels, the concept of meta-metamodel

exists. Meta-metamodels are recursive, meaning that they are conforming to themselves (i.e., they are self-describing). Figure 4.3 presents MDE levels of abstractions from the real-world to meta-metamodels.

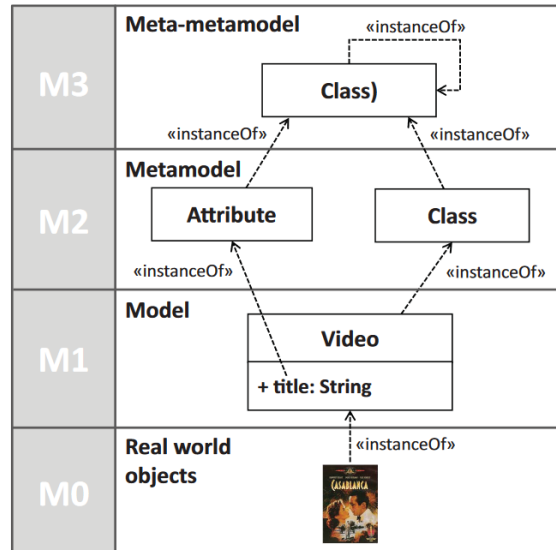


Figure 4.3 – MDE levels of abstraction (models, metamodels, meta-metamodels) (Brambilla *et al.* 2012)

Hence, MDE is a vast research field offering both a theoretical foundation (e.g., models/metamodels/languages, model transformation/composition/verification) to support specification needs and techniques and tools (e.g., ecosystems exploiting formalisms such as Ecore or Epsilon) to support implementation needs. However, meta-modelling is a subjective (i.e., there are several ways of specifying/modelling information) and complex activity that requires specific skills for creating and managing models as well as checking their accuracy in relation to the reality they intended to represent.

4.3 Research Method & Evaluation

Our proposal has been developed as part of an exploratory study centred on an iterative design from an initial case study, i.e., multiplication tables. This exploratory research involved a user group and the participation of domain and game experts. Therefore, this is an inductive method from which results obtained, in relation to the **AdapTABLES** project, are generalised and re-evaluated in the context of multiplication tables, but also in two other domains. Since the context of multiplication tables does not involve questioning declarative knowledge on pictures (e.g., legend on a map), another domain has then been used as an initial domain to consider these needs: history-geography facts that are required for the *Diplôme National du Brevet des Collèges* (DNB) a French exam taken in 9th grade.

Figure 4.4 illustrates our research method. First, information has been gathered from experts in order to identify training and game elements as well as adaptation needs for training (i.e., adaptation to be considered from the teacher’s perspective). Second, this information has been abstracted and generalised in order to be specified through models/metamodels and to define the generation algorithm (i.e., formalisation of the framework). Next, each component specified has been implemented, along with directives for extending the framework (= software infrastructure). Finally, our proposal has been applied and evaluated through the creation of several extensions: extension to the initial didactic domains and to another domain (i.e., judo facts). Furthermore, software verification (i.e., testing and model validation) as well as experimentation with an engineer have been conducted. The purpose of this experimentation was to have an independent (i.e., outside our research) engineer create an extension of the framework to another domain (i.e., solar system facts). Our method is not linear, and the process going from data collection to formalisation to development has been subject to many back and forth.

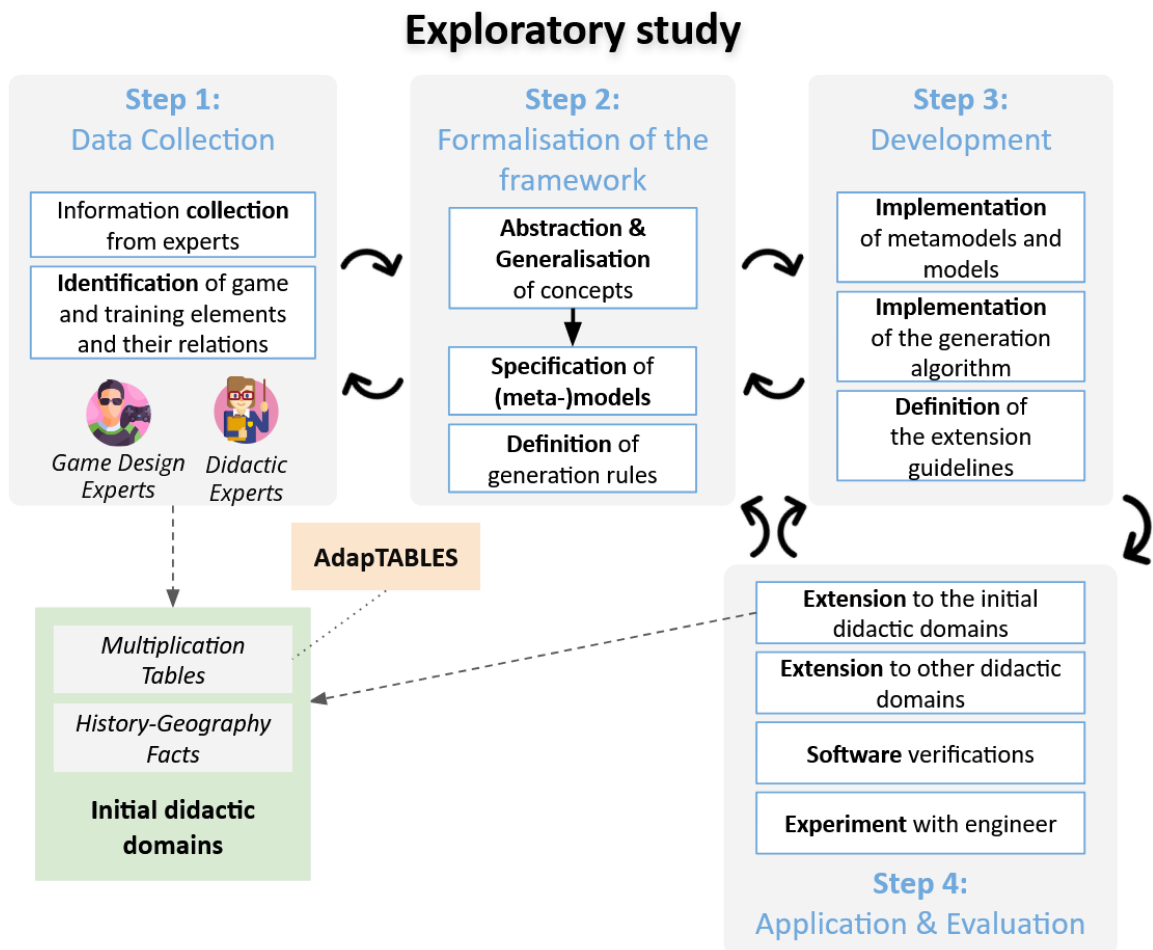


Figure 4.4 – Our research method

Now that our work has been positioned, the next part will present our contribution and the different sub-contributions developed to build the framework.

PART II

Design and Implementation Framework of Activity Generators

Oriented Roguelite for Declarative Knowledge Training

DESIGN FRAMEWORK OF ACTIVITY GENERATORS

Contents

5.1	General Overview	61
5.2	Definition: Game Activity for DK Training	63
5.3	Declarative Knowledge Training Elements	65
5.3.1	Training Path	66
5.3.2	Training Task Types	67
5.3.3	Training Tasks Parameters	67
5.4	Roguelite-oriented Game Elements	68
5.4.1	Analysis method for Roguelite Design	69
5.4.2	Design Choices for Activity Generation	70
5.4.3	Gameplay Categories	72
5.5	Synthesis	74

In order to address our research issue, our proposal is to provide a framework (i.e., conceptual and software infrastructure) to guide the design and implementation of generators of `Roguelite` oriented game activities for declarative knowledge training. To address our first research question (i.e., “How to propose an approach sufficiently generic to consider declarative knowledge independently of a specific didactic domain?”), our aim is to make the framework extensible to different didactic domains (i.e., reuse of code/models). To design such a framework, the first step consists of answering our second research question by defining what is a game activity for declarative knowledge training and what are its elements.

This chapter aims at presenting an overview of the framework, defining game activities for declarative knowledge training, and specifying elements composing them. First, we provide an overview of the proposed framework. Next, we define activities and present their components. Finally, we introduce the problem of aligning the game and training elements.

5.1 General Overview

The proposed framework is a conceptual ([Lemoine and Laforcade 2023a](#)) and software infrastructure ([Lemoine *et al.* 2023c](#)) encompassing models and tools to formalise and

guide the implementation¹ of varied and adapted game activity generators. The produced generators can be considered as components of a declarative knowledge training game of the *Roguelite* genre. Such generators create a new training activity on each request, i.e., a detailed text description of a dungeon for a given learner-player. Then, these descriptions have to be interpreted by a *game engine* to provide playable game levels to learner-players.

From an MDE standpoint, the generation algorithm can be perceived as performing a model transformation. Therefore, the produced generators require a model as input conforming to each input metamodel of the framework (i.e., concrete data, e.g., knowledge to work on, game elements available, results/progress of the learner-player) and produce a model as output also conforming to a metamodel. These generators, see Figure 5.1, are composed of the source code corresponding to the metamodels of the framework, enabling them to interpret and use the data contained in the models.

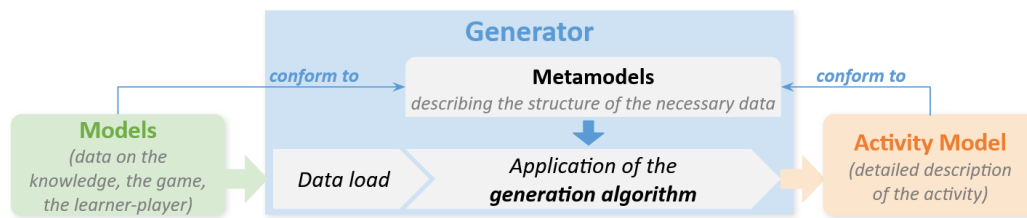


Figure 5.1 – Coarse-grained components of a training game activity generator

The conceptual approach of the framework enables to structure all necessary information for the generation through conceptual models (see Chapter 8) whereas the software infrastructure, implemented using an MDE approach (see Chapter 9), provides a tool to guide generators design through the reuse of a maximum of elements (i.e., models/metamodels and source code). Therefore, the software infrastructure is composed of generic components that must be extended to specific didactic domains according to extension rules. Generic components include models, metamodels and an algorithm to generate adapted and varied activities using these models.

However, some elements cannot be processed or generated independently of the didactic domain. In particular, handling facts (i.e., declarative knowledge) and the creation of questions about these facts is domain-specific, hence the need for extension. Figure 5.2 presents an overview of the components of our framework. More specifically, the framework is broken down into two parts: “generic” components comprising models, metamodels and source code for activity generation, and “domain-specific” components comprising an extension of models, metamodels to domain-specific knowledge and source code for generation of questions about facts. As described in Figure 5.2, the framework allows the production of domain specific generator that can be used as software component in educational *Roguelite* games. These generators use learner-players profiles to produce adapted activities that must be interpreted by the *game player* or *game engine*, which

1. Implementation is used to refer to software programming. Moreover, generators are components designed to be used by other software. Hence, our target audience for the framework is software engineers.

then have to update the profiles according to learner-players' results.

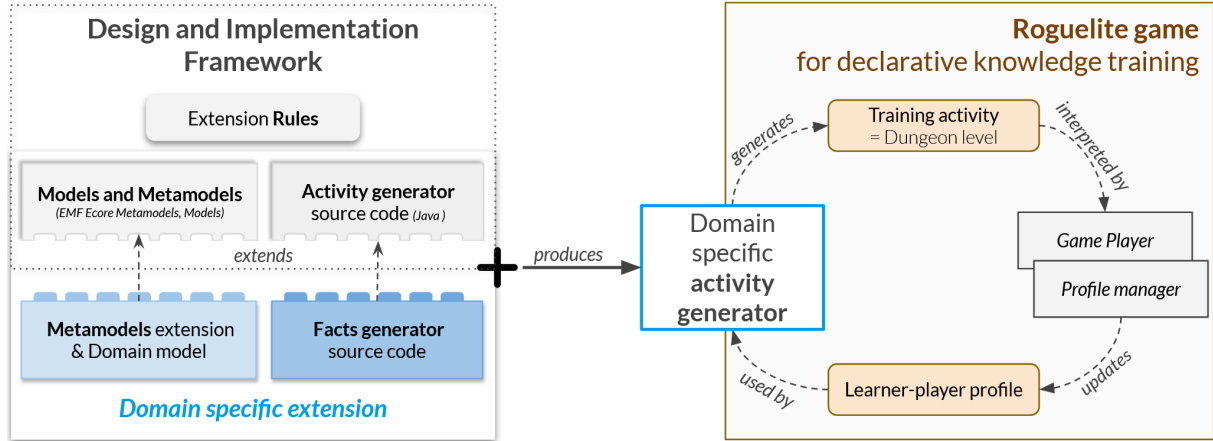


Figure 5.2 – Design Framework Overview

According to our research questions, this framework must aim at declarative knowledge training that is non-specific to a didactic domain. Furthermore, the produced generators must provide varied and adapted activities according to three viewpoints: the teacher, the learner and the player. As a result, our framework and the generators it produces are founded on the respect of five properties (i.e., two related to the framework [FP*i*], three related to the generators [GP*i*):

- FP1** possibility of expressing different didactic domains;
- FP2** possibility of expressing teachers' views on individual learners' training.

- GP1** activities generated must be adapted to the learner's level and results in their training path;
- GP2** activities generated must be adapted to the player's game preferences;
- GP3** activities generated must be varied in terms of both education and game.

5.2 Definition: Game Activity for DK Training

Activity, according to activity theory², can be defined as “a purposeful interaction between subject and object, in a process in which mutual transformations are accomplished” (Carvalho *et al.* 2015) and happens simultaneously at three levels, see Figure 5.3. An activity is realised by a series of *actions* and directed by a *motive*, i.e., the object that the subject wants or needs to reach. Each action is composed of a series of *operations* and

2. Theory that aims at understanding and explaining human behaviour in complex activity contexts, based on the research of Russian psychologists such as Lev Vygotsky and Alexis Leontiev.

is directed by a goal (i.e., object to attain)³. Operations are low-level units performed according to given *conditions*.

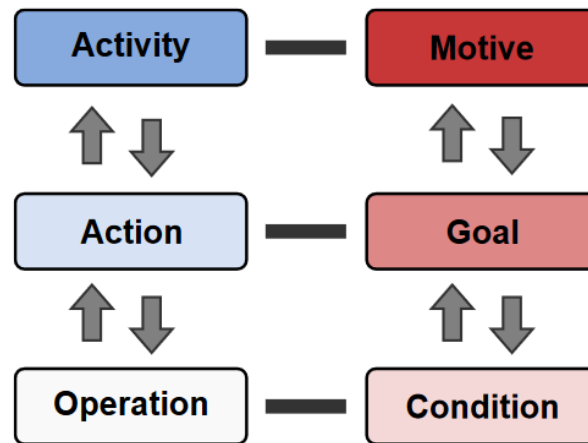


Figure 5.3 – Levels of activity, as defined in activity theory (Carvalho *et al.* 2015)

Following the definition of activity in activity theory, a **game activity for declarative knowledge training** is a dungeon which consists of a set of interconnected rooms in which training takes place. These activities are directed by two motives: an educational motive, e.g., “training on multiplication table 2”, and a game motive, i.e., “Find the exit to the dungeon”⁴. Each room of the dungeons is composed of game and/or training operations (e.g., open a chest, complete a missing fact) and has a goal either: answer a question or avoid traps.

So now that our understanding of a game activity for training is established, our questions are: How is training through game actually carried out? What elements are in the dungeons rooms? Combining fun and educational elements to design activities is not easy (Kiili 2005; Prensky 2005) as both kinds of elements need to be designed conjointly to avoid a game imbalance (i.e., not enough learning or too much, thereby removing the fun aspect). As a result, Prensky (2005) proposed a three-step process to create digital game-based learning: “(1) Find or create a game with great gameplay that will engage our audience, (2) Find the learning activities and techniques that will teach what is required (doing each with the other in mind), and then (3) successfully blend the two”. A similar process can be used to design training game activities that will be generated.

Accordingly, the next sections present the elements of our context defined from this process: 1) the training elements defined and abstracted/generalised from exchanges with experts, particularly experts in mathematics; and 2) the design choices⁵ in terms of game elements, and abstraction made from exchanges with game designers.

3. Subject are usually aware of their goals, but may not be aware of their motive.

4. Other game motives could be considered, but in our current work, only this motive is.

5. As mentioned in Chapter 3, the design of a game content generator requires choices to be made in terms of game design.

5.3 Declarative Knowledge Training Elements

A first exploratory study has been carried out, in the context of the **AdapTABLES** project, with the members of the user group presented in (Laforcade *et al.* 2022)⁶. This study aimed to define the teacher’s perspective on:

- how to train on multiplication tables;
- the adaptations to be considered for multiplication tables training (i.e., sources and targets);
- how to implement the adaptations.

As a result of this study an initial structure for describing training, called **training path** had been defined. Additional informal exchanges with 4 members of the user group highlighted the different ways of training learners on multiplication tables (i.e., possible exercises). These complementary collective exchanges resulted in the definition of five types of training tasks for maths, namely:

- **COMPLETION 1**, i.e., complete an incomplete multiplication fact having one missing element either the result, the multiplicand or the table (e.g., $3 \times ? = 15$, $3 \times 5 = ?$).
- **COMPLETION 2**, i.e., complete an incomplete multiplication fact two missing elements either the (result – multiplicand) pair, the (result – table) pair, or the (table – multiplicand) pair (e.g., $? \times ? = 15$ with a set of given choices [3, 6, 5, 10], $? \times 5 = ?$ or $3 \times ? = ?$ also with sets of given choices⁷).
- **RECONSTRUCTION**, i.e., correctly replace the multiplicand, table and result of a multiplication (e.g., $? \times ? = ?$ with a set of given choices [3, 6, 5, 10, 15]⁶).
- **IDENTIFICATION**, i.e., identify if multiplications are correct or incorrect (e.g., $3 \times 5 = 15$, true or false?);
- **(NON-)MEMBERSHIP IDENTIFICATION**, i.e., identify elements that are or not results or a given table (e.g., [3, 5, 9, 12, 14, 21] which are results of the table 3?).

Furthermore, individual discussions to identify possible training exercises have been carried out with two history-geography teachers and an analysis of exercises proposed to the *Brevet National des Collèges* on history-geography facts, enabled us to define six training tasks for history-geography facts, namely:

- **ASSOCIATION**⁸, i.e., associate two items of a historical fact together either the (event – date/period) pair, the (event – picture) pair, or the (date/period – picture) pair (e.g., “World War II happened between ? and ?”, with a set of given choices [1939, 1914, 1945, 1918]).
- **LEGENDING A MAP**⁹, i.e., complete the legend of a map (i.e., a fact is a symbol and a description text of the map) having one missing element either the symbol or the

6. This study has been carried out before the beginning of this thesis.

7. Incorrect choices must be selected wisely to avoid other correct combinations.

8. When several facts are questioned at once, incorrect proposals are usually those of the other facts.

9. When several facts are questioned simultaneously, every proposal is correct and must be placed at the right location on the map (i.e., it is the positions chosen by the learner that evaluate their answers). Moreover, some or all of the elements may be missing.

description text.

- IDENTIFICATION¹⁰, i.e., identify if historical facts are correct or incorrect. (e.g., “World War II happened between 1914-1918”, true or false?);
- TIMELINE, i.e., complete the timeline having several missing historical facts.
- NAME AND LOCATE¹⁰, i.e., complete a map having several missing elements (e.g., cities, regions, countries).
- MEMBERSHIP IDENTIFICATION, i.e., identify elements that share or not a given property (e.g., [France, Switzerland, England, Spain] which are country of the European Union?, [Paris, Nancy, Bordeaux, Avignon] which are urban areas of France?).

Additionally, these exchanges have also been used to discuss and approve the training structure, as there were no disagreements with the concept of training path.

From this work, an observation had been that some tasks appear to be similar in both domains. Therefore, in a perspective of genericness, an abstraction of the training tasks from the two domains resulted in the definition of four generic task types¹¹.

5.3.1 Training Path

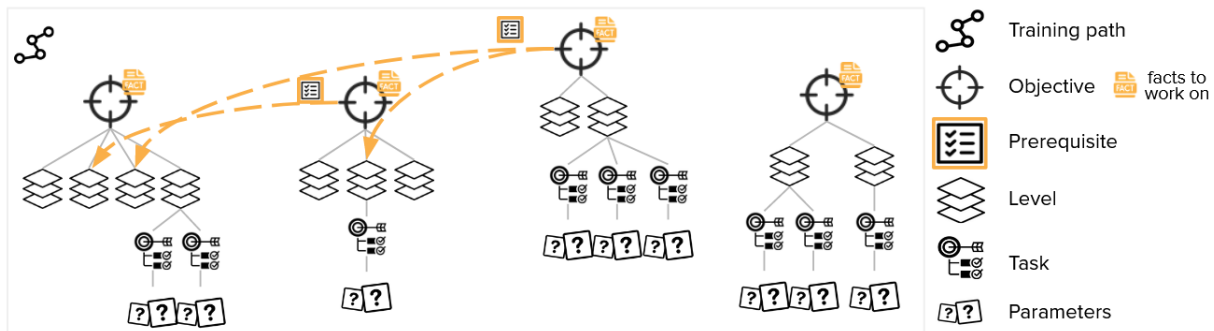


Figure 5.4 – Description of the training structure

A training path, see Figure 5.4¹², is represented by a set of objectives ordered by prerequisite relationships. An objective is composed of a description such as “Training on the multiplication table of 2”, the set of knowledge to work on (e.g., raw facts of the table of 2), and is broken down into progressive levels of difficulty. Prerequisites are conditional relations between an objective $O1$ and a level of an objective $O2$. These conditions consist of the percent of facts encountered (i.e., facts that have been questioned/presented to the learners) and a percent of achievement to reach. Therefore, an objective is considered

10. Note that, as for LEGENDING A MAP, it is the positions chosen by the learners that impact the correctness of their answers.

11. Please note that although our abstraction is based on two domains, we kept in mind facts and questions that may come from other domains, e.g., facts about the solar system.

12. All levels have tasks. However, to avoid cluttering up the illustration, it is not depicted here.

eligible when the conditions of its prerequisites are met (i.e., an objective without prerequisites is by default eligible). Each level and prerequisites have two main objectives: 1) allow working on new objectives even if the previous one has not been finished, but a sufficient level of training has been reached, and 2) allow defining more difficult/advanced training levels that are not mandatory to progress. In order for objectives to be unlocked/available, conditions of prerequisites must be reached by learner-players. Each level is itself broken down into training tasks (e.g., “Level 1: Completion 1 with search for the result, Identification by choice of the correct facts”). A task is defined by its type and parameters. The achievements of the levels are considered from both a percent of encountered facts and a percent of achievement to reach. This reduces the training time needed to achieve a level in cases where the set of facts to be questioned would be substantial.

5.3.2 Training Task Types

Drawing on the tasks defined for different domains, initially three generic types of task have been specified (i.e., independent of a specific didactic domain): Completion, Identification, and Membership Identification. Following an initial iteration of our work, presented in Chapter 6, carried out using mathematics training tasks, a lack of task for ordering facts (e.g., ordering the planets according to their distance from the sun, ordering the stages of meiosis in chronological order) emerged. Since the abstraction using the history-geography tasks was not sufficient to cover this particular need, an additional task type was added to cover this possibility for other domains (i.e., order). Our generic types of task do not claim to be exhaustive, only to cover a variety of possible question forms for declarative knowledge training. Below are our four generic task types:

1. **Completion:** complete a fact that having one or several missing elements (e.g., complete $3 \times ? = 15$, reconstitute $? \times ? = ?$ using elements in $[3, 6, 5, 10, 15]$, complete *World War II happened between ? - ?*);
2. **Order:** order facts based on a given heuristic (e.g., chronologically order: *World War II, Storming of the Bastille, Treaty of Rome*, order starting from the closest to the sun: Mars, Earth, Jupiter, Neptune);
3. **Identification:** attest of the validity or invalidity of one or several facts (e.g., true or false: $3 \times 5 = 15?$, Did *World War I* happen between 1915–1919?);
4. **Membership Identification:** identify elements that share or not a given property (e.g., $[3, 5, 9, 12, 14, 21]$ which are results of the table 3? [France, Spain, England, Switzerland, Italy] which are part of the European Union?);

5.3.3 Training Tasks Parameters

Tasks and task types are described by parameters. In particular, task types are described by their response modality (i.e., choice from a set of proposals or direct input awaited), the maximum time allowed to answer, and the number of consecutive successful answers. In order to avoid correct answers by chance (i.e., random success), teachers have

proposed to validate the retention of a fact by considering several consecutive successes for a same question related to a task. Consecutive successes do not mean that a same question is interrogated twice in a row in a same activity, but that the success for successive appearances of a same question over multiple activities is considered (e.g., if 3 consecutive successes are required and the learner succeeded twice on a question but missed the third time, their number of consecutive successful answers is reset to 0).

However, domain-dependent tasks, i.e., subtasks of generic task types, have parameters specific to the didactic domain concerned. Let's take as an example the COMPLETION 1 task, defined for multiplication tables as completing an incomplete fact that has one missing element (e.g., $3 \times ? = 15$, $15 = ? \times 5$, $3 \times 5 = ?$). Several parameters are domain-dependent in this example:

- the missing element;
- the position of the result or equal symbol;
- the multiplier interval;
- the position of the multiplicand.

Several of these parameters (i.e., result and multiplicand position, multiplier interval) enable the construction of various forms of facts (i.e., multiplication tables can be built differently). In this case, the parameters are common to every task of a level. As a result, such parameters are directly associated to the level and not to the tasks (i.e., to avoid repetition). Table 5.1 presents the possible parameters for multiplication tables.

Parameter	Possible Values	Some Examples
Multiplicand Position	Left \vee Right	$1 \times 2, 1 \times 3, 1 \times 4.. \vee$ $2 \times 1, 3 \times 1, 4 \times 1..$
Result Position	Left \vee Right	$1 \times 2 = 2 \vee 2 = 1 \times 2$
Multiplier Interval	Integer Min/Max in $[1, 12]^{13}$	$[1, 5] \vee [5, 10] \vee [1, 12]$
Missing Element	Result \vee Multiplicand \vee Operand	$1 \times ? = 2 \vee ? \times 2 = 2$ $\vee 1 \times 2 = ?$
Response Modality	Choice between proposals \vee Input	$[2, 3, 4, 5]$
Consecutive Answers	Positive integer	$1 \vee 2 \vee 3$
Max Response Time	Time in seconds (integer)	$10s \vee 30s$

Table 5.1 – Parameters for multiplication tables

5.4 Roguelite-oriented Game Elements

Although generation is independent of a game, in the way that generators can be seen as independent software components, the design of game activities to be generated requires making game design choices. More precisely, generation requires knowledge of the elements of the game that impact it (i.e., generation choices depend on these elements). As a result, in the context of the AdapTABLES project, an interest had been expressed in the design of educational Roguelites games, particularly by studying existing Roguelites

(i.e., testing existing games, watching gamers reviews of game mechanics). This work not only highlighted the fundamental questions that must be addressed for the design of an educational **Roguelite** game, but also guided the definition of our design choices.

Nonetheless, these questions focus on the overall design of educational **Roguelite** games, but as [Prensky \(2005\)](#) stated “Although learning games can fail as real games in many ways, the failure happens mostly commonly in their lack of gameplay—the fun things that the player gets to decide, control, and do”. Therefore, in a second phase, our interest had been focused on the design of gameplays for declarative knowledge training. For this purpose, informal interviews with two game designers led us to define gameplay mock-ups. An abstraction of these gameplays resulted in the creation of our concept called *gameplay categories*.

5.4.1 Analysis method for Roguelite Design

To design educational **Roguelites** games, both the educational and gaming dimensions must be equally considered (i.e., none should be neglected). Our analysis and design of a game, in the context of the **AdapTABLES** project, led us to the conception of a method for the analysis of design needs for educational **Roguelites**. This method provides a means of specifying needs according to both dimensions through specific criteria ([Lemoine et al. 2023a](#); [2024c](#)).

Criteria		Educational Dimension	Game Dimension
Generation	What elements are generated?		
	When are these elements generated?		
	Based on what criteria are they generated? (i.e., sources of generation)		
Death/Hurt	Under what circumstances can the avatar be injured or die?		
	What are the consequences of being injured or killed?		
	Where can the avatar sustain injuries or be killed?		
Variety	Which elements exhibit variation?		
	How do these elements vary? (i.e., are the variations triggered by player action? Are they random? Is it a combination of both? Are they guided by heuristics?)		
Progress	What is preserved or carried over between each death? (i.e., which elements?)		
Difficulty	What factors contribute to increasing or decreasing the difficulty?		
	How is the difficulty progression designed? (i.e., if multiple elements affect the difficulty, what is the sequence in which they occur?)		

Table 5.2 – Grid for the Design Needs Analysis of educational **Roguelite** games

To design a **Roguelite**, the initial step involves specifying the game mechanics. As previously indicated, the three main mechanics of **Roguelites** are: generation, permanent death, and progression. Therefore, specifying these mechanics involves defining how the game world is generated (e.g., what is generated and how), when permanent death occurs, and how progression works (e.g., which elements are carried over). Additionally, because the generation mechanism is stochastic, within this context, it encompasses specifying elements that vary. Moreover, in games, as in learning, an essential concept is the progression of difficulty as it is crucial to define how difficulty increases and when. As a result, we have identified these five main mechanics (**Generation, Death/Hurt, Variety, Progress, and Difficulty**) as criteria for analysing the needs that must be specified from both dimensions. Each criterion consists of a series of questions that relates to the same mechanism. Each question should be answered in order to clarify the design needs of the game. Table 5.2 presents a structure to complete for the needs analysis. Each row represents a criterion and is divided into X sub-rows (i.e., one sub-row per question). Each column represents a dimension (i.e., one for the game, another for the educational dimension). Columns can be merged if both dimensions have common information.

5.4.2 Design Choices for Activity Generation

In order to design a **Roguelite** for multiplication tables training, our analysis framework has been applied to define the game design choices. As a reminder, the **AdapTABLES** project is centred around a prototyping design method, i.e., an iterative process that involves creating a game step by step (i.e., adding features gradually). The idea is to use the framework to design the various prototypes.

A first iteration allowed the design of a first prototype in which the game flow, see Figure 5.5, is viewed as a series of game sessions (i.e., a temporal session beginning when the player starts the game and ends when he stops it). A run is a succession of successfully completed game levels of increasing difficulty that ends once the end of the game has been reached or the avatar has died. A game level is a dungeon level.

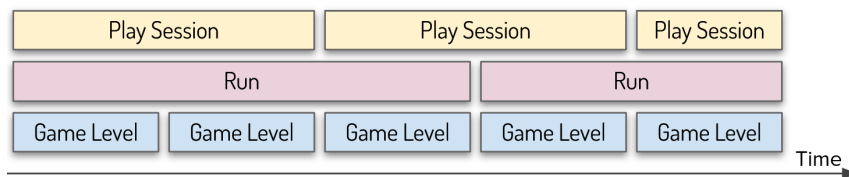


Figure 5.5 – AdapTABLES game flow

This first prototype included three gameplay variants for a single completion task (Lemoine *et al.* (2023a) and Lemoine *et al.* (2024c) detail the analysis). On the basis of the feedbacks gathered from tests under ecological conditions, a second analysis had been conducted. Table 5.3 summarises the design choices after the second application of the analysis framework.

Criteria		Educational Perspective	Game Perspective
Generation	<i>Q1: What?</i>	One task and one or several questioned facts per room-with-question	Dungeon structure + rooms
	<i>Q2: When?</i>	When a new game level is required	
	<i>Q3: Based on?</i>	All tasks set-up Current progress among possible facts Task parameters have priority on activated game elements if conflict	Previous level number and state Equipped items
Death/Hurt	<i>Q4: When?</i>	Incorrect answers or time out	Being touched by foes, falling into holes
	<i>Q5: What?</i>	Injuring causes heart lost, no more hearts causes death	
	<i>Q6: Where?</i>	Question rooms	Only rooms with no question
Variety	<i>Q7: What?</i>	Facts	Different types of rooms, types of gameplays, types of elements
	<i>Q8: How?</i>	Progress and past results and in relation to the tasks \iff gameplays mappings	Based on the available equipments, gameplays, elements,
Progress	<i>Q9: What?</i>	Success or failure on met questioned facts	Coins collected during successful game levels + purchased items
Difficulty	<i>Q10: What?</i>	Questioned facts	Dungeon level length + curses
	<i>Q11: How?</i>	In relation with the task parameters related to the objective/level	According to previous level number and state

Table 5.3 – Design choices for AdapTABLES

The design choices from this second analysis that are important for generation are highlighted below:

- Generated dungeons contain rooms with questions and rooms with traps. This separation aims to allow learners not to be disturbed or injured (i.e., their avatar) by “fun” game elements (i.e., that do not have a training purposes) while answering questions about facts. The teachers mentioned that the game perspective should not or the least possible hinder the training.
 - Rooms with questions are related to a training task found in the learner-player’s training path.
 - Rooms without questions are “fun” rooms with only game elements such as enemies and traps.
- Since **Roguelites** are often based on a purchase/activation mechanism, the selection of gameplays is adapted according to the purchased items (i.e., player preferences) made and activated. Hence, any gameplay type that is not appreciated can be deactivated (i.e., unless it’s a default gameplay) by deactivating the corresponding purchased item.
- Training difficulty should increase as the learner progresses through their training path, i.e., a dungeon trains the learner on an objective/level pair of their path. However, a dungeon targeting a pair *objective 2/level 1* can be followed by a pair *objective 1/level 3*. These levels are not necessarily considered as having an increase in difficulty by learners since different knowledge are worked on, it is therefore very

subjective and context-dependent. By contrast, game difficulty increases in a run by gradually increasing the number of dungeon rooms with and without questions (e.g., starting at 5, then 7, then 9, and so on).

- *Curses* are regularly encountered in Roguelites, e.g., dark level, labyrinthine dungeon, one life only. This means that the progression of the game is structured around different minimum thresholds, whereby each threshold unlocks curses that may or may not occur over the course of the generated dungeon levels. For instance, considering thresholds every three dungeon levels, the generation of level #10 may involve a maximum of three curses or, with a little luck, none.

These design choices have an impact on the generation. Even though they have been made as part of the project, we consider them for the generation of any declarative knowledge training activities independently of a didactic domain (i.e., for the framework) since these choices are not specific to mathematics or multiplication tables training. Although our inspiration came from mechanics found in Roguelites and advice provided by game designers, these choices are subjective and other choices could have been made. Evaluating these design choices is out of the scope of this PhD thesis, but it is a relevant future perspective for the AdapTABLES project.

5.4.3 Gameplay Categories

Reflecting on Prensky (2005)'s statement, the design phase led us to focus on gameplay design. Our aim is to offer a wide variety of gameplays for each type of training task. Hence, the previously given definition of gameplay, *fun things that can be controlled, decided, and done by players (ibid.)*, can be refined to: **descriptions of contextualised actions that players can perform to interact with the environment, through their avatar, in order to answer questions.**

First, a collective discussion had been held with four teachers and a didacticien about some gameplay ideas from which a constraint emerged: *the gameplays must be simple*, i.e., interactions to answer have to be quick, to avoid interfering with training. Then, informal interviews were conducted with two game designers to gather ideas for the design of gameplay mock-ups. An observation could be made from these mock-ups: *some gameplays seemed to belong to the same category*, e.g., breaking a pot or opening a chest bearing an answer are similar ways of selecting an object. That observation is consistent with the game classification proposed by Djaouti *et al.* (2008), which consists of describing games in terms of gameplay bricks (i.e., categories of actions that can be performed within the games). Consequently, further reflection resulted in the definition of five gameplay categories, see Figure 5.6, in the context of Roguelite training games:

1. SELECT: select (e.g., touch, kill, break, open) objects having an answer, through avatar actions;
2. MOVE: correctly place objects at specific locations through avatar actions;
3. ORIENT: orient objects (e.g., rotate), through avatar actions, towards an answer;
4. POSITION: move the avatar to the necessary positions to choose or type answers;

5. DIRECT RESPONSE: no action is required through the avatar, learners directly type down their answer by using an input device (e.g., keyboard).

Like the task type, these categories do not claim to be exhaustive, only to provide a variety of gameplays.



Figure 5.6 – Example of mock-ups by gameplay categories

5.5 Synthesis

In summary, the proposed activities consist of a set of interconnected rooms (i.e., linear or labyrinthine), whose game objective is to find the exit and whose training objective is related to the learner’s training path (i.e., a path defined by a teacher). A room is associated with a gameplay that aims purely at entertainment (i.e., avoiding enemies/traps) or education (i.e., answering a question that matches a training task). These gameplays are implemented as game elements with which players interact through an avatar. Figure 5.7 shows the structure of our activities as defined in Figure 5.3.

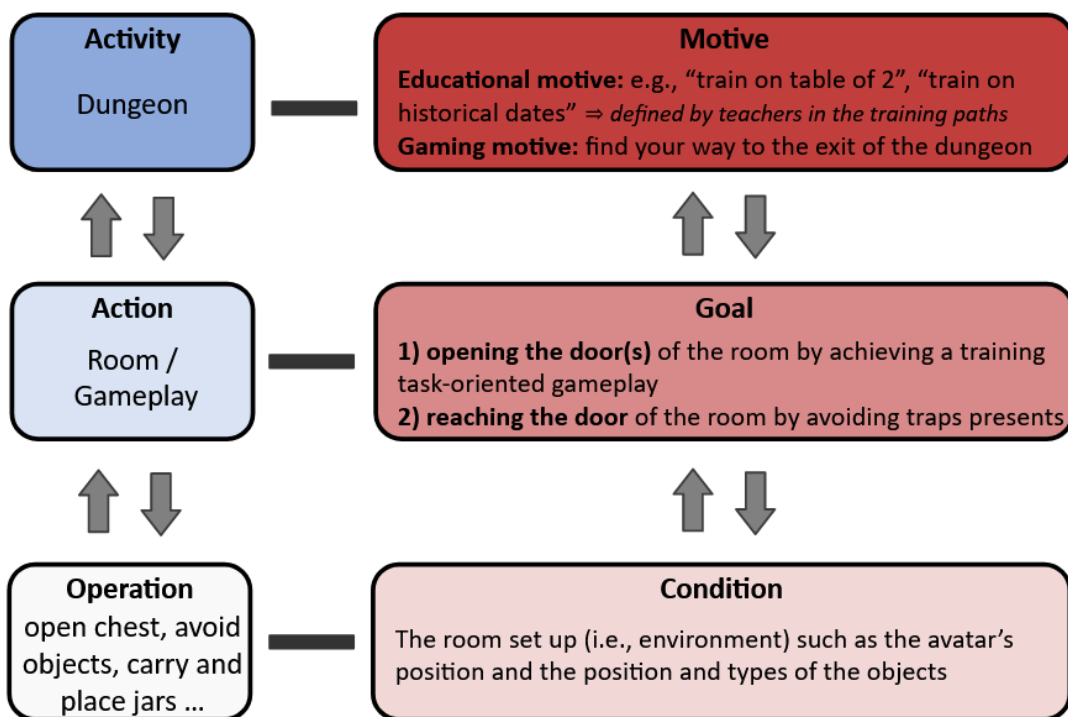


Figure 5.7 – Illustration of the levels of proposed activities from activity theory

Our training game activities propose task oriented gameplays. The task types and gameplay categories involved have been defined based on exchanges with experts (i.e., teachers, didactic experts, game designers) and do not claim to be exhaustive. Figure 5.8 describes the interactions with experts involved in specifying the training and the gameplays. Gameplay categories have been defined to offer a wide variety of possible game interaction to answer questions, while task types have been designed to meet the training needs which have been expressed by our experts. As a result, these task types could be refined according to other didactic domains or other experts’ viewpoints, and the game categories proposed could probably be extended.

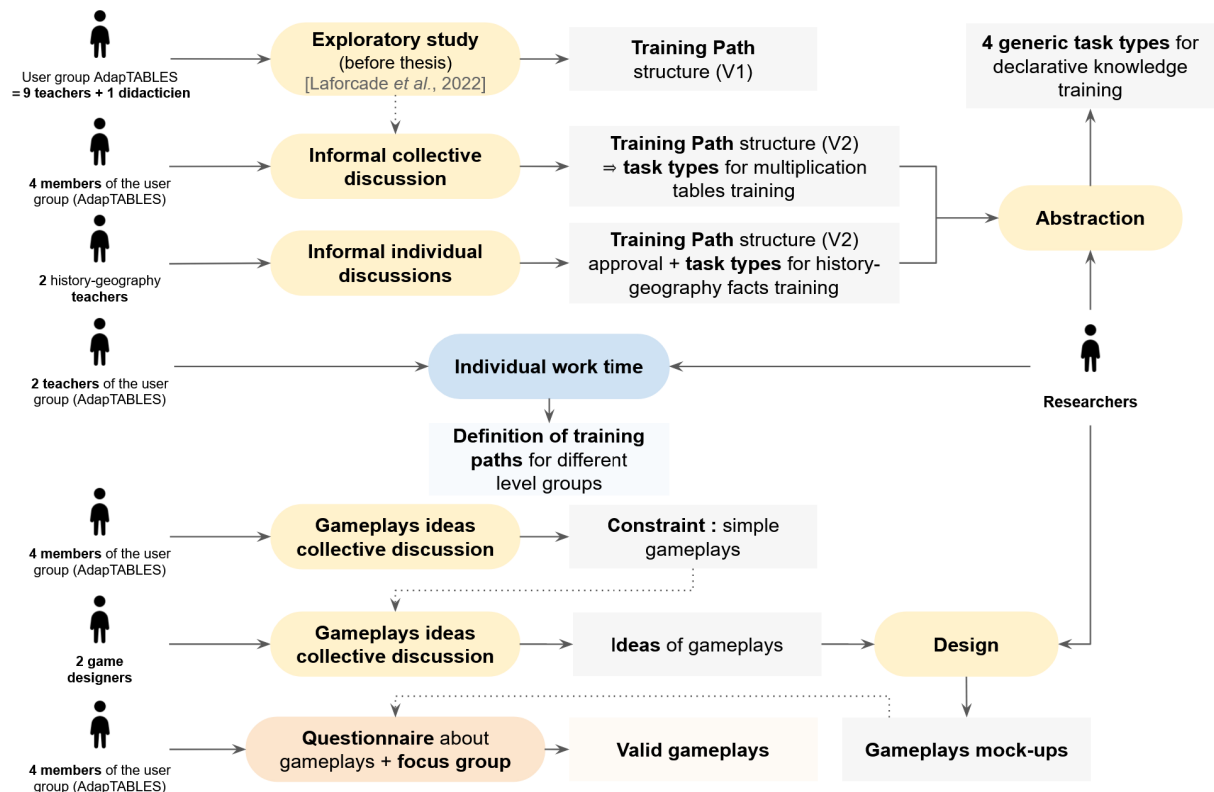


Figure 5.8 – Overview of exchanges made with experts to specify training and gameplays elements (in orange : exchanges described in Chapter 6, in blue : exchanges described in Chapter 9)

Now that the activities and their elements are clearly defined, the third step of Prensky (2005)'s process consists of successfully blending educational and game elements together to build the activities. Accordingly, our question is as follows: *How to coherently associate the gameplay categories and task types to drive activity generation?* The following chapter addresses this issue.

MAPPING GAME AND EDUCATIONAL ELEMENTS

Contents

6.1 Existing Work	78
6.1.1 Relations Between Dimensions	78
6.1.2 Methods to Define Relations Between Dimensions	79
6.1.3 Synthesis	80
6.2 Mapping Approach Development	80
6.2.1 Identification of the <i>Pivot</i>	81
6.2.2 Mapping Task Types onto Gameplay Categories	84
6.3 A Systematic Mapping Approach	88
6.3.1 Proposed Mapping Approach	88
6.3.2 Relations Between Task Types and Gameplay Categories	90
6.3.3 Evaluation of the Relations	90
6.4 Synthesis	92

As a result of the ever-growing interest in game-based learning, a new research topic has recently emerged regarding the alignment of educational and game elements to ensure a balance and coherence in the content delivered. Although such an alignment is necessary for good educational game design, it is a complex task (Kiili 2005; Prensky 2005) that requires to be addressed from a transdisciplinary perspective. Notably, because multiple context-dependent variables (i.e., didactic domain, targeted knowledge or game genre) must be considered.

Previously, Chapter 5 defined the game activities for declarative knowledge training and their components. These activities are dungeons composed of rooms with gameplays designed to answer training task-oriented questions. To ensure variety in terms of game elements within an activity, it is necessary to identify the gameplays (i.e., or at least a set of gameplays) that are suitable for each training task. Indeed, knowledge about these relationships is essential at the design phase to guide the identification of practical gameplays for each specific task, and at the runtime to control the generation process.

Consequently, the question is the following: **how to determine and specify the relationships between task types and gameplay categories necessary to automatically create activities?** Figure 6.1 illustrate the research question. Our assumption is that answering that question at a higher level of abstraction (i.e., task types and abstract gameplays) may enable the reuse of the relationships in various declarative knowledge contexts.

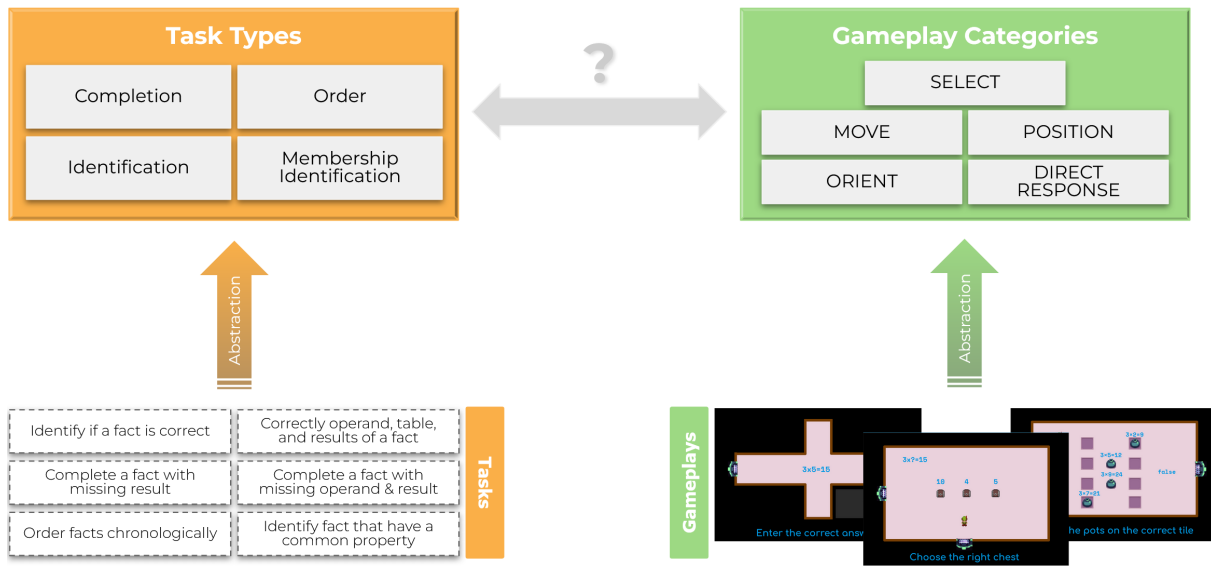


Figure 6.1 – Illustration of the educational-game dimensions *mapping* research question

This chapter aims at presenting existing works on educational and game alignment¹ and introduce our solution to specify machine-readable relations to map our task types and gameplays categories coherently. First, we provide an overview on the existing works on educational-game alignment (i.e., definition of relations between element and methods to specify relations). Next, we present the proposed method based on the use of numerical questionnaires/quizzes as a pivot point (Lemoine *et al.* 2023b; 2024b)².

6.1 Existing Work

The alignment of educational and gaming dimensions has been tackled from two angles in the literature: 1) proposition/definition of relationships between gaming and educational elements, and 2) proposition of methods for specifying relationships between educational and gaming elements for analysis or design purposes.

6.1.1 Relations Between Dimensions

Numerous works have identified relationships between various educational and game elements. A pioneer is Prensky (2005) who proposes relations between game styles (e.g., action, role-play, adventure, flashcards, detective games), knowledge to be learned (e.g.,

1. Alignment can be seen as ensuring that the overall design of a game supports the educational objectives. Mapping generally is more about the specific links between gameplay and learning objectives. Mapping can be seen as a “low-level alignment” that contributes to the overall alignment.

2. Lemoine *et al.* (2024b) is an extended version of Lemoine *et al.* (2023b) and present a complete version of our mapping approach. Thus, some sections of this chapter are exact extracts of this article.

facts, skills, judgement, behaviour) and learning activities (e.g., questions, coaching, imitation, observation). Rapeepisarn *et al.* (2008) extend Prensky (2005)'s work by linking already defined relations to Chong *et al.* (2005)'s learning styles (i.e., activists, reflectors, theorists, and pragmatists). Similar work has been developed by Dondi and Moretti (2007), who propose to link knowledge types and learning objectives to high-level game features (e.g., content engine, evaluation engine, chance) that the game should possess, as well as game types (e.g., puzzle, quiz, action, driving) and the number of players (e.g., one, many, one vs PC) expected. Khenissi *et al.* (2016) define relations between four Felder-Silverman learning styles dimensions (i.e., sequential, global, sensing, intuitive) and four game genres (i.e., game based on puzzle, god games, casual games, games based on simulation). Likewise, Jafari and Abdollahzade (2019) propose relations between Felder-Silverman learning styles dimensions (i.e., sequential, global, intuitive, visual) and three game genres (i.e., puzzle games, god games, simulation games). Sherry (2010) identifies relations between eight game genres (i.e., shooters, action/fantasy/role-playing, sports/sims, puzzle, quiz) and the six levels of Bloom (1956)'s taxonomy. Based on a literature review, Lameris *et al.* (2017) associate learning attributes (e.g., information transmission, collaborative, discussion and argumentation) with game attributes (e.g., task description, role-playing, nested dialogues), learning outcomes (e.g., remembering, understanding, applying), feedback/assessment (e.g., progress, affect, formative and/or summative) and teacher roles (e.g., player, motivator, facilitator). Some more context-dependent approaches have been developed such as the one proposed by Kanaan *et al.* (2022) which defines relations between competencies of the PIAF competency Framework (i.e., classification of competencies related to the development of computational thinking in basic education, established in 2021) and context-dependant gameplay features (e.g., enable library, select the appropriate exit, select wrong blocks).

6.1.2 Methods to Define Relations Between Dimensions

Some works have proposed frameworks to specify relations between the education and game dimensions for existing game analysis. Degens *et al.* (2015) propose a 3-dimensional model for analysing relations to identify discrepancies between game, learning, and users from three view points: Learning vs User (i.e., determine the discrepancies between user properties and learning goals), Game vs User (i.e., determine the discrepancies between game mechanics and user properties), Game vs Learning (i.e., determine discrepancies between game mechanics and learning goals).

Other works have proposed frameworks to specify relations between the education and game dimensions for helping the design of new games. Hall *et al.* (2014) propose a framework³ to guide the designer in specifying the transition from learning content to core-gameplay. Their framework is composed of five categories (i.e., goal, choice, action, rules, feedback) in which a series of questions need to be answered from a real-world and

3. In this section, the term framework, used by the authors, refers to: “structured method that helps to systematically organize and analyse data, theories, concepts, and ideas”.

a game-world perspective to guarantee game and education content alignment. For the design of a game for learning programming, [Debabi and Champagnat \(2017\)](#) propose an approach to associate categories of programming concepts with game missions based on a modelling approach. Missions are modelled according to the categories they cover (i.e., a percentage is associated to each category covered). Using a Trace-Based Subjective logic algorithm, the categories of concepts to work on are selected for a learner, then a TF-IDF algorithm is used to select the mission best suited to the player. However, this low-level mapping is strictly context-dependent and cannot be reapplied to other contexts.

Finally, some works have proposed frameworks to specify relations between the education and game dimensions for game analysis and design. [Gosper and McNeill \(2012\)](#) propose a theoretical framework called MAPLET having a bidirectional representation of the learning environment (i.e., horizontal dimension consists of the alignment principles, vertical dimension consists of learners' intellectual maturity). This framework allows matching aims (and learning outcomes), processes, learner expertise and game genre based on pre-defined relations. [Lim et al. \(2013\)](#) propose the LM-GM Framework that uses a concept called Serious Game Mechanics to support the transition between learning mechanics/objectives (e.g., guidance, participation, observation) to game mechanics (e.g., orientation, collaboration, exploration). These mechanics are high-level concepts because they can have many practical in-game implementations.

6.1.3 Synthesis

Although works which define relations between educational and game concepts/elements are very useful for the alignment during the general design of a game, due to the high-level concepts that they link, they do not provide guidance for the mapping of elements during the creation of an educational game activity. Additionally, existing methods to guide the definition of relations are either based on high-level concepts that can have many implementations or strictly context-dependent approach that cannot be directly reused (i.e., do not allow the definition of machine-readable relations). Therefore, these methods are more oriented towards analysing existing games or assisting in the high-level design of games, rather than specifying relations for low-level design purposes.

As a result, to answer our research question, our proposal consists of **a systematic method to guide the specification of machine-readable relationships describing the conditions under which gameplay categories are compatible with task types** ([Lemoine et al. 2023b; 2024b](#)). The following sections describe: 1) how the mapping method was developed and 2) the resulting method for mapping gameplay categories and task types, as well as the resulting relationships.

6.2 Mapping Approach Development

Declarative knowledge training and assessment are commonly performed through questionnaires and quizzes. Compared to paper quizzes, digital quizzes feature user interac-

tions that are closer to the ones found in basic training games (e.g., a multiplication table training game in which correct answers make an avatar run faster or jump onto higher platforms). Accordingly, using exercise types from numerical questionnaire formats as a pivot point seems promising, especially since using existing material can reduce subjectivity. Figure 6.2 illustrates the general idea of the proposed mapping approach.

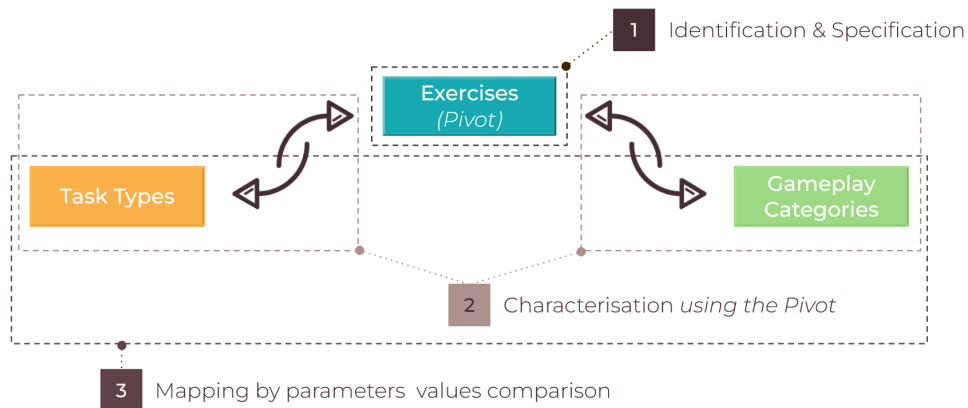


Figure 6.2 – General idea to map task types onto gameplay categories

Elaborating our approach required several stages, beginning with an analysis of questionnaire/quizzes design formats to define the types of existing exercises (i.e., pivot, see step 1 in Figure 6.2). From there, the following questions were raised: 1) How to draw a parallel between the types of tasks and the exercises identified? and 2) How to draw a parallel between the gameplay categories and the exercises identified? Since interactions offered by quiz exercises are closer to game interactions and each concept (i.e., task types, gameplay categories and exercises) is already characterised by its response modality (e.g., entering an answer, choosing between several proposals), our idea is to use exercise types in order to identify criteria and parameters to specify each concept and ease the identification of mappings (see step 2 in Figure 6.2). Then, the values of the parameters of task types and gameplay categories are compared to identify matches (see step 3 in Figure 6.2).

This section presents how the mapping approach has been developed, whereas the next section presents the proposed mapping approach, and the resulting relations between dimension in our context (i.e., application of the mapping approach to the defined task types and gameplay categories presented in Chapter 5).

6.2.1 Identification of the *Pivot*

Foremost, in order to extract existing types of exercise, six tools allowing the creation of digital questionnaires/quizzes, extracted for the most part from Learning Management Systems (LMS), have been analysed:

- the eponymous and proprietary format from the *itsLearning* (#1) LMS;
- *GIFT* (#2) a mark-up language for describing tests that is used within the *Moodle* LMS;

- *Performance Matters Assessment and Analytics* (#3) format associated with the *PowerSchool* LMS;
- *NetQuizzPro* (#4) a software allowing the creation of questionnaires;
- *QTI* (Question & Test Interoperability specification) (#5) from the IMS global learning consortium that defines a standard format to exchange and store assessment content;
- *Tactileo – Maskott* (#6) format associated with the French pedagogical platform of the same name.

Our analysis mainly consisted of finding the different possibilities (i.e., form of questions possible and their parameters) offered by the exercises of the formats. A comparison of the found possibilities led to the definition of twelve different types of exercises useful for declarative knowledge (i.e., only exercises for which result verification can be automated). Exercises from different formats having an identical type of statement, number of desired answers and for which the interaction to answer was similar, have been merged to create a single type of exercise. Moreover, even though some formats combine several exercises into one (e.g., *itsLearning* merges multiple choice and answers), our choice is to consider them independently. Additionally, having intruders (i.e., elements that should not be associated) had been requested by domain experts, but none of the “Associate” type exercises analysed from the formats offered this possibility. Consequently, in our definition it has been considered as a possibility. The exercise types defined are as follows:

- ⇒ *Alternative*: choosing one answer between 2 options;
- ⇒ *Multiple choice*: choosing one answer between X (i.e., $X \geq 2$) options;
- ⇒ *Multiple responses*: choosing Y (i.e., zero or more) answers between X (i.e., $X \geq 2$) options;
- ⇒ *Short answer*: enter the correct answer. Multiple form of answers can be accepted, e.g., for example, *How much is 3 times 5?* as two possible answers, which are *15* and *fifteen*;
- ⇒ *Fill-in-the-blanks*: enter for each gap of a text the wanted “short” answer;
- ⇒ *Fill-in-the-blanks choices*: choose for each gap of a text the correct answer from a list. Each gap can have an associated list of options, or one list can be associated to all gaps;
- ⇒ *Reconstruction*: reassemble each significant element of an information;
- ⇒ *Associate-Group*: associate elements from a list or multiple lists together. The association can be done by pairs, or not. The elements can be associated with zero to several other ones;
- ⇒ *Order*: replace a set of information in the correct order (i.e., following a heuristic);
- ⇒ *Graphic choice*: point or locate X (i.e., $X \geq 1$) elements on a picture.
- ⇒ *Graphic identification*: write the correct label for each area-to-complete of a picture;
- ⇒ *Graphic association*: associate the correct labels to X areas of a picture.

As a reminder, these types of exercises aim to deal with declarative knowledge in general. As a result, some exercises offer a more visual approach that could be useful in the context of geographical facts, for example. Table 6.1 presents an overview of which format provides which exercises. It is worth mentioning that none of the formats provides every possible form of exercise.

	<i>itsLearning</i>	<i>GIFT</i>	<i>PMAA</i>	<i>NetQuizzPro</i>	<i>QTI</i>	<i>Tactileo</i>
Alternative	✓			✗	✗	✗
Multiple Choice	✓	✓	✓	✓	✓	✓
Multiple Resp.	✓	✓	✓	✓	✓	✓
Short Answer	✗	✓	✓	✗	✓	✓
Fill-in	✓		✓	✓	✗	✓
Fill-in Choice	✓		✗	✓	✓	✓
Reconstruction	✗	✗	✗	✓	✗	✗
Association						
Order	✓	✗	✓	✓	✓	✓
G. Choice		✗		✗	✓	
G. Identification	✗	✗	✗	✓	✗	✗
G. Association	✗	✗	✗		✓	✓

Table 6.1 – Exercises by quiz format (✓ present; ✗ absent; | present but incomplete)

Exercises can be characterised by several parameters, see Table 6.2, namely: their *interactions*, their *response modality* (i.e., input or choice), their *statement type* (i.e., format of the question asked), the *number of answers* desired, and the *number of propositions* presented (i.e., if the response modality is “Choice”). From our analysis, six types of interaction have been identified:

- *Select Y From X* (i.e., the learner must select Y answers from a set of X values);
- *Y (Select 1 from X_1 to X_Y)* (i.e., the learner must make a selection of one answer from each set of proposals);
- a variant is *Y (Select 1 from X)* (i.e., the learner must select Y answers, one by one, from a set of proposals);
- *Write X* (i.e., the learner has to enter X answers);
- *Order X* (i.e., the learner must order X elements correctly);
- *Point X* or *Locate X* (i.e., the learner must point X elements on a picture or locate them);
- *Match Y with X 1-to-1* or *Match Y with X* (i.e., the learner must associate elements from Y with those from X by pairs or not).

Furthermore, three types of statements have been found: 1) classic statement (i.e., text question that can be supported by an image), 2) graphical statement (i.e., classic statement accompanied by a graphical element with which interactions are required to answer) and 3) fill-in-the-blank statement (i.e., classic statement with embedded fill-in areas).

	Number of Facts	Statement Types	Response Modality	Number Answers	Number of Choices	Interactions
Alternative	1	Classic	Choice	1	2	<i>Select 1 from 2</i>
Multiple Choice	1	Classic	Choice	1	2 to ∞	<i>Select 1 from X</i>
Multiple Resp.	1	Classic	Choice	0 to ∞	2 to ∞	<i>Select Y from X</i>
Short Answer	1	Classic	Input	1	0	<i>Write 1</i>
Fill-in	1	Fill-in	Input	1 to ∞	0	<i>Write Y</i>
Fill-in Choice	1	Fill-in	Choice	1 to ∞	2 to ∞	<i>Y (Select 1 from X)</i> <i>Y (Select 1 from X_1 to X_Y)</i>
Reconstruction	1	Fill-in	Choice	2 to ∞	2 to ∞	<i>Match Y with X 1-to-1</i>
Association	2 to ∞	Classic	Choice	2 to ∞	4 to ∞	<i>Match Y with X 1-to-1</i> <i>Match Y with X</i>
Order	2 to ∞	Classic Fill-in	Choice	1 to ∞	2 to ∞	<i>Order Y</i>
G. Choice	1 to ∞	Graphic	Choice	1 to ∞	2 to ∞	<i>Point Y or Locate Y</i>
G. Identification	1 to ∞	Graphic	Input	1 to ∞	0	<i>Write Y</i>
G. Association	1 to ∞	Graphic	Choice	1 to ∞	1 to ∞	<i>Match Y with X 1-to-1</i>

Table 6.2 – Characterisation of the exercises

6.2.2 Mapping Task Types onto Gameplay Categories

After specifying the pivot, the remaining questions are: How to map 1) task types onto exercises and 2) gameplay categories onto exercises? The main idea consists of using the parameters characterising each concept (i.e., task types, gameplay categories, and exercises) to match them up.

Task Types to Exercises

Like exercises, task types can be characterised by several parameters: the *number of facts* targeted by the task (i.e., the number of questions about facts asked simultaneously, e.g., an identification task may simultaneously present several questions about different facts), the *types of statements* allowed for such a task, the *response modalities*, the *number of desired responses*, and the *number of propositions* presented (i.e., when the response modality for a concrete task of this type is “Choice”). As an example, **Identification** task type is defined as follows: 1 to ∞ facts can be targeted, only classic statements are allowed, both response modalities can be used (i.e., input and choice), the number of desired answers is equal to the number of facts targeted, and at least 2 propositions must be presented when the modality is “Choice” (i.e., true/false).

Assigning values to parameters is not an easy task. Let’s take *T1*, a task consisting of completing a fact having two missing elements, such as $? \times ? = 12$ (i.e., *number of facts* = 1 and *number of expected answers* = 2). Initially, it would seem possible to perform *T1* using the *Input* response modality. However, presenting a statement in the context of declarative knowledge, such as “ $? \times ? = 12$ ”, does not give enough information about the fact to work with, i.e., is the answer expected 3×4 or 6×2 . As another example, for a *T1*-like task, depending on how choices are displayed, it is possible to have to choose

one or two answers. If the set of proposals represents numbers, such as [3, 5, 7, 4], two answers must be chosen. However, if each proposal represents a multiplication (without the result), such as [3 × 4; 4 × 5; 6 × 3], a single answer is required. Table 6.3 presents the task types characterisation. Thus, except for the *interactions* parameter, task types and exercises are characterised by the same parameters.

	Number of Facts	Statement Types	Response Modalities	Number of Answers	Number of Choices
Completion	1	Classic Graphic Fill-in	Input	1	0
			Choice		2 to ∞
	2 to ∞		Input	∞	0
			Choice	$\geq Nb\ Facts$	2 to ∞
Order	2 to ∞	Classic Fill-in	Choice	<i>Nb Facts</i>	<i>Nb Facts</i>
Identification	1 to ∞	Classic	Input	<i>Nb Facts</i>	0
			Choice		2 to ∞
Membership Identification	1 to ∞	Classic	Input	2 to ∞	0
		Graphic	Choice		2 to ∞

Table 6.3 – Characterisation of the task types

Consequently, the mapping between task types and exercises consists of comparing the values of their common parameters. For example, **Identification** is mapped onto *Short answer* because of the specification of *Short answer*, i.e., {number of facts = 1; type of statement = classic; modality = input; number of desired answers = 1}, is a possible configuration of a concrete task of the type **Identification** (i.e., the parameter values are included into those of the type **Identification**). This gives questions such as “Did World War II happened between 1914-1918?” and “Is 2 × 5 equal to 12?”. Figure 6.3 illustrates this example.

	Number of facts questioned	Type of statement	Response modality	Number of expected answers	Number of choices	Interactions
Identification	1 to ∞	Classic	Input Choice	<i>Nb of facts questioned</i>	0 2 to ∞	
Short Answer	1	Classic	Input	1	0	Write 1
Identification	1 to ∞	Classic	Input Choice	<i>Nb of facts questioned</i>	0 2 to ∞	Write 1

} Comparison of values by pair

Figure 6.3 – Mapping between task types and exercises illustration

Additionally, **Completion** is mapped to *Fill-in-the-blanks choices* exercise specified as {number of facts = 1; type of statement = fill-in; modality = choice; number of desired answers = [1 – ∞]; number of choices = [2 – ∞]}. This gives questions such as “_ times 5 equals 15”.

Gameplay Categories to Exercises

Each category represents similar gameplays in terms of the actions to be performed, such as opening the right chest, choosing the right pot, crossing the right bridge, which belong to the SELECT category. Consequently, the common parameters of these gameplays (e.g., number of facts queried, number of possible answers) are those of the category itself.

After analysis, these categories have been characterised using the following parameters: the *interactions*, the *response modality* (i.e., input or choice), the *statement type* (i.e., format of the question asked), the *number of facts* targeted, the *number of answers* desired, and the *number of propositions* presented (i.e., if the response modality is “Choice”). These parameters are similar to those used for the exercises, and represent a minimal and relevant set of parameters to discriminate the different categories and gameplays. As an example, the SELECT category is characterised as follows: 1 to many facts can be targeted, both classic and fill-in statement types are allowed, choice is the only possible response modality, 1 to many answers can be desired, and two interactions (i.e., *Select Y from X*, and *Y (Select 1 from X₁ to X_Y)*) are possible.

	Number of facts questioned	Type of statement	Response modality	Number of expected answers	Number of choices	Interactions
MOVE	1 to ∞	Classic Graphic To Fill-in	Choice	1 to ∞	2 to ∞	Select 1 from X Point 1/ Locate 1 Match 1 with 1 Match Y with X Point Y/ Locate Y Select Y from X Y (Select 1 from X ₁ to X _Y)
↓						
	Number of facts questioned	Type of statement	Response modality	Number of expected answers	Number of choices	Interactions
MOVE (S)	1	Classic Graphic To Fill-in	Choice	1	2 to ∞	Select 1 from X Point 1/ Locate 1 Match 1 with 1
MOVE (M)	1 to ∞	Classic Graphic To Fill-in	Choice	2 to ∞	2 to ∞	Match Y with X Point Y/ Locate Y Select Y from X Y (Select 1 from X ₁ to X _Y)

Figure 6.4 – Division of gameplay categories illustration ((S) = Single, (M) = Multiple)

However, during the characterisation phase (i.e., association of values to parameters), it became apparent that the possible interactions and the statement type changed depending on whether one or more responses were desired. Therefore, in order to simplify the mappings, each category allowing one or more possible responses have been divided into two sub-categories: single (i.e., only one possible response) and multiple (i.e., from two to several possible responses). Figure 6.4 illustrates the division of the MOVE cate-

gory into MOVE(S) and MOVE(M). As a result, our five categories have been cut down into nine categories. Table 6.4 presents the characterisation of these gameplay categories. Afterwards, the mappings consisted of directly comparing the values of the parameters.

	Number of Facts	Statement Types	Response Modality	Number Answers	Number of Choices	Interactions
SELECT (S)	1	Classic Graphic Fill-in	Choice	1	2 to ∞	<i>Select 1 from X</i> <i>Point 1 or Locate 1</i>
SELECT (M)	1 to ∞	Classic Graphic Fill-in	Choice	2 to ∞	2 to ∞	<i>Select Y from X</i> <i>Point Y or Locate Y</i> <i>Y (Select 1 from X_1 to X_Y)</i>
MOVE (S)	1	Classic Graphic Fill-in	Choice	1	2 to ∞	<i>Select 1 from X</i> <i>Point 1 or Locate 1</i> <i>Match 1 with 1</i>
MOVE (M)	1 to ∞	Classic Graphic Fill-in	Choice	2 to ∞	2 to ∞	<i>Match Y with X</i> <i>Point Y or Locate Y</i> <i>Select Y from X</i> <i>Y (Select 1 from X_1 to X_Y)</i> <i>Order X</i>
ORIENT (S)	1	Classic Fill-in	Choice	1	2 to ∞	<i>Select 1 from X</i>
ORIENT (M)	1 to ∞	Classic Fill-in	Choice	2 to ∞	2 to ∞	<i>Y (Select 1 from X_1 to X_Y)</i> <i>Y (Select 1 from X)</i>
POSITION (S)	1	Fill-in	Input	1	0	Write 1
		Classic Graphic	Choice		2 to ∞	<i>Select 1 from X</i> <i>Point 1 or Locate 1</i>
POSITION (M)	1	Graphic	Input	2 to ∞	0	Write Y
DIRECT RESP.	1	Classic	Input	1	0	Write 1

Table 6.4 – Characterisation of the gameplay categories ((S) = Single, (M) = Multiple)

Task Types to Gameplay Categories

On this basis, all the necessary information has been gathered to answer our main question: Which type of task is suitable for which gameplay category? And under which conditions? Accordingly, as a last step, the task types and categories have been compared according to their parameter values (i.e., comparing Table 6.3 with Table 6.4). Figure 6.5 illustrates this mapping through an example. Throughout this process, it was observed that four parameters represented the conditions of the mappings according to their values: the type of statement, the number of facts targeted, the number of expected responses and the response modality. As a result, the relations obtained are 6-tuplets composed as follows: (\langle task type \rangle , [\langle statement type1 \rangle , \langle statement type2 \rangle , ...], \langle number of facts \rangle , \langle number of expected answers \rangle , \langle response modality \rangle , [\langle category1 \rangle , \langle category2 \rangle , ...]).

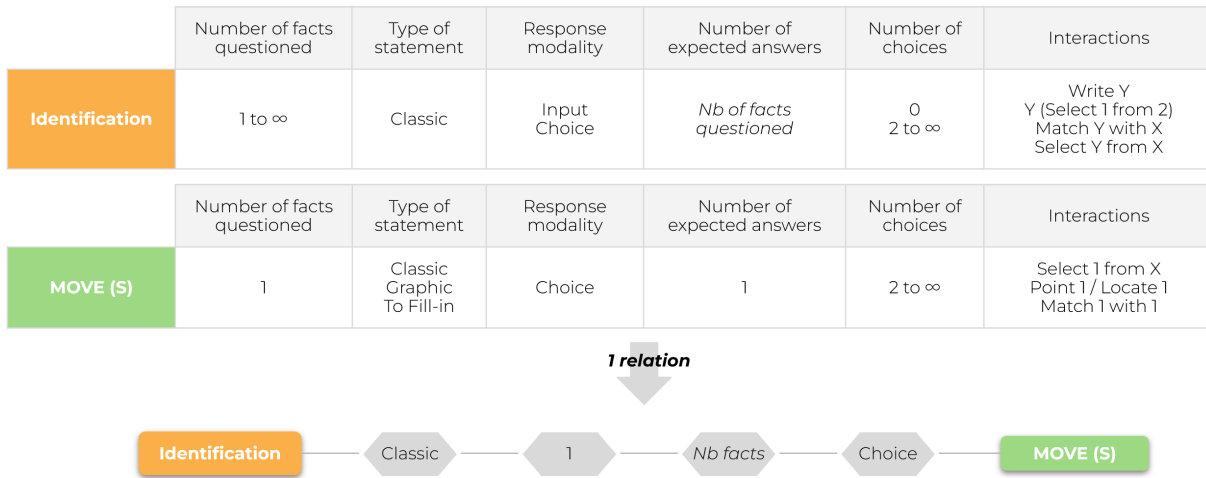


Figure 6.5 – Mapping between task types and gameplay categories illustration

As a conclusion, this section presented the process followed to map task types for declarative knowledge training onto gameplay categories for the **Roguelite** game genre. Instead of only displaying the relations identified, our intention has been to propose an approach that allows us to reproduce our approach and extend the results to other types of tasks or categories of gameplays that may be identified in other contexts. In the following section, the methods and relations obtained are presented.

6.3 A Systematic Mapping Approach

The previously presented work resulted in two contributions: 1) an approach for mapping designers’ own task types to their own game categories, and 2) mappings between our task types and our gameplay categories.

6.3.1 Proposed Mapping Approach

The proposed mapping approach is a two to five-steps approach, illustrated in Figure 6.6. It is composed of two initial steps:

1. abstraction of the concrete tasks using the types of tasks presented (e.g., a task “associate the right date with the historical event” becomes complete a fact with a missing element) or by creating new task types;
2. association of the gameplay to one of the categories presented or to a new gameplay category.

At this point, there are four possible states: new task types and categories have been created, only new task types have been created, only new game categories have been created, or nothing has been created. According to the state, the instructions below must be followed:

1. If new task types and new gameplay categories were created:
 - (a) Characterise the task types using the six parameters defined above (i.e., number of facts, types of statements, response modalities, number of desired responses, number of propositions, and interactions). In a sub-step, map task types and quiz exercises (see Table 6.2) to define the values of the *interactions* parameter.
 - (b) Characterise the gameplay categories using the same parameters.
 - (c) Finally, compare both tables (i.e., characterisation) through their values. As a reminder, the values of the *Statement Type*, *Number of Facts*, *Number of Answers*, and *Response Modality* parameters are possible conditions of the relations.
2. If only new task types were created, then realise step 1a and step 1c.
3. If only new gameplay categories were created, then realise steps 1b and 1c.
4. If no new elements have been created, the work is already done, see Figure 6.7.

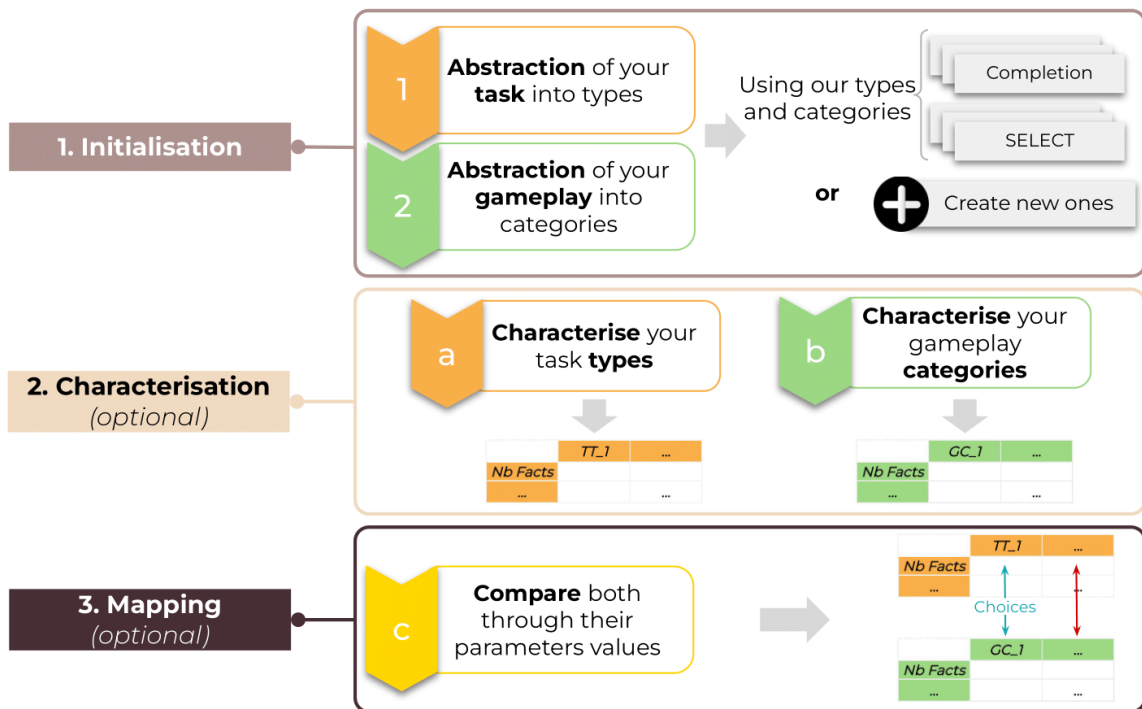


Figure 6.6 – Proposed Mapping Approach

Let's take as example a task type **T1** characterised as {number of facts = 1; type of statement = classic; modality = input or choice; number of desired answers = 1}, and a gameplay category **C1** = {number of facts = [1 – ∞]; type of statement = classic or fill-in; modality = choice; number of desired answers = [1 – ∞]}. In this case, only one relationship would result: (**T1**, *Classic*, 1, 1, *Choice*, **C1**).

6.3.2 Relations Between Task Types and Gameplay Categories

As a result, several conditional relations have been defined between our task types and gameplay categories. Figure 6.7 presents these relations. For example, the task type **Order** has a unique relationship: (**Order**, [Classic, Fill-in], [2 – ∞], Nb Facts, Choice, [MOVE (M)]). Whereas, the task type **Identification** has four relationships, including: (**Identification**, Classic, 1, Nb Facts, Input, [POSITION (S), DIRECT RESPONSE]) and (**Identification**, Classic, [2 – ∞], Nb Facts, Choice, [SELECT (M), MOVE (M), ORIENT (M)])

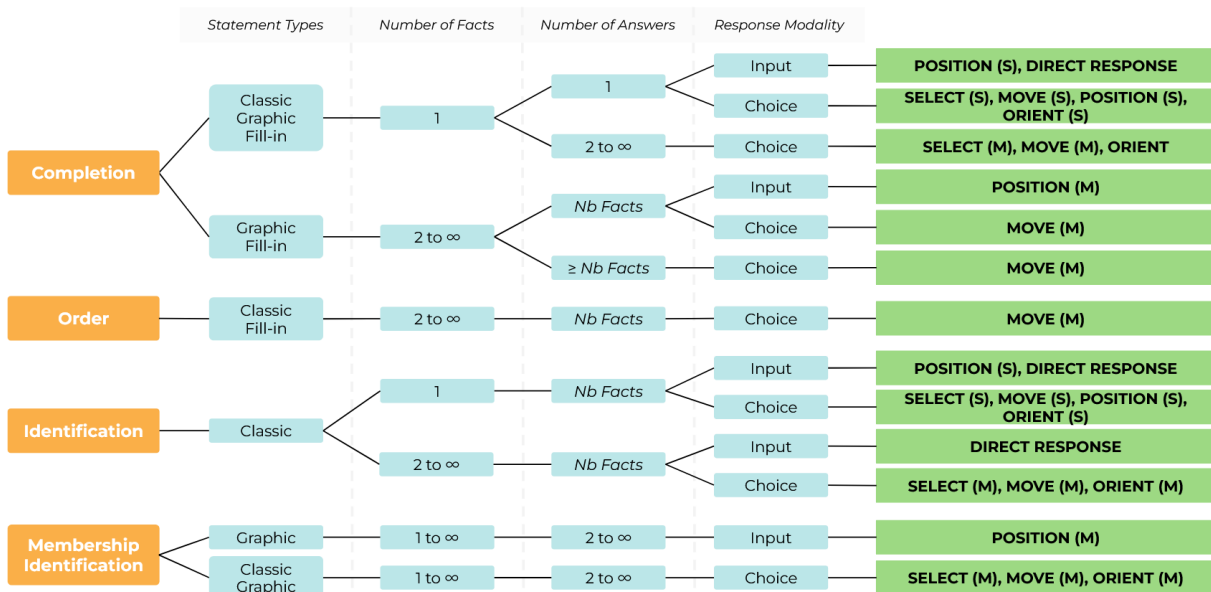


Figure 6.7 – Conditional relations between task types and gameplay categories

6.3.3 Evaluation of the Relations

In order to gather feedback on the gameplay mock-ups (i.e., identify relevant gameplays and game elements), members of the user group, from the AdapTABLES project, have been invited to participate in a survey outlining possible gameplays for each type of mathematical training tasks. The survey has also been an opportunity to validate some mappings, i.e., the relations for which gameplay categories have a mock-up compatible with a task. Appendix B presents the original survey questions.

Since the survey has been conducted in the context of multiplication tables training, none of the relations related to the completion of several facts, the ordering of facts, or having a condition with a *Graphic* type of statement have been evaluated, as the mathematics training tasks do not cover them. Nevertheless, for all other relations, gameplay mock-ups had been defined for each category and task type.

In the survey, the experts were shown an image of a gameplay for a specific training

task. A description of the gameplay specifying its category and how it functions has been provided to give an understanding of how the player interacts with the game elements to answer the question. Finally, the experts had to assess the relevance of the gameplay to the task by answering a question “Do you find this gameplay relevant? yes or no”, in addition, a comment box allowed them to detail their answers. Let’s take the example of a gameplay which consists of selecting the right jar among several having proposals (i.e., **SELECT** with **Choice**) to answer a textual question of the type “ $3 \times ? = 15$ ” (i.e., *Completion 1* of the generic type **Completion**). If this gameplay is validated by the survey, then so is the relationship (**Completion**, *Classic*, 1, 1, **Choice**, **SELECT**).

According to the results, the **mappings seemed relevant**. Negative comments had been about didactic issues or a lack of precision. For example, gameplays that asked players to place objects on the correct answer (i.e., category **MOVE**) have been rejected because the selected answer remained hidden by the object, which can have an impact on learners’ thinking. This is a cognitive issue, unrelated to the game mechanism, which can be corrected by displaying the value above the object or by displaying the chosen value within the question, in the right place and with a different colour. Figure 6.8 illustrates both solutions: the statue pushed on the left tile hides the associated ‘5’ value, but 1) the value appears on top of the statue, 2) the value appears now in purple inside the room’s statement.

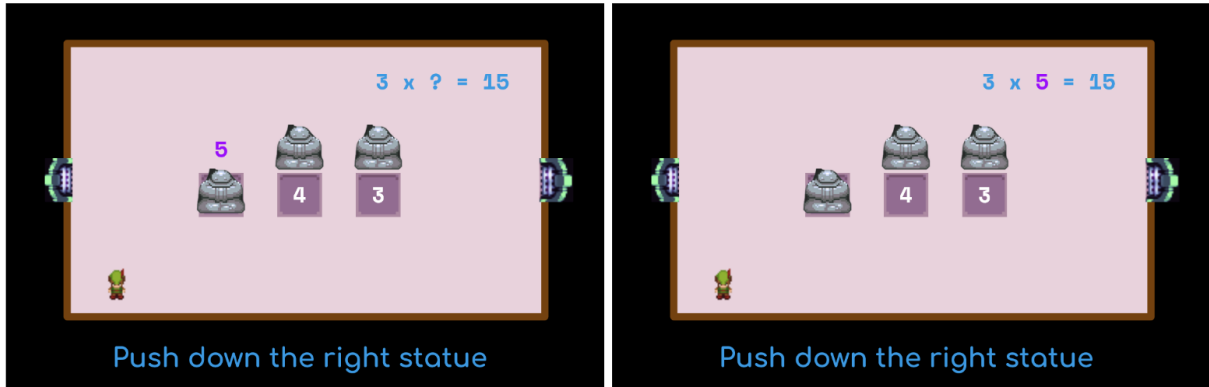


Figure 6.8 – Examples of possible solutions

Another main issue had been that the gameplay mock-ups for the **ORIENT** category relied (at the time) on an object *lantern* where the avatar had to orient the light towards the answer. This gameplay received mixed reviews because of the lack of cognitive meaning of the object (i.e., light is emitted in every direction). In order to reach a satisfactory set of gameplays, within the **AdapTABLES** project we held a focus group in which disagreements about the gameplays had been discussed and solutions to the problems observed have been proposed (e.g., using statues rather than lanterns).

6.4 Synthesis

In a previous chapter, a definition of declarative knowledge game training activities (i.e., dungeons) and descriptions of the elements that constitute these activities (i.e., training task-oriented gameplays) have been provided. In this chapter, the challenge related to the alignment of educational and game elements has been addressed, by proposing a systematic method for mapping training task types with gameplay categories using exercises from numerical quiz formats as the pivot. This work has enabled the definition of machine-readable relations between our task types and gameplay categories. These relations are necessary to generate coherent game activities, since some gameplays are not compatible with some tasks. For example, a gameplay consisting of orienting statues is irrelevant for a task consisting of chronologically ordering events, whereas a gameplay consisting of opening chests is consistent with a task consisting of completing a fact having one missing element.

However, our proposed approach relies on the use of specific parameters, which characterise the different concepts (e.g., type of statement, number of desired answers), to map the concepts. These parameters are subjective in that they represent the minimum set of parameters necessary to represent each concept in our opinion. Therefore, these parameters can be debated from different viewpoints. Moreover, not all the relations have been evaluated, but only the ones required for multiplication tables training.

At this point, all the elements, and their relations, required for generation are defined. Therefore, the next step consists of looking into the modelling of these elements in order to drive the generation process. The following chapter presents the conceptual aspects of our design and implementation framework, describing the various interconnected models involved in the generation.

CONCEPTUAL DESIGN APPROACH

Contents

7.1	Conceptual Models for Activity Generation	94
7.1.1	Domain Model: Training and Knowledge	95
7.1.2	Game Model	97
7.1.3	Activity Model	99
7.1.4	Learner-Player Model	100
7.1.5	Relation Model	102
7.1.6	Synthesis & Discussion	103
7.2	Mapping Questioned Facts with Game Elements	104
7.2.1	Generic Modelling of Questions about Facts	105
7.2.2	Modelling Gameplays Descriptions	107
7.2.3	Generic Generation of Varied Task-oriented Gameplays	108
7.3	Synthesis	111

The automatic creation of varied and adapted activities requires structuring all the previously defined information to allow its use by the activity generation algorithm. Therefore, to answer the third research question, i.e., *How to structure the required data and their relations in order to drive the generation of coherent activities?*, our proposal consists of a set of conceptual models required to generate *Roguelite*-oriented game activities for declarative knowledge training (Lemoine and Laforcade 2023a; b).

Based on an observation related to structured data sources used in research to deal with generation and adaptation (see Chapter 3), it is necessary to provide various models providing information about:

- the game (i.e., description of the elements available and the progression of the game);
- the structure of the content to be produced (i.e., elements that constitute the content);
- the didactic domain (i.e., knowledge, how to work on it);
- the learner (i.e., knowledge, skills, progress/results);
- the player (e.g., preferences or profile, progress).

Additionally, given the objective of genericness for the framework, the conceptual models proposed have been defined to be generic, i.e., to consider educational elements at a level of abstraction that allows an independence of the framework of any specific didactic domain. As a result, some conceptual models include *extension points*, i.e., portions of the models that must be extended according to the targeted didactic domain in order to specify domain-specific information such as the raw facts or the concrete tasks of the didactic domain and their parameters.

This chapter aims at introducing each of the conceptual models involved in the generation process, how they are constructed and their relationships, as well as a modelling approach (i.e., included in the conceptual models) for the domain-independent mapping of questions about facts onto gameplays, in order to generate varied gameplays at an implementation level. First, we provide a detailed description of each conceptual model one by one. Next, we present the need for a generic way of modelling questions about facts. Finally, we introduce our proposal for generating varied training task oriented gameplays for declarative knowledge based on a generic modelling of questioned fact.

7.1 Conceptual Models for Activity Generation

In this section six interconnected conceptual models involved in the generation process are presented, namely: the knowledge, the training, the learner-player, the relationships, the game, and the activity models.

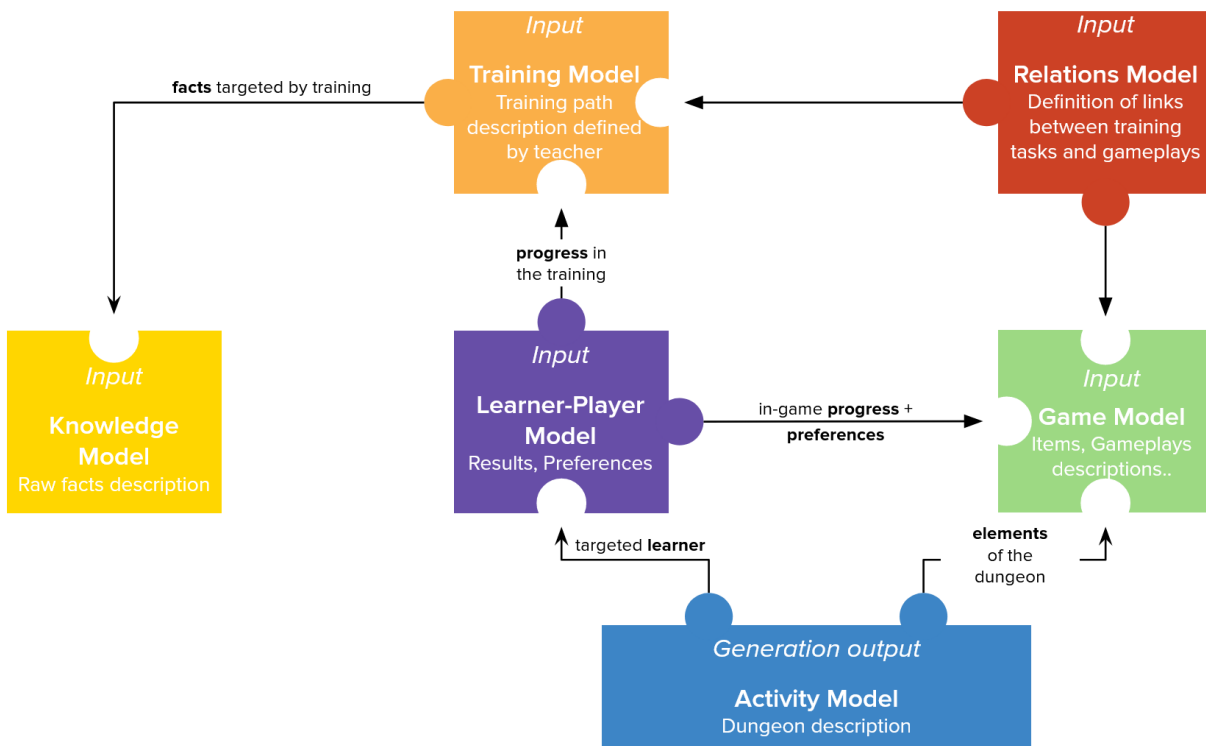


Figure 7.1 – Interconnected conceptual models involved in activity generation

Since the aim is to generate activities, these **conceptual models only include the required elements for generation**. In the following section, the domain model will be presented as a composite of the knowledge and training models. Figure 7.1 provides an overview of these conceptual models and their relations. The Activity Model to be generated targets a learner-player and is composed of game elements, some of which

are specifically dedicated to the display of training content. The Learner-Player Model presents the progress in the training of the learner, and the preferences and progress in the game of the player. Therefore, this model is related to the Game Model and the Training Model. The Training Model itself is linked to the knowledge it addresses¹. Finally, the Relation Model links the training and game elements by specifying relations².

7.1.1 Domain Model: Training and Knowledge

The first necessary conceptual model to drive generation is the domain model, which structures the knowledge and training path. Figure 7.2 presents the domain conceptual model, yellow concepts are related to the knowledge (i.e., knowledge conceptual model) and orange concepts are related to the training (i.e., training conceptual model).

As previously described, training paths (i.e., *TrainingPath*) are composed of objectives (i.e., *Objective*), which in turn are composed of levels (i.e., *Level*), and levels are themselves composed of tasks (i.e., *Task*). An objective refers to the set of knowledge (i.e., *SetOfFacts*) containing the facts it intends to work on, such as the multiplications of a table, a list of historical dates, or descriptions of the planets of the solar system. A set of facts can be composed of a visualisation (i.e., *Visualisation*) that is common to all the facts described by the set, i.e., a map (e.g., a set of facts describing the regions of France can be associated with a map of France, in which case each fact is assigned a position on the map).

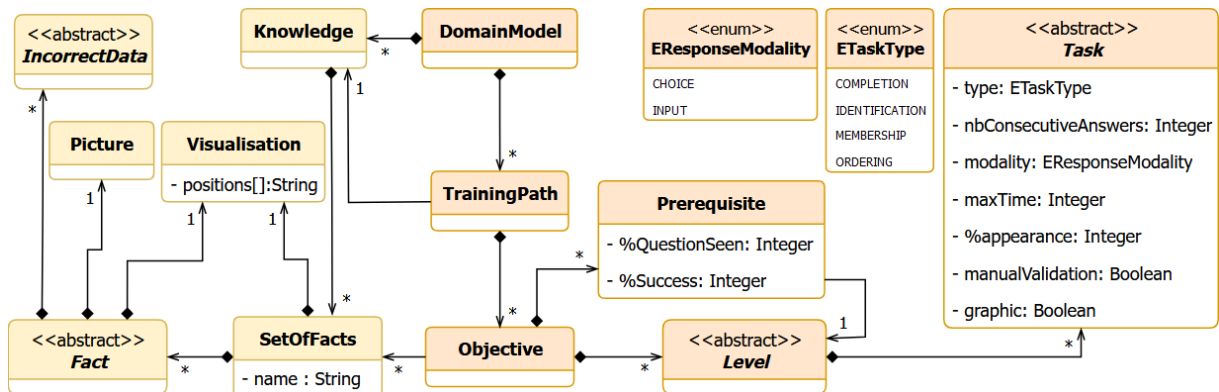


Figure 7.2 – Conceptual domain model describing **knowledge** and **training path**

Facts within this conceptual model (i.e., *Fact*) represent raw knowledge. Facts can be textual (e.g., $3 \times 5 = 15$) or graphic (i.e., a fact associated with one or more positions on a visualisation, e.g., “Position of Paris on a map of France”) and can have a visual representation, i.e., *Picture* (e.g., the historical date representing the fall of the Berlin Wall can be associated with a picture of the population destroying it). From a pedagogical strategy standpoint, in some cases, using incorrect answers predefined by the teacher,

1. In the literature, both these models are often known as the Domain Model (see Chapter 3).
2. Relations are often directly implemented in the code, and rarely made explicit through a dedicated model. Our choice has been to make it explicit.

rather than randomly generated incorrect choices can be favourable, e.g., in the case of multiplication using the sum of the elements as an incorrect proposal. Accordingly, facts can be associated to wrong answers (i.e., *IncorrectData*) allowing the specification of possible incorrect proposals. Since this knowledge differs according to didactic domains, the concept of fact and wrong data are abstract (i.e., they are therefore extension points).

Furthermore, an objective can be conditioned by prerequisite relationships to levels of other objectives. More precisely, it means that a learner can only start an objective once the prerequisite is satisfied, i.e., the percentage of questions encountered by the learner and the percentage of the learner's success to the questions satisfy the prerequisite.

Like previously mentioned, some parameters can be linked to the construction of facts when they have several forms (e.g., the position of the equal for multiplication tables, the position of the multiplicand $1 \times 1, 1 \times 2 \dots$ or $1 \times 1, 2 \times 1 \dots$). In this case, all the tasks of a level are parameterised the same way. Consequently, these parameters are directly associated to the levels. Since these parameters are specific to the didactic domain, the concept of level is abstract (i.e., it is an extension point).

Finally, a task is defined by its type (i.e., *type* corresponding to one of our four generic training task types) and is composed of the three parameters previously mentioned: the response modality (i.e., *modality* either choice or input), the maximum response time allowed (i.e., *maxTime*), and the number of consecutive successful answers to each question expected (i.e., *nbConsecutiveAnswers* which is the success validation criteria of facts).

In order to enable the generation process, some parameters must be added. In particular, for the algorithm to be in the capacity of computing the number of times a task must be present in the activity, a parameter specifying the percentage of appearance (i.e., *%appearance*) of the task has been added. This parameter allows teachers to prioritise one task over another depending on their training strategy. For example, for learners having difficulty with multiplying, a teacher might favour a *Completion 1* task over a *Completion 2* task, and vice versa for learners with an aptitude for multiplication. Note that the sum of the percentages of appearance of all the tasks at the same level must equal 100. Another parameter has been added to allow selection of the gameplay category according to the type of statement of the relations (defined in Chapter 6): a boolean indicating if the task is graphic (i.e., *graphic* meaning that the task is based on questions with visualisations). Moreover, another parameter is a boolean describing if the validation of answers is automatic or realised manually by learners (i.e., *manualValidation*). This parameter is necessary to select a gameplay having compatible elements, i.e., a manual validation requires a button or a switch to validate answers, this element must be specified in gameplays to be generated. Furthermore, this parameter is also necessary from a pedagogical standpoint, as manual validation allows for correction and self-reflection on the given answer, but it costs time in a game context.

Finally, although certain parameters are common to all tasks, others are specific to the domain and related to a specific type of task (e.g., the type of element researched for in a completion task). Accordingly, the concept of task (i.e., *Task*) is abstract (i.e., another extension point).

7.1.2 Game Model

The second conceptual model required is the game model, which describes the structure of the game elements available for activity generation. Figure 7.3 presents the game conceptual model.

As previously described, the training is carried out through task oriented gameplays (i.e., *Gameplay*). Gameplay consists of the set of game elements with which players interact or which provide them with in-game information. One of our design choices has been to have two types of gameplays: question gameplay (i.e., *QuestionGP*), and trap gameplays (i.e., *NoQuestionGP*) only describing situations involving traps or enemies to avoid in order to preserve the essence of *Roguelites*.

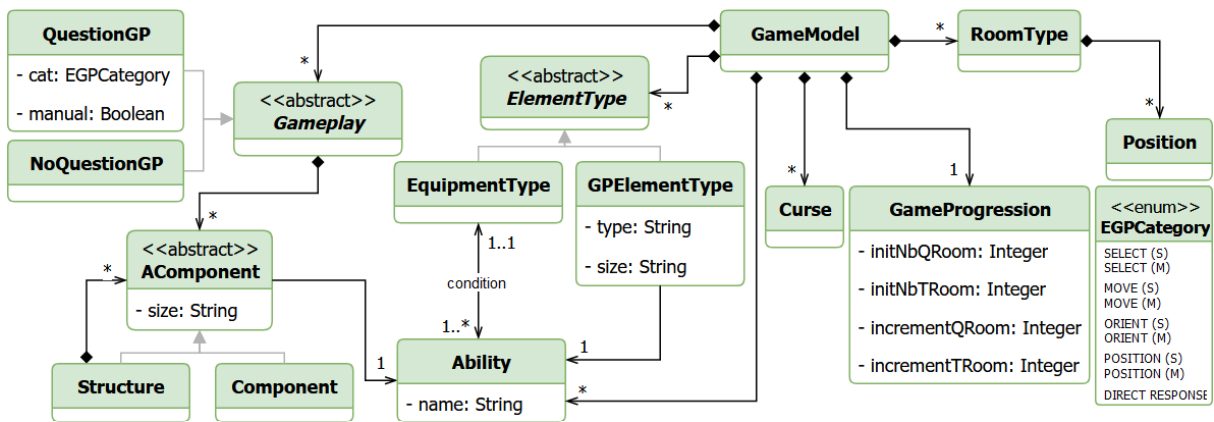


Figure 7.3 – Conceptual game model

Each question gameplay is associated to a category (i.e., *cat*) for declarative knowledge training through *Roguelite* (i.e., *EGPCategory*), as defined in Chapter 5, that represents different means of answering questions about facts (e.g., moving a jar on an answer, opening a chest having an answer, orienting a statue towards an answer, pushing a block on an answer). As previously mentioned, when a training task requires a manual validation, specific gameplays disposing of a manual validation system have to be used. In order to differentiate between manual versus automatic validation gameplays, question gameplays have a parameter stating whether they provide *manual* validation. Moreover, question gameplays can be specially designed for a specific task type (e.g., presence of display true/false for identification tasks). Therefore, they can have a parameter restricting their availability to a specific type of task.

Having such a variety of gameplays (i.e., different categories and the possibility of modelling different gameplays per category and different trap gameplays) enables greater variability from the algorithm in choosing game elements to construct the activity. However, while a static definition of gameplays in terms of specific game elements provides a certain level of variety, it also introduces two limitations:

1. it is time-consuming, i.e., gameplays must be described one by one according to the game elements available;

2. it is static, i.e., the addition of a new game element implies the need to specify new gameplays for that element.

Consequently, in order to increase the variability of the proposed gameplays, our proposal is to define the gameplays and element types (i.e., *ElementType*), which can be used to build instances of these gameplays, through **abilities**. Abilities define the behaviour of game elements (i.e., the way in which the player avatar can interact with the game element). As an example, a block can be pushed (PUSHABLE), a pot can be moved (MOVABLE), a bridge can be crossed (CROSSABLE), and so on. Hence, several types of element can be defined with the same ability, e.g., a cube and a jar can be moved (MOVABLE). Thanks to this modelling, it is possible to define for example a *gameplay* of MOVABLE elements, but generate playable *gameplays* (i.e. an actual description of the *gameplay* which is going to be played) with different types of MOVABLE elements (e.g., the same description of gameplay can produce instances with jars and with cubes). Gameplays are then described by components³ (i.e., *AComponent*) that rely on a specific ability rather than a specific game element. This creates gameplay variability as the ability is known, but the actual game elements will be selected by the generation algorithm (i.e., the same game element is chosen for the same ability required in a gameplay definition). Moreover, gameplays can specify an expected *size* of game elements, limiting the choice of a game element type by the algorithm to those having the correct ability and size (e.g., only small MOVABLE elements). There are two components types:

- simple components, i.e., elements not composed of other components (e.g., chest, jars or enemies);
- structure components, i.e., elements composed of other components (e.g., components describing blocks to be pushed on specific tiles: structure = [block, tile]).

Once again, to provide more variety, one of our choices has been to model different types of rooms (i.e., *RoomType*) for the dungeons, which describe their access (e.g., north, south, north-east) as well as all the possible positions for game elements. These descriptions provide some flexibility to the algorithm in selecting from a number of room type variants and in organising the rooms (i.e., positioning the game elements) differently at each generation.

As mentioned previously, the design choices for generation include a mechanic for purchasing/activating items. Basically, the idea is for items (i.e., *EquipmentType*) to be able to lock/unlock some abilities, and therefore enable players to define preferences in terms of gameplay within a game. Such a design choice ensures that players' game preferences are not purely aesthetic as the selection of gameplays, during generation, depends on the items purchased and activated.

Furthermore, to generate game activities, it is necessary to know how the game progresses so that the difficulty can be increased as players progress through the game. Like most of the **Roguelites**, progress is based on a difficulty level approach. Game levels have

3. Gameplays are described by means of components, since this is a conceptual description of the gameplays and not a description of their implementation in the activities. The concrete gameplay elements of activities are called *PositionedElement* in Figure 7.4.

an initial number of rooms with and without question (i.e., *initNbQRoom* and *initNbTRoom*). Based on the players' game level and incremental factors (i.e., *incrementQRoom* and *incrementTRoom*) these numbers are incremented:

- number question rooms = $initNbQRoom + (incrementQRoom \times \text{player's game level})$.
- number trap rooms = $initNbTRoom + (incrementTRoom \times \text{player's game level})$.

Moreover, many **Roguelites** feature a curse mechanism which appears once players have reached a specific game level. As an example, from level eight, dark level curse is unlocked, players then have to answer questions with a limited beam of light illuminating them (i.e., simulation of a torch). As a result, game levels greater than or equal to eight may be in the dark purely at random. Our decision has also been to keep this mechanism. In our case, four curses have been considered: dark level (i.e., the level is in the dark), labyrinthine (i.e., the dungeons are not linear⁴), one life (i.e., the player has only one life to finish the entire dungeon), out of time (i.e., the player must finish the level within a given time). However, it is important to note that certain curses could be problematic from a training standpoint (e.g., a dark level makes it difficult to read the questions and therefore increases the response time and the risk of failure). Consequently, adding curses is optional and should be discussed with teachers or configurable by teachers.

7.1.3 Activity Model

The third conceptual model involved in the generation process is the activity model, which describes the structure of the activities that must be produced. As defined in Chapter 5, an activity is a dungeon (i.e., *Dungeon*) which is described by the learner-player it is generated for (i.e., *lp*), the objective and level of the training path targeted (i.e., *objective/level*), the curses it includes (i.e., *curses* such as labyrinthine or dark dungeon), and it is composed of rooms (i.e., *Room*) including two specific rooms: the entry (i.e., starting point of the dungeon) and the exit (i.e., ending point of the dungeon). It is important to note that from a generator standpoint, three out of four of our chosen curses only are declared information included in the generated dungeon that have to be handled by the *game engine* (e.g., dark mode, one life, out of time). However, the labyrinthine curse has an effect on the generation algorithm, as the structure of the dungeon changes based on the presence of this curse.

Furthermore, each room is described by the type of room (i.e., *type*) and gameplay (i.e., *gameplay*) it implements, as well as its accesses to other rooms (i.e., *otherRoomsAccesses*, its neighbours). In the case of a question gameplay, the room is also described by the training task (i.e., *task*) and the facts it questions (i.e., *QuestionedFact* which are built from this task). Questioned facts represent questions about facts accompanied by a set of proposals (i.e., correct and incorrect) or a set of expected answers (i.e., depending on the

4. Linear does not mean that the rooms always have one and the same direction (e.g., always towards the north), but means that they have a single exit, different from the entrance, that can be, for example, north, east or south. The exit is chosen by the generation algorithm in a pseudo-random way to maintain the consistency with the previous rooms.

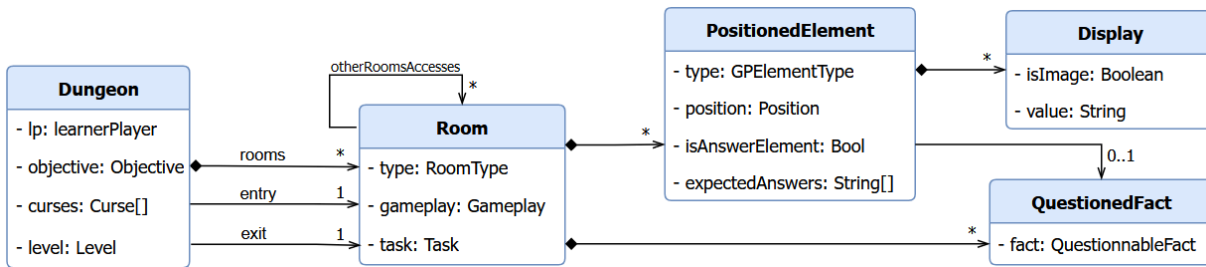


Figure 7.4 – Conceptual activity/dungeon model

response modality of the task from which they are built). Taking the example of two tasks, T1 and T2. T1 consists of choosing from a set of proposals the answer that matches the result of the multiplication for each fact. Based on the parameters of T1, the questioned facts are constructed to produce, for instance, questions such as $2 \times 6 = ?$ with a set of proposals like $\{8, 12, 14\}$. T2 consists of choosing from a set of proposals the possible results of a given table. Based on the parameters of T2, the constructed questioned facts would yield questions such as “Which are results of the table of 3?” and a set of proposals like $\{3, 5, 7, 9, 12\}$.

Moreover, according to the room type and gameplay, each room will possess different positioned game elements (i.e., *PositionedElement*). A positioned element is a game element with a *type* (i.e., *GPElementType*) which has the ability specified in the description of the selected *gameplay*, and to which a *position* of the room type has been assigned. These elements have different parameters depending on their type of gameplay (i.e., with or without a question). In particular, if they represent one part of a question about a fact (e.g., proposal, statement), the elements can have one or more display values (i.e., *Display* which can either be text values or pictures described by their identifier) and a verification parameter (i.e., *isAnswerElement*, e.g., a boolean describing whether the proposal is correct or incorrect).

7.1.4 Learner-Player Model

The fourth conceptual model required is the learner-player model, which describes the learner’s progress, the player’s progress and their preferences. In order to keep track of a learner’s progress in their training path (defined by the teacher), their results on the questioned facts for each task of an objective/level pair must be registered. This recording can be used to calculate the number of questioned facts encountered/worked on and the percentage of success for each task and level.

First, a learner-player is described by an *identifier*, its training *path* that is defined in the training model because it can be shared among several learners (i.e., depending on the teachers viewpoint), and its current game level (i.e., *currentGameLevel*) in order for the algorithm to generate a dungeon according to the correct game difficulty (i.e., if the last successful dungeon was level 8, then the next generated dungeon should be level 9). Furthermore, in order to adapt to the player, their game preferences (i.e., *GamePreference*)

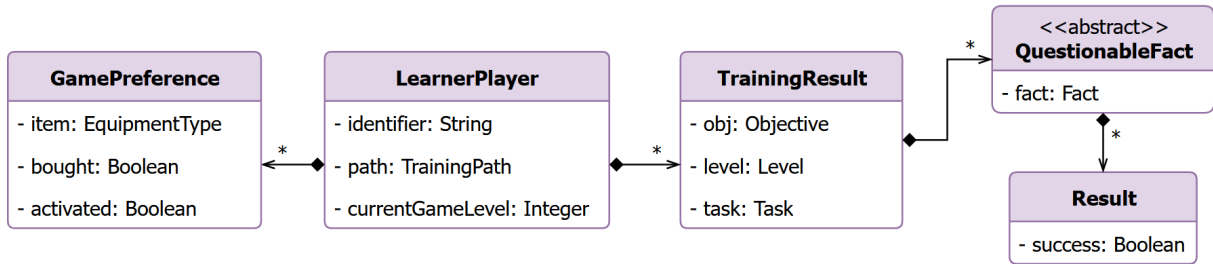


Figure 7.5 – Conceptual learner-player model

have to be recorded. Therefore, learner-player are associated to a set of preferences describing for each item if they are *bought* and *activated*. Moreover, to adapt to the learner, their training results (i.e., *TrainingResult*) for a *task* of a *level* of an *objective* of their training path must be stored.

The previous models referred to raw facts and questioned facts (i.e., questions on raw facts). Questioned facts have, in the case of the choice modality, the right and wrong proposals. However, according to our discussions with the experts, it seems more interesting from a didactic standpoint to constantly vary the incorrect proposals. Additionally, it is necessary to be able to compare the results they have obtained for a given question in order to evaluate whether a fact is considered achieved. However, if the results are recorded on the basis of the facts questioned, the comparison will be irrelevant, since the incorrect proposals of the questioned facts vary at each generation. Two questioned facts for the same raw fact will therefore be considered as distinct. Furthermore, by recording the results on the basis of the raw facts, the information about the type of question asked is lost. As a result, in order to compare learners' results on a given fact, it is necessary to have a common base that does not vary (i.e., preservation of the question format, without the elements that vary).

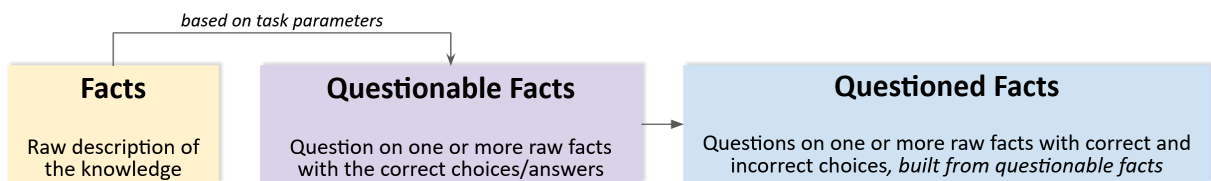


Figure 7.6 – Transformation process of raw facts into questioned facts

To this end, our proposal is a two-stage transformation process. First, questionable facts are constructed on the basis of the raw facts present in the knowledge model and in accordance with the context related triplet (objective – level – task). A questionable fact (i.e., *QuestionableFact*) represents a question about a fact without incorrect proposals. These facts are used to store learners' results (i.e., *Result*) in the learner-player model (e.g., response times, answers given). Second, questioned facts are built on the basis of questionable facts (i.e., questionable facts with incorrect proposals). Figure 7.6 illustrates this transformation process. Questionable facts are built up from domain-specific tasks.

Therefore, the concept of questionable fact is abstract, since these questions on facts have a domain-dependent form. Consequently, the generation of questionable and questioned facts is necessarily domain-dependent.

7.1.5 Relation Model

The fifth and final conceptual model required is the relations model, which describes conditional relations between task types and gameplays categories. Previously, we have proposed a systematic method for defining machine-interpretable relationships between training task types and gameplay categories, as well as relationships between our task types and categories. In order to be used by the generation algorithm to correctly select a gameplay, these relationships need to be specified.

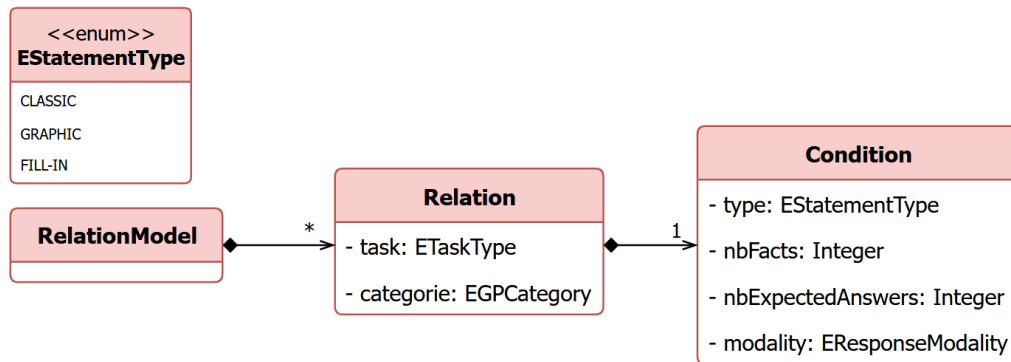


Figure 7.7 – Conceptual relation (between task types and gameplay categories) model

In the literature, the definition of explicit relationships between the educational and game dimensions is addressed at a general design level (e.g., Gosper (2011) and Lim *et al.* (2013)) rather than at an algorithmic level. As a result, the relationships are directly taken into account in the design of the game activity specific to a didactic domain (e.g., Debabi and Champagnat (2017)). However, modelling these relations at an algorithm level allows:

1. modularity, i.e., possibility of modifying the relations without modifying the source code of the generator;
2. extensibility, i.e., adding a task type (resp., gameplay) does not require adding any code, only new relations between the new task type (resp., gameplay category) and existing gameplay categories (resp., task types) to the existing “instance” of the conceptual model.

Therefore, the relation conceptual model is “fixed”, meaning that it is intended to be “instantiated” just once. It represents the structure of the relations (i.e., *Relation*) between tasks and categories from Figure 6.7, that are conditioned by four parameters: the type of statement (i.e., *type*), the number of questioned facts (i.e., *nbFacts*), the number of expected answers (i.e., *nbExpectedAnswers*), and the response modality (i.e., *modality*).

On the basis of these relations, for a given domain-dependent task, the main steps performed by the algorithm for selecting a compatible gameplay are as follows:

1. get the associated task type of the targeted task;
2. collect every relation from the relation model related to this task type;
3. restrict the collected relations to those whose associated condition is satisfied (compare the statement, number of facts, number of expected answers, and response modality values of the condition and the original task);
4. collect the gameplay categories of the remaining relations;
5. restrict the gameplays of the previously collected categories, according to their specific parameters (i.e., verify that the validation criteria are compatible and that the gameplay is not restrained to another type of task);
6. randomly select a gameplay from the remaining set.

7.1.6 Synthesis & Discussion

In this modelling, despite the use of task types and gameplays categories connected through the relation conceptual model, gameplay generation remains domain dependent because modifications of the generation algorithm are required according to the different forms of facts. Initially, it would seem that modelling questioned facts must be performed dependently of the domain, since questions on facts have different shapes (e.g., “Which are the results of table 3? {3, 5, 9, 13}”, “ $3 \times ? = 15$ ”). Moreover, gameplays also have different structures and elements based on their categories and layout (e.g., some game elements have only one choice, others several, some are simple, others are composite). Consequently, it would preliminarily seem that mapping questioned facts with game elements must be performed specifically for each task/gameplay pair. Therefore, our question is the following: How can questioned facts (i.e., from the activity conceptual model) be defined domain-independently in order to be used generically in the generation of game elements corresponding to gameplays descriptions? Figure 7.8 illustrates this question.

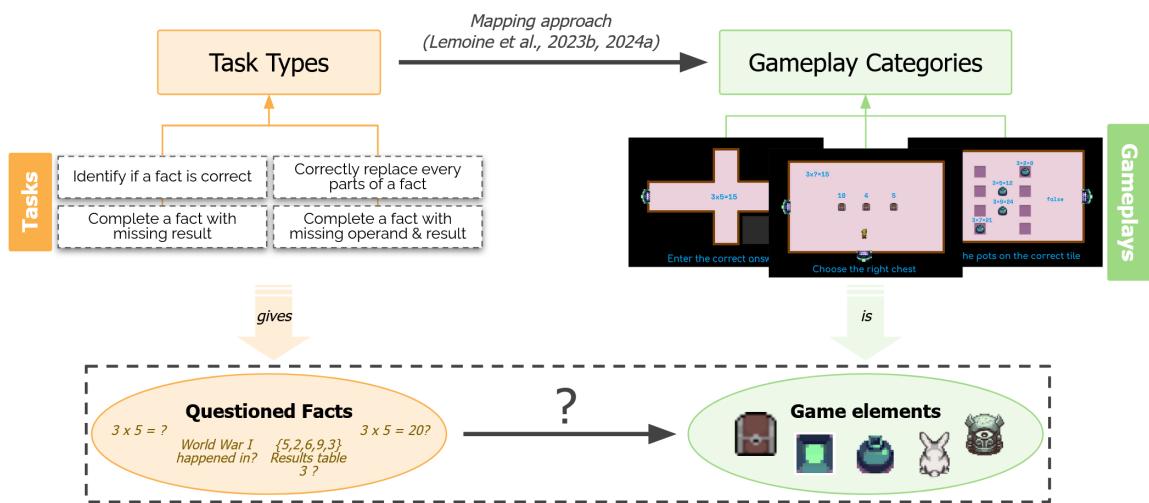


Figure 7.8 – Illustration of questioned facts to game elements mapping problem

Our proposal consists of a generic modelling of questioned facts, and an addition of parameters to the gameplay components of the game conceptual model. These parameters enable the association of the different parameters of the facts questioned with the gameplay components (e.g., a component can be specified to display the propositions of a questioned fact), in order to build elements that are positioned in the dungeon rooms. The advantage of having a generic modelling of questioned facts is that it allows using them and their parameters to instantiate concrete gameplays independently of any didactic domain (i.e., generic representation of questioned facts that can be manipulated as such). However, generating questions about facts (i.e., build the questions and their generic representation) remains domain-dependent, since the construction of questionable facts and the instantiation of questioned facts are specific to the didactic domain targeted. Therefore, the general idea is that the questioned facts would be viewed as parametrisable concepts and the questionable facts, constructed according to the domain, would be used to correctly instantiate the parameters of the questioned facts. Figure 7.9 illustrates the principle behind this idea. The purple concepts represent the domain-dependent source code that creates the questionable facts and instantiates their generic form using the ‘required interfaces’ of the generic format, blue concept, for the proposed questioned facts (i.e., by implementing the methods required according to the specific task and the form of questions it produces).

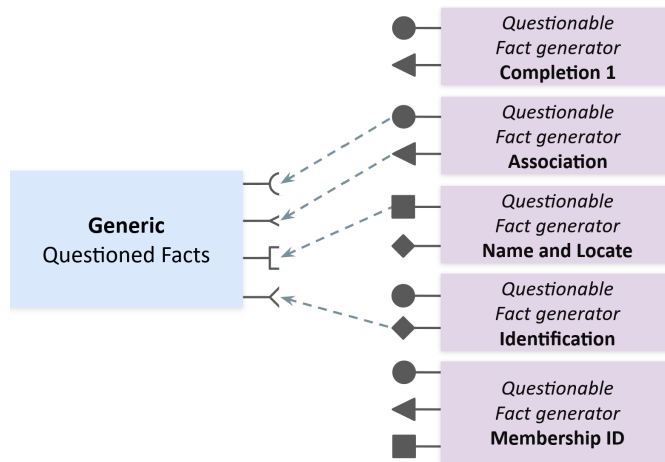


Figure 7.9 – General idea behind the concept of generic questioned facts

7.2 Mapping Questioned Facts with Game Elements

This section presents a generic way to model questions about facts, a focus on the game conceptual model to specify required parameters to drive a domain-independent generation of task-oriented gameplays, as well as a possible generic algorithm for task oriented gameplay generation. The approach has been presented in [Lemoine and Laforcade \(2023b\)](#), therefore some passages in the next sections are taken from this article.

7.2.1 Generic Modelling of Questions about Facts

In order to generically (i.e., independently of any didactic domain) generate gameplays (i.e., set of positioned elements in dungeon rooms) that question facts, questioned facts must have a generic form. Our main idea consists of modelling these questioned facts as a concept with several possible parameters. Even though, the form of question about facts varies according to the training tasks, the elements composing them can be handled in a generic way. Let's consider the four following training tasks:

- T1 involves choosing an answer, from a set of proposals, to find the result of a multiplication.
- T2 involves choosing from a set of proposals the possible results of a given multiplication table.
- T3 involves naming 10 French urban areas on a map.
- T4 involves typing the answer of the result of a multiplication table.

Based on the parameters of each task (i.e., T1, T2, T3, T4) respectively, the algorithm would produce questions about facts such as:

- T1) “ $2 \times 6 = ?$ ” with a set of proposals like 14 (incorrect), 8 (incorrect), 12 (correct).
 T2) “Which are results of table 5?” with a set of proposals such as 13 (incorrect), 25 (correct), 8 (incorrect), 15 (correct), 10 (correct).
 T3) “Name 10 urban areas” with a map of France where cities are marked using points and a set of accepted/possible solutions and their position on the map, for example.
 T4) “ $2 \times 6 = ?$ ” and would expect the solution 12.

Even though the questions are different, these questions on facts have common components. Every question is composed of a textual question. Moreover, depending on the answer modality (i.e., choice or input) they either have a set of proposals (i.e., visual or textual) or a set of expected/possible solutions (i.e., it can be composed of a single expected solution, or multiple accepted solutions). Additionally, some questions can be accompanied by a visualisation.

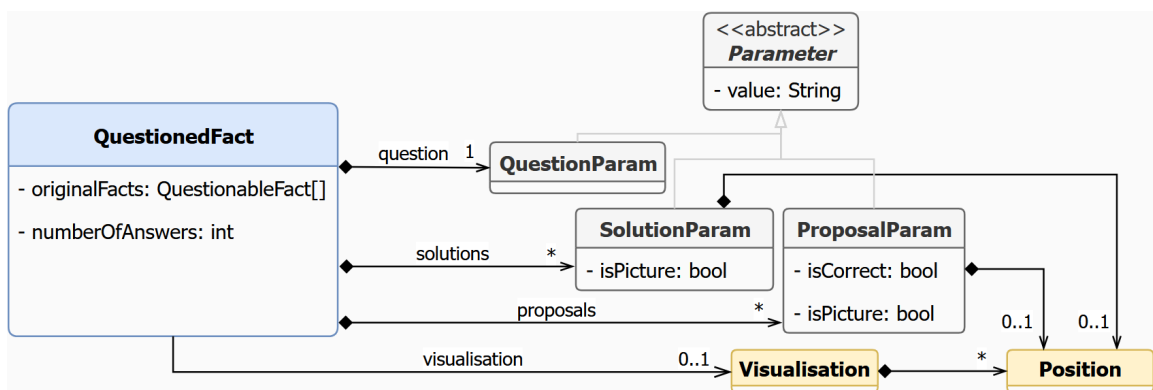


Figure 7.10 – Conceptual modelling of generic questioned facts

As depicted in Figure 7.10, questioned facts are concepts having parameters that must

be instantiated if necessary. The first parameter is the *question*, i.e., a text describing the question to be asked (i.e., *QuestionParam*). The second parameter is the *proposals*, i.e., list of possible choices (i.e., *ProposalParam*) when the response modality is choice. These proposals are described by a *value* (i.e., text describing the proposal, it can also be the identifier of a picture), a boolean expressing whether it is a picture or not (i.e., *isPicture*), a boolean expressing whether it is a correct or incorrect proposal (i.e., *isCorrect*), i.e., this is necessary to access learners' answers. Additionally, a proposal can have a position (i.e., *Position*) on a visualisation (e.g., map). The third parameter is the *solutions*, i.e., list of possible or expected answers (i.e., *SolutionParam*) when the response modality is input. Like for the proposals, these solutions are described by a value, a boolean expressing if it is a picture or not, and they can have a position on a visualisation. Moreover, a questioned fact refers to the questionable facts it is built on (i.e., *originalFacts*) and can refer to a *visualisation*. Furthermore, from an automation standpoint, it is necessary to know the number of expected answers (i.e., *numberOfAnswers*, e.g., for a question such as “ $3 \times ? = ?$ ”, two answers are expected) to declare whether a learner has entirely answered a question or not. Figure 7.11 describes our examples of questioned facts for T1, T2, T3, and T4 where only the necessary parameters have been instantiated.

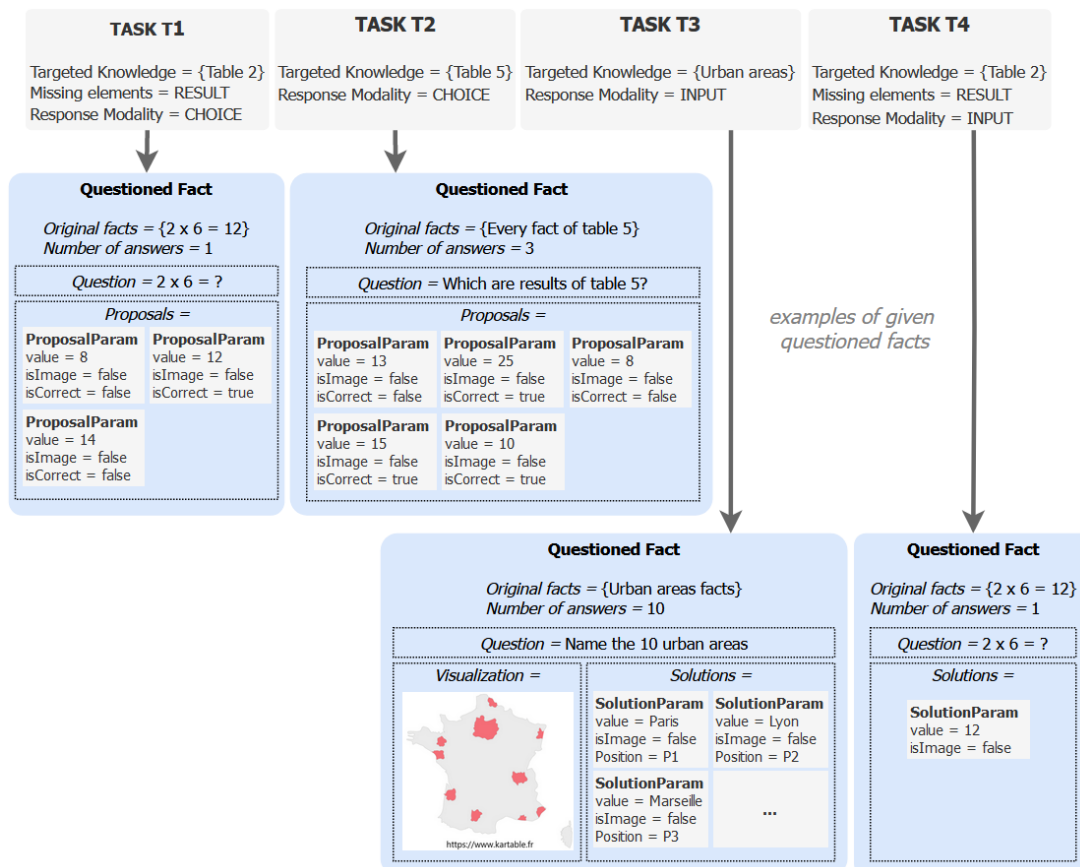


Figure 7.11 – Examples of questions about facts in generic form

7.2.2 Modelling Gameplays Descriptions

Due to the context of declarative knowledge training, gameplays are task-oriented, which means that their components have an intention, i.e., they represent parts of questioned facts such as a statement or a proposal. As shown in Figure 7.13, structures (i.e., composite components) can be instantiated for each questioned fact in a room (i.e., *isPerFact* meaning it possesses simple components displaying the statement and every proposal), for each proposal of a questioned fact (i.e., *isPerProposal* meaning it possesses simple components displaying the proposals and another game element), for a visualisation (i.e., *isPerVisualisation* meaning it possesses simple components expecting the proposals/answers and a component for displaying the visualisation), or the statement of a questioned fact (i.e., *isPerStatement* meaning it displays the statement of a fact with answer areas such as fill-in statements, it possesses simple text components and other game elements). Whereas simple components can be instantiated to represent statements (i.e., *isPerStatement*), proposals (i.e., *isPerProposal*), or input areas (i.e., *isPerInput* which allows the declaration of input elements expecting specific input answers). These components parameters are necessary to the algorithm in order to correctly instantiate the positioned elements corresponding to the gameplay, as they enable it to know which parts of questioned facts must be associated to which components.

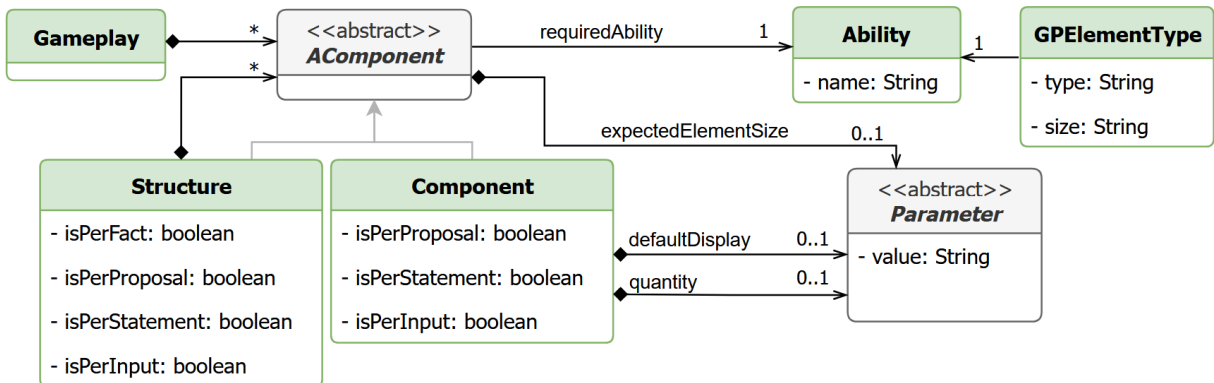


Figure 7.12 – Focus and detail of the game conceptual model

Furthermore, some simple components can describe decoration elements or answer areas (e.g., tiles where the player must place elements). These elements can have a default display (i.e., *defaultDisplay*, e.g., specific decoration text) or specify a default necessary quantity (i.e., *quantity*) that depends on the expected number of answers. As previously discussed, game elements are described according to their size. Hence, abilities can be represented by different sized elements. Therefore, gameplay components can specify an expected size in order for the generation algorithm to maintain consistency when instantiating a gameplay.

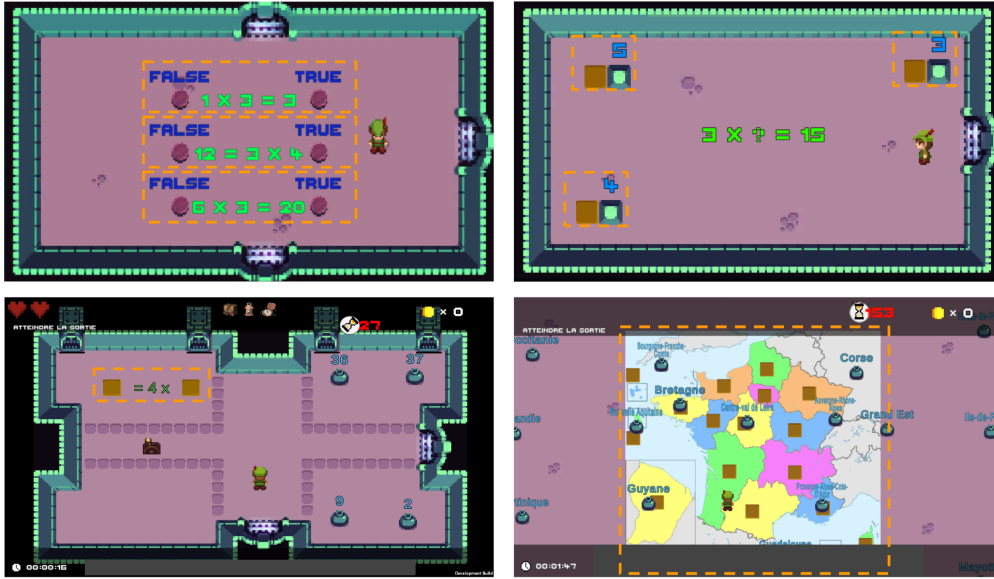


Figure 7.13 – Gameplays with structures (orange dashed borders) per fact (up-left) / per proposals (up-right) / per statement (bottom-left) / per visualisation (bottom-right)

7.2.3 Generic Generation of Varied Task-oriented Gameplays

The generation algorithm is based on the previously described modelling approach and consists in associating the values of the instantiated parameters of the generic questioned facts with the game elements according to their intention. Therefore, for a gameplay G and a questioned fact QF , the algorithm consists in scanning the components of G ⁵ and, for each component to build positioned elements according to the values of the instantiated parameters of QF and the properties of the component type. For example, for a simple component with *isPerProposal* set to true, a positioned element will be created for each element in the list of *proposals* of QF . More precisely, to generate varied task-oriented gameplays for a given task, the algorithm consists of the following steps:

1. selecting a compatible gameplay, based on the steps defined in Section 7.1.5;
2. recursively scanning each component of the gameplay (i.e., to also consider components of structures) and for each component:
 - (a) finding a game element with the right size and ability;
 - (b) selecting an available position of the room type to place the element;
 - (c) creating the corresponding positioned elements by correctly linking questioned facts parameters values to positioned elements parameter values based on components parameters values (e.g., if a MOVABLE component is for proposals, then the positioned elements with MOVABLE ability will

5. For structure components, a recursive call is made to build its own components.

- display the proposals of the questioned fact);
 (d) adding the built positioned elements to the room.

Appendix C presents the main structure of the generation algorithm.

Let's consider a first example, involving gameplay $G1$ in which the player answers by orienting an object. $G1$ is described by two simple components, one with the intention of carrying the proposals (i.e., defined as such $\{\text{isPerProposal}=\text{true}, \text{isPerStatement}=\text{false}, \text{isPerInput}=\text{false}, \text{ability}=\text{ROTABLE}\}$), another with the intention of displaying the statement of the questioned fact (i.e., defined as such $\{\text{isPerProposal}=\text{false}, \text{isPerStatement}=\text{true}, \text{isPerInput}=\text{false}, \text{ability}=\text{DISPLAYABLE}\}$). From this description, gameplays comprising a positioned element carrying the proposals of a questioned fact and an element displaying its question could be generated. Figure 7.15a presents a possible generated gameplay for $G1$ and a questioned fact $QF1$, defined as follows $\{\text{question}=\text{"}2 \times 9 = \text{"}, \text{proposals}=[20 \text{ (incorrect), } 18 \text{ (correct), } 16 \text{ (incorrect)}], \text{numberOfAnswers}=1\}$.

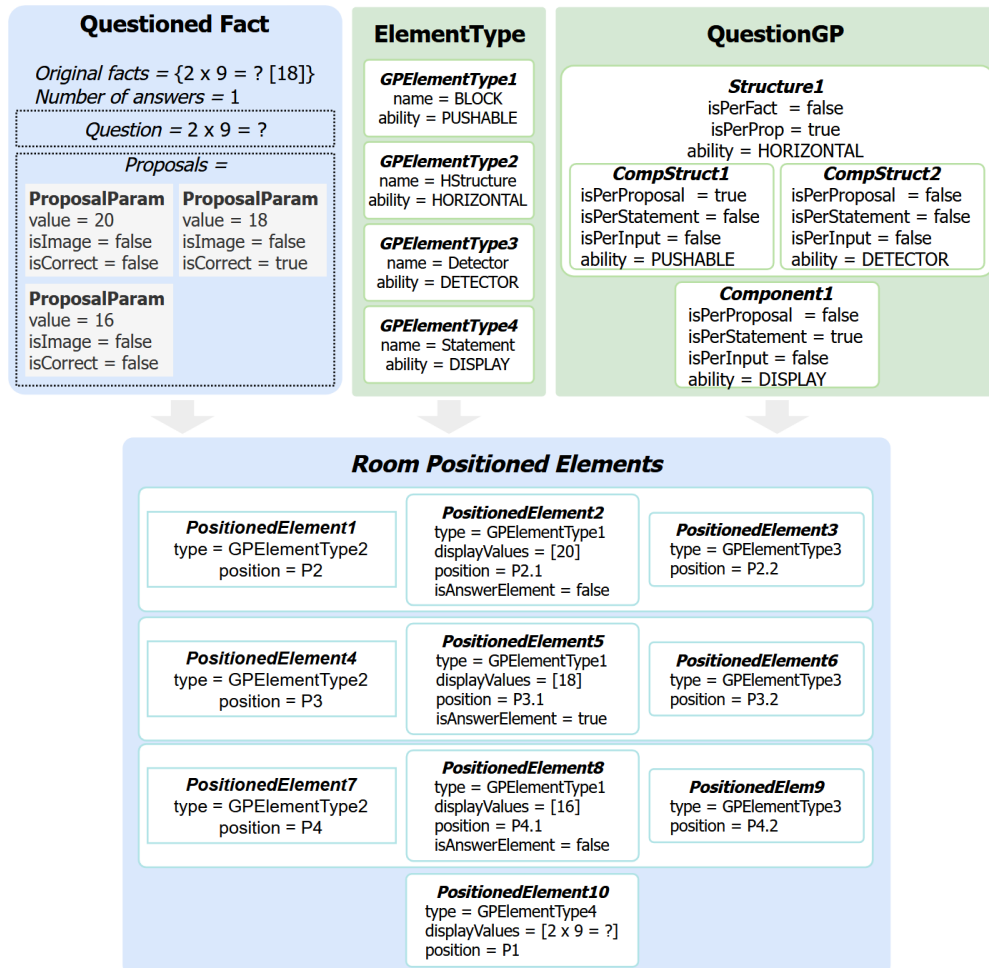


Figure 7.14 – Example of generated positioned elements from a questioned fact, game elements and a gameplay description

Let’s now consider a rather more complex example describing a gameplay $G2$ of blocks to be pushed, for the proposals of a questioned fact. $G2$ is described using a simple component for displaying the statement and a structure comprising two simple components (i.e., a PUSHABLE component for proposals and a DETECTOR component of PUSHABLE component). Based on the description of $G2$ and the questioned fact $QF1$, Figure 7.14 illustrates an example of possible instantiations of positioned elements. That specific instantiation results in the generated gameplay shown in Figure 7.15b.

As a final example, consider a gameplay $G3$ that consists of catching (i.e., touching) an object. $G3$ is described by two simple components, one having the ability CATCHABLE for the proposals of the questioned fact (i.e., defined as such $\{\text{isPerProposal}=\text{true}, \text{isPerStatement}=\text{false}, \text{isPerInput}=\text{false}, \text{ability}=\text{CATCHABLE}\}$), the other having the ability DISPLAYABLE for the question of the questioned fact (i.e., defined as such $\{\text{isPerProposal}=\text{false}, \text{isPerStatement}=\text{true}, \text{isPerInput}=\text{false}, \text{ability}=\text{DISPLAYABLE}\}$). Let’s suppose three game elements are available, a rabbit with the ability CATCHABLE, a cow with the ability CATCHABLE and a display with the ability DISPLAYABLE. The algorithm will create a game element of the display type with the value “ $2 \times 9 = ?$ ” and three elements, one for each proposal, of either the cow or rabbit type (see Figures 7.15c and 7.15d).

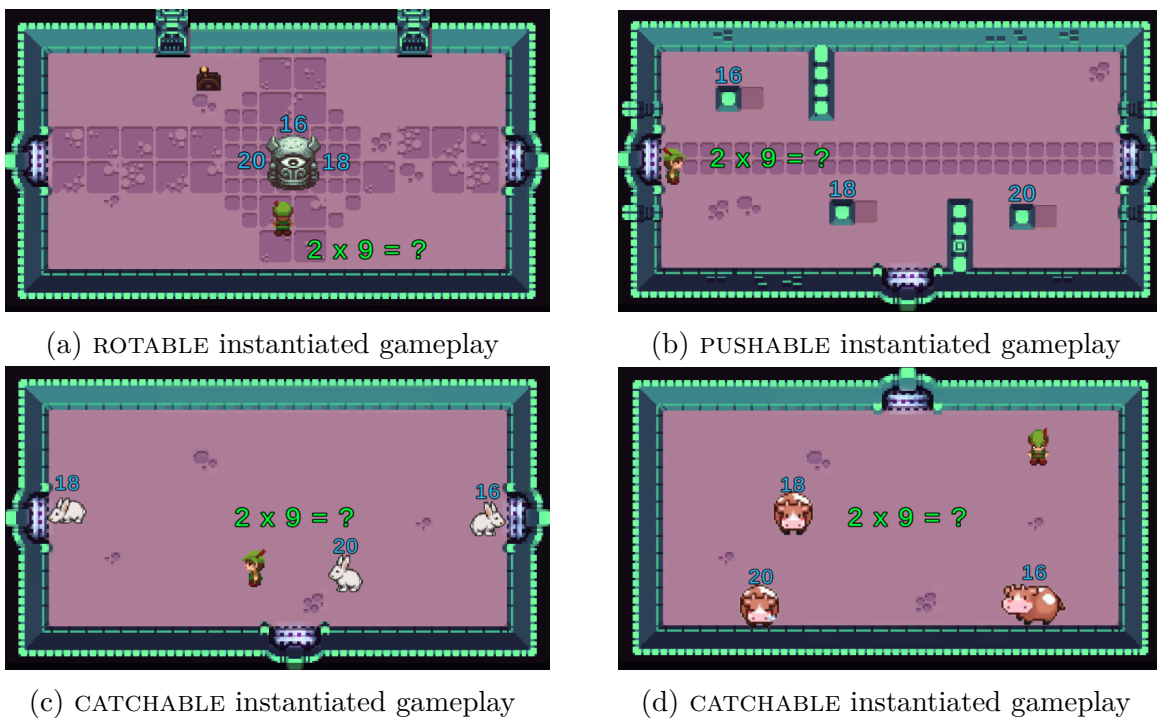


Figure 7.15 – Examples of generated gameplays, based on the same questioned facts but different gameplay descriptions

Therefore, the variety of gameplays according to the tasks depends on: the number of gameplay descriptions available, the number of gameplay categories compatible with the training task, and the number of game elements defined with the same ability.

7.3 Synthesis

In summary, this chapter has presented the conceptual design approach involved in the proposed framework for the design and implementation of generators of adapted and varied activities. Drawing on a generic modelling of the questioned facts (i.e., representation of the questions about facts present in the conceptual model of the activity), this approach enables the generation of varied gameplays independently of any didactic domain.

However, it is important to note that this conceptual modelling is based on several justified but subjective game design choices (i.e., the **Roguelite** game genre with an economic game mechanic allowing the purchase of items having an impact on the selection of gameplays). Therefore, modifying this specific context would require to re-specify and re-design the game (meta-)model and possibly the gameplay categories, thereby requiring to re-specify and re-design the relationships between the task types and gameplay categories. Nevertheless, such a change in context and its consideration in the approach is beyond the scope of this thesis.

In order to adopt this conceptual modelling approach and support the implementation of generators, we have developed a software infrastructure based on Model-Driven Engineering principles, encompassing the various concepts previously presented. The following chapter details this software infrastructure.

SOFTWARE INFRASTRUCTURE

Contents

8.1	Model-Driven Engineering Foundations	113
8.1.1	Conceptual Models to Computerised Metamodels	114
8.1.2	Models as Inputs and Outputs of Generation	118
8.2	Activity Generation Algorithm	119
8.2.1	Algorithm for Generating Training Game Activities	119
8.2.2	Algorithm for Generating Questions about Facts	122
8.3	Extension Rules	123
8.4	Synthesis	125

Now that all the requirements have been defined for the design of generators of adapted and varied game activities for declarative knowledge training, the next step consists in considering how to implement these generators. More specifically, our focus is about answering the fourth research question: *How to specify every information required for generation to enable computer interpretation for the development of activity generators?*

The main idea is to offer a tool that facilitates the implementation of generators, mainly for engineers/developers, in particular by making it possible to reuse existing code and data. Our proposal consists of an **extensible software infrastructure** (i.e., a framework in the sense of software engineering), based on the principles of Model-Driven Engineering, providing a generation algorithm, models, and metamodels that limit the extension to the addition of elements related to the targeted didactic domain (Lemoine *et al.* 2024a).

This chapter aims to present the various components of this infrastructure and how its extension to a didactic domain can be achieved. First, we present the MDE foundations upon which the infrastructure is based by introducing a translation of the conceptual models, presented in Chapter 7, into machine-interpretable metamodels and models. Then, we describe the activity generation algorithm (i.e., generic) and the question about facts algorithm (i.e., extension). Finally, we introduce the rules for extending the framework.

8.1 Model-Driven Engineering Foundations

In order to enable the use of information by the generation algorithm, this information must be specified in a machine-interpretable format. By using Model-Driven Engineering, in particular model and metamodel concepts, such information can be specified in order to be used (i.e., generation of the code related to the metamodel concepts) and interpreted

by the generation algorithm. Accordingly, our conceptual models, presented in Chapter 7, have been translated into metamodels using the Eclipse Modeling Framework (EMF) plug-in (Steinberg *et al.* 2009). On the basis of these metamodels, models conforming to them can be created to detail the concrete elements to use for generation. Physically, metamodels are XML files. The EMF tooling provides a visual representation close to UML (Unified Modeling Language) class diagram notation. This representation is used in our approach for its human readability. Figure 8.1 illustrates this approach.

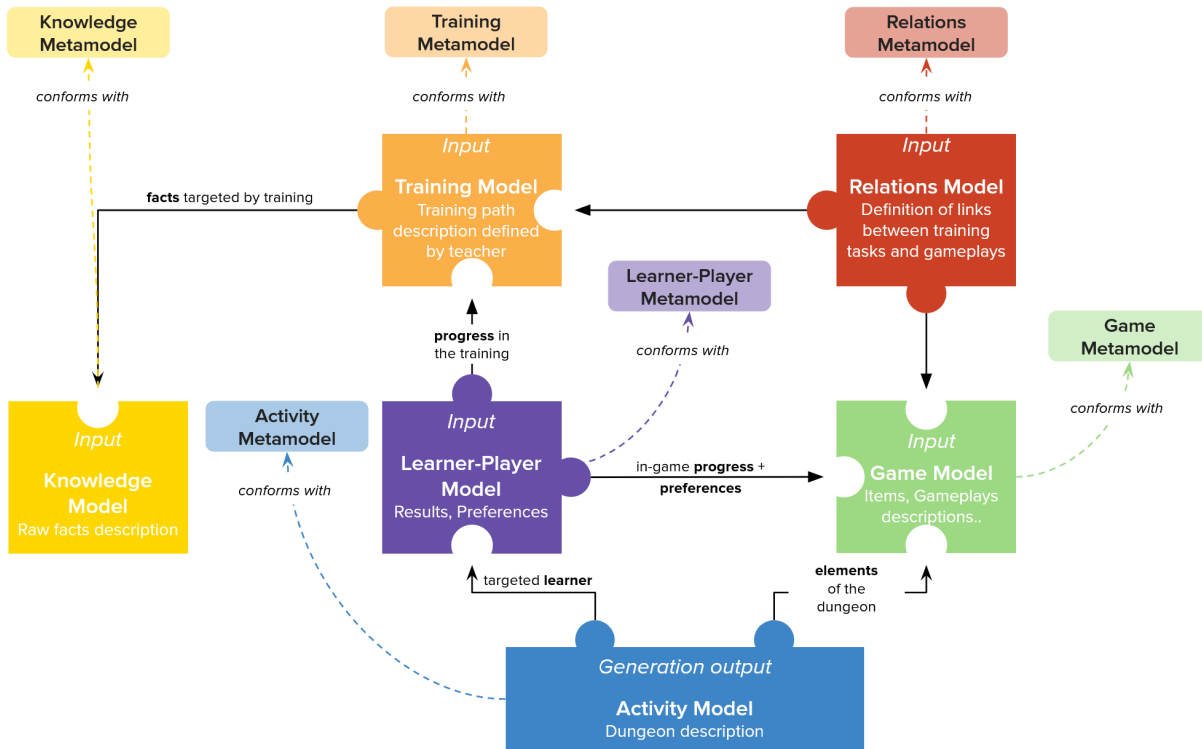


Figure 8.1 – Interconnected models conform to metamodels involved in activity generation

8.1.1 Conceptual Models to Computerised Metamodels

The various interconnected metamodels are computerised representations in *ecore* format (i.e., a format defining the concepts that can be manipulated in EMF) of the conceptual models previously identified. The metamodels define the structure of the different data (i.e., input and output) involved in the generation process, and enable the automatic generation of the code associated to the concepts that are modelled (i.e., productivity of the models provided by the MDE approach). It should be noted that modelling and meta-modelling are subjective activities since they depend on the modeller’s interpretation of the domain and the conventions and styles they are accustomed to (e.g., specific naming of elements, structure and appearance of the metamodels). This section presents each of the metamodels involved in the framework.

First, Figure 8.2 depicts the knowledge metamodel, which represents the yellow portion of the domain conceptual model (see Figure 7.2) and describes the structure of the knowledge of the domain, i.e., knowledge is composed of sets of facts having possible parameters (e.g., they can belong to a visualisation, have an image as visual representation).

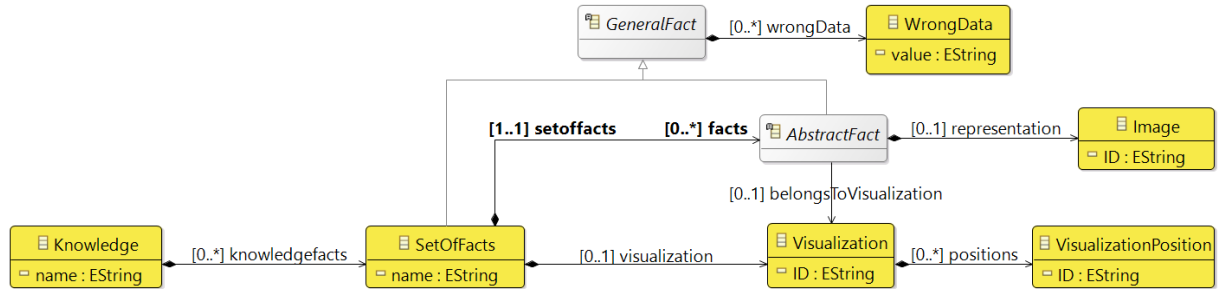


Figure 8.2 – Knowledge metamodel

Then, Figure 8.3 displays the training metamodel, which represents the orange portion of the domain conceptual model (see Figure 7.2) and outlines the structure of the training paths defined by the teachers in terms of objectives with prerequisites, levels with completion criteria, training tasks having four subtypes corresponding to the four task types and associated to a response modality¹.

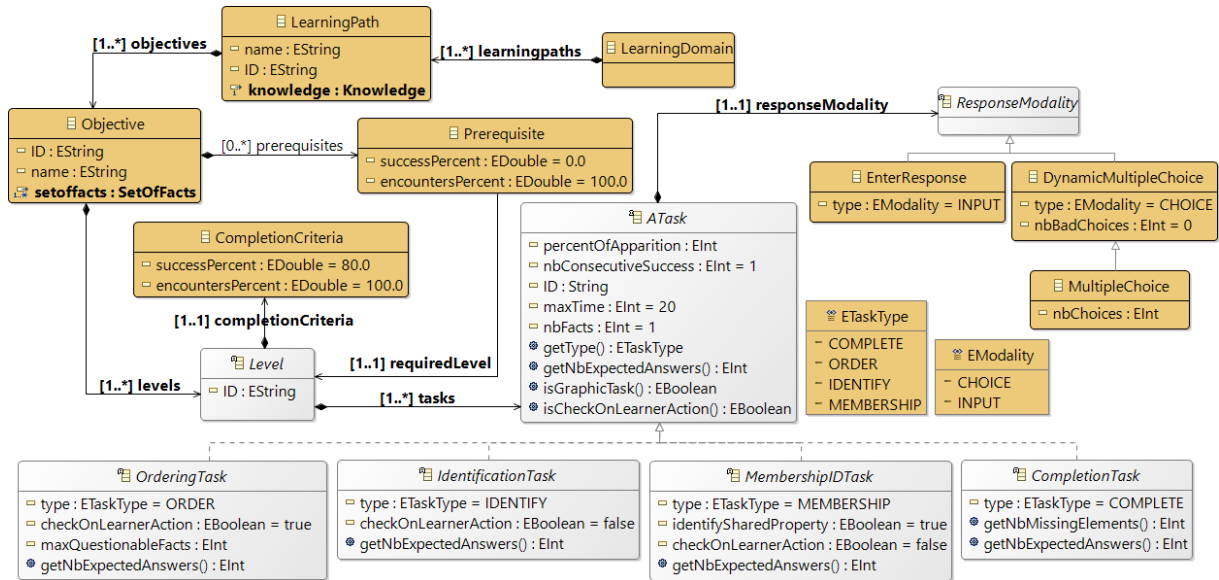


Figure 8.3 – Training metamodel

1. Note that there are two types of choice response modality: **MultipleChoice** which allows expressing the total number of choices and the number of bad choices expected, and **DynamicMultipleChoice** that only allows expressing the number of bad choices. This is purely strategic to facilitate the specification of models for the tasks where the number of good answers can automatically be computed.

Next, Figure 8.4 depicts the game metamodel, which represents the game conceptual model (see Figure 7.3) and details the structure of the game progression, the different gameplays, the different types of game elements (i.e., gameplay elements and equipments), the types of rooms, the abilities, and the curses.

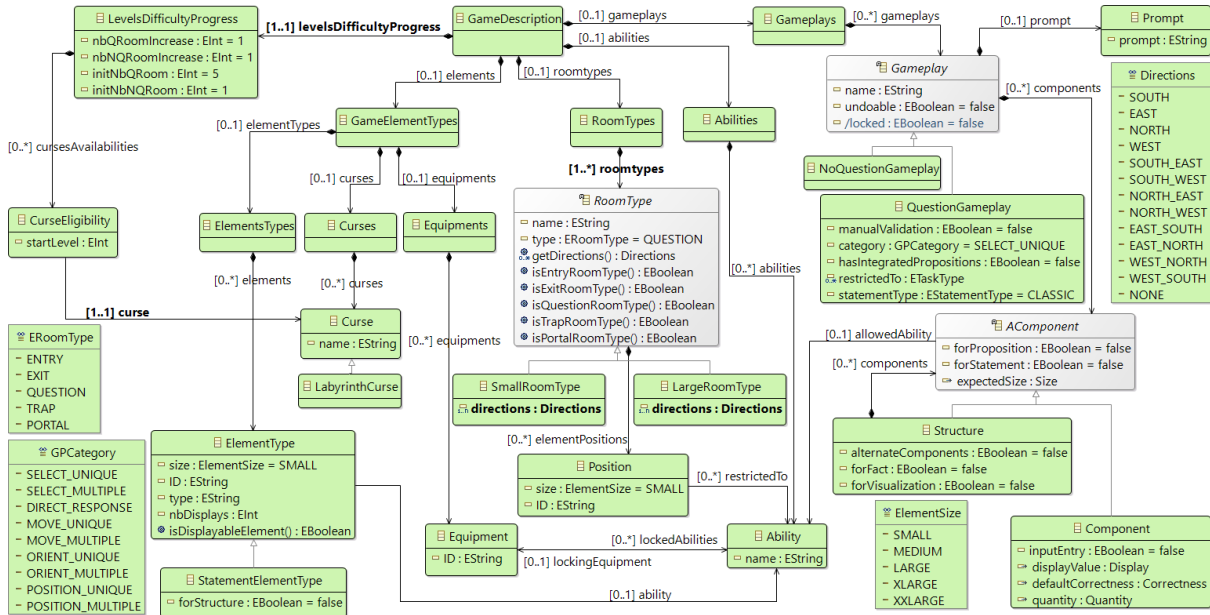


Figure 8.4 – Game metamodel

To continue, Figure 8.5 displays the learner-player metamodel, which represents the learner-player conceptual model (see Figure 7.5) and outlines the structure of the learner-player’s progress in training (i.e., results per task for each objective/level pair) and in the game (i.e., game level reached), as well as the structure of their game preferences (i.e., equipment purchased and activated/deactivated).

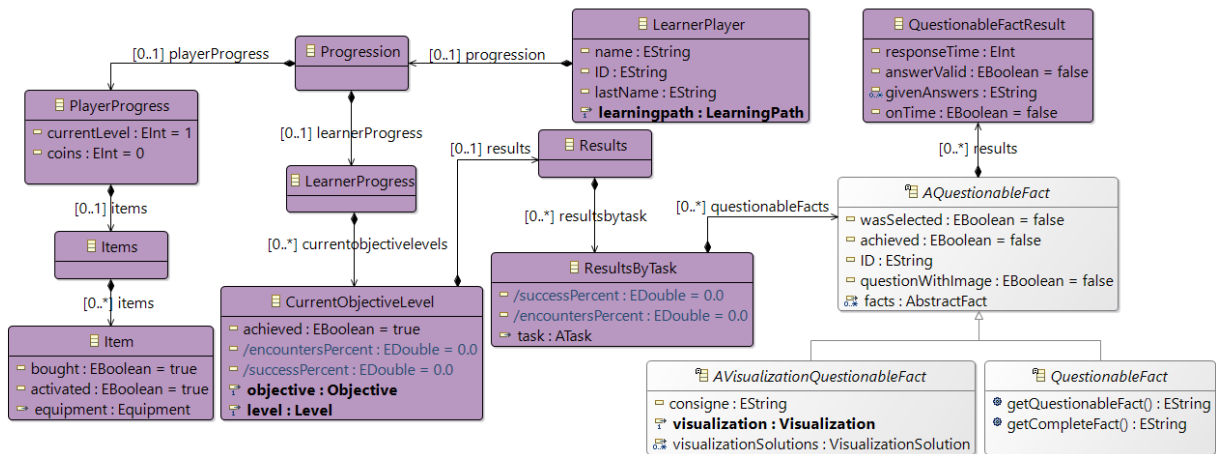


Figure 8.5 – Learner-player metamodel

In addition, Figure 8.6 depicts the relation metamodel, which represents the relation conceptual model (see Figure 7.7) and details the structure of the conditional relations between training task types and gameplay categories that have been defined in Chapter 6.

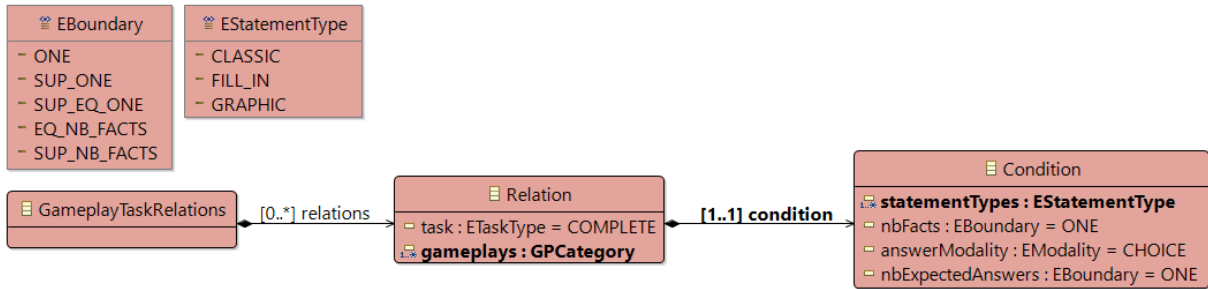


Figure 8.6 – Relation metamodel

Finally, Figure 8.7 displays the activity metamodel, which represents the activity conceptual model (see Figure 7.4), and describes the structure of the training game activities for declarative knowledge, i.e., dungeons which are composed of rooms having accesses to others rooms, positioned elements², and questioned facts. Every time an activity is demanded, a model conforming to the activity metamodel has to be generated.

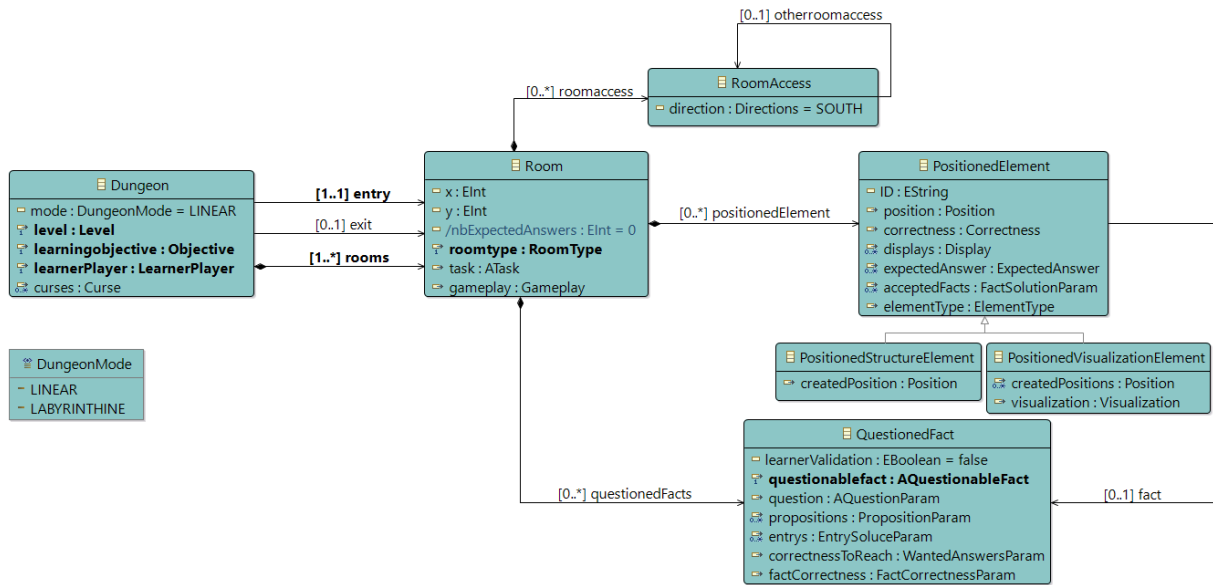


Figure 8.7 – Activity metamodel (to be generated)

2. Two subtypes of positioned elements have been included: one for structure components that can create an inner position (i.e., assigned to each element of the structure) and another for structure components intended to display a visualisation that references the visualisation and can create inner positions (i.e., each of the positions of the visualisation).

8.1.2 Models as Inputs and Outputs of Generation

Concrete data on a given didactic domain or on the game are described through models. Models are conforming to metamodels and are represented as XMI (*XML Metadata Interchange*) files. The generation algorithm requires several models as input and builds a model as output. The required input models are:

- The knowledge model which describes the raw facts to work on, such as multiplication tables, judo techniques and historical dates.
- The training model which describes the different training paths. Note that the knowledge and training models and metamodels depend on the didactic domain targeted.
- The game model which describes all the concrete game elements available to the generator, i.e., describes the different abilities available (e.g., *pushable*, *movable*, *rotatable*), the different gameplays (see Section 7.2.2), the different room types, the game progression, the different curses (e.g., labyrinthine, only one life, dark mode), the types of elements (e.g., jars, blocks, statues), the equipments (e.g., power belt unlocking the ability to push objects).
- The learner-player model which describes results, progress, and preferences of a learner-player. Hence, a model conforming to the learner-player metamodel must be created for each learner-player. These models have to be automatically updated after each completed game level to allow the generation algorithm to take into account new results (i.e., updates must be performed by the game → independent component of the framework, see Figure 5.2).
- The relation model³ which describes the relations presented in Figure 6.7.

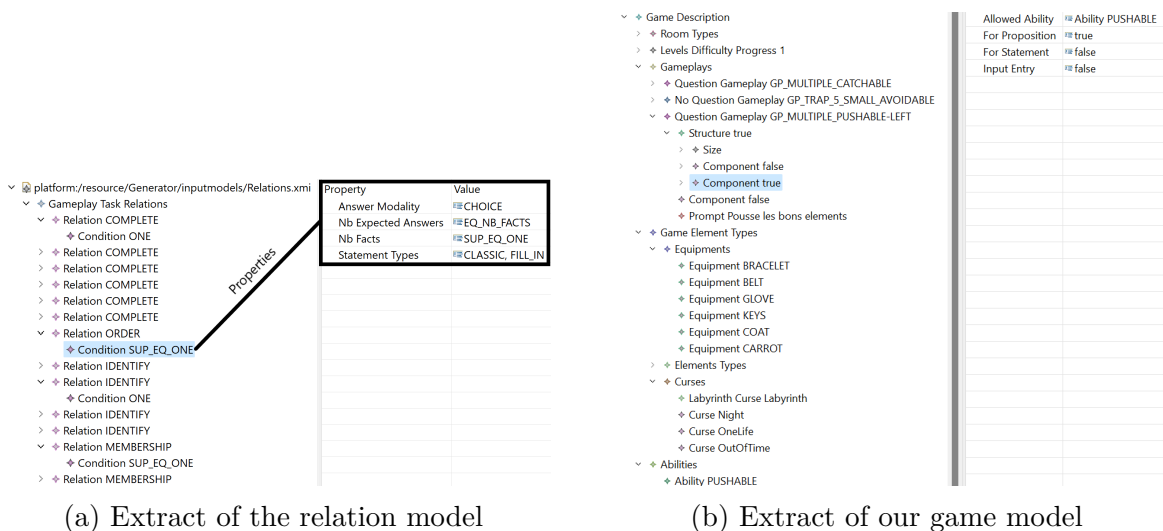


Figure 8.8 – Tree-based EMF model view of models with properties on the selected node

3. Note that the relation model should not be modified or remade, as relations are not supposed to change unless new task types or gameplay categories have been created (or in case of disapproval of our approach). Therefore, this model has already been created.

Figure 8.8 presents examples of tree-views of the relation and game XMI models.

As a result, generation automatically produces an activity model describing a dungeon for an objective/level pair of the training path of a learner-player based on the information provided in the input models.

8.2 Activity Generation Algorithm

The core idea behind the activity generation algorithm is to capture as many common elements as possible, and only add the code necessary for the elements related to the didactic domain, in particular, the creation of questionable facts and the methods for instantiating questioned facts within the generation algorithm. Figure 8.9 illustrates this idea. The following subsections present the different steps of the activity generation algorithm and specify the extension mechanism used to enable the creation of questionable facts and the associated methods.

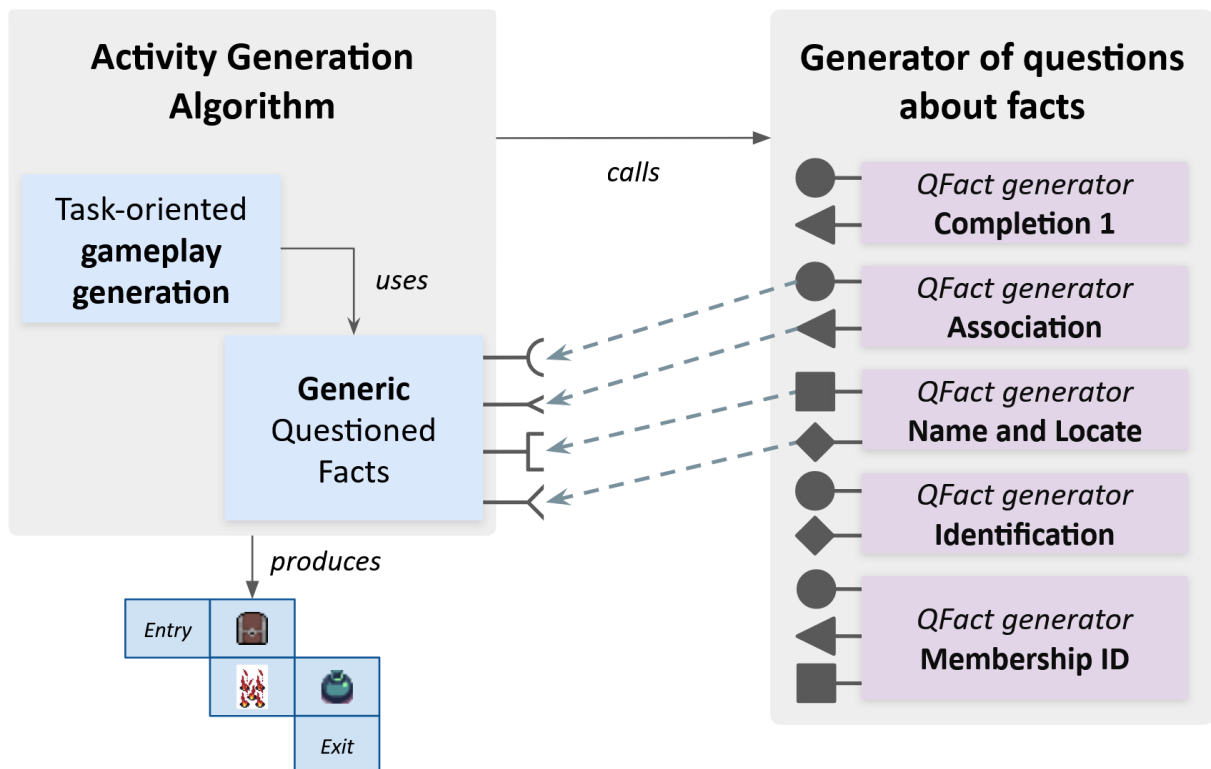


Figure 8.9 – Principle of extension for questions on facts generations

8.2.1 Algorithm for Generating Training Game Activities

The activity generation algorithm has been developed in Java and consists of a procedural algorithm decomposed of several incremental steps (similar to [Laforcade and](#)

Laghouaouta (2018) and Sehaba and Hussaan (2013)). As depicted in Figure 8.10, the algorithm is divided into four parts that are broken down into nine steps:

- P1) selection of all dungeon elements (i.e., steps one to six);
- P2) creation of the structure of the dungeon (i.e., step seven);
- P3) instantiation of the elements of the dungeon (i.e., step eight);
- P4) transformation of the dungeon (i.e., step nine).

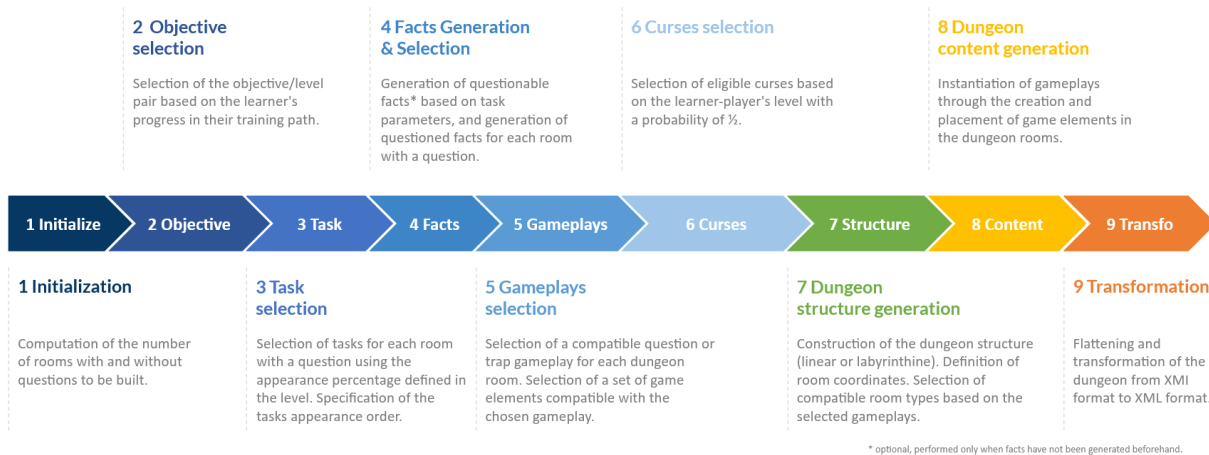


Figure 8.10 – Activity generation algorithm steps

The algorithm works as described below. First, it computes the number of trap rooms and question rooms. Then, it selects an eligible objective/level pair from the learner's training path based on their previous results and progress. An objective/level pair is considered eligible as soon as their prerequisites are satisfied (i.e., the learner's results meet the conditions of the prerequisites). In our current proposal, an objective/level pair is randomly selected from the set of eligible ones. However, strategies to choose this pair, defined by teachers or experts for example, could be implemented as proposed by Melero *et al.* (2016)⁴.

Next, for each question room of the future dungeon, tasks are chosen by the algorithm based on the selected level (i.e., levels are described by a set of training tasks to complete). These tasks are selected based on their percentage of appearance and learners' progress in their training path. Once a task has been completed, it no longer appears in the dungeons, and its percentage of appearance is distributed proportionally to the other tasks.

To continue, if the objective/level pair had never been selected before, its related questionable facts are generated. Next, according to the previously selected tasks, the required number of questioned facts for the dungeon are generated based on these questionable facts. Note that the questioned/questionable fact generation methods called by the activity algorithm are specific to the didactic domain (see Section 8.2.2 for more details on questioned/questionable fact generation).

4. Note that for each random choice regarding educational content, a predefined strategy or heuristic could be implemented.

After that, a gameplay (i.e., trap or question) is selected for each room (i.e., except for entry and exit rooms) as well as compatible game elements. As a reminder, since player’s preferences consists of bought and activate/deactivate items that unlocks abilities which describes gameplays, selected gameplays are restricted based on players’ preferences. Finally, the curses in the dungeon are selected based on the learner’s current game level and the game progression defined, with a probability of one out of two.

The generation of the structures of the dungeons is based on the principle of *Grid-Based Dungeon Generator*, i.e., space is divided into cells into which rooms can be placed. However, our approach differs in that two types of room are managed (*small* = 1 square cell and *large* = 4 square cells), creating a need to handle overlapping rooms. Large rooms are necessary to deal with tasks requiring large visualisations such as maps. For linear dungeon generation, the algorithm is based on a principle of *backtrack* in order to avoid dead-ends due to room types not having all possible accesses (i.e., north, south, east, west, north-east, etc.) making generation impossible. No such problem arises for labyrinthine dungeon generation, as each room is eligible at each iteration.

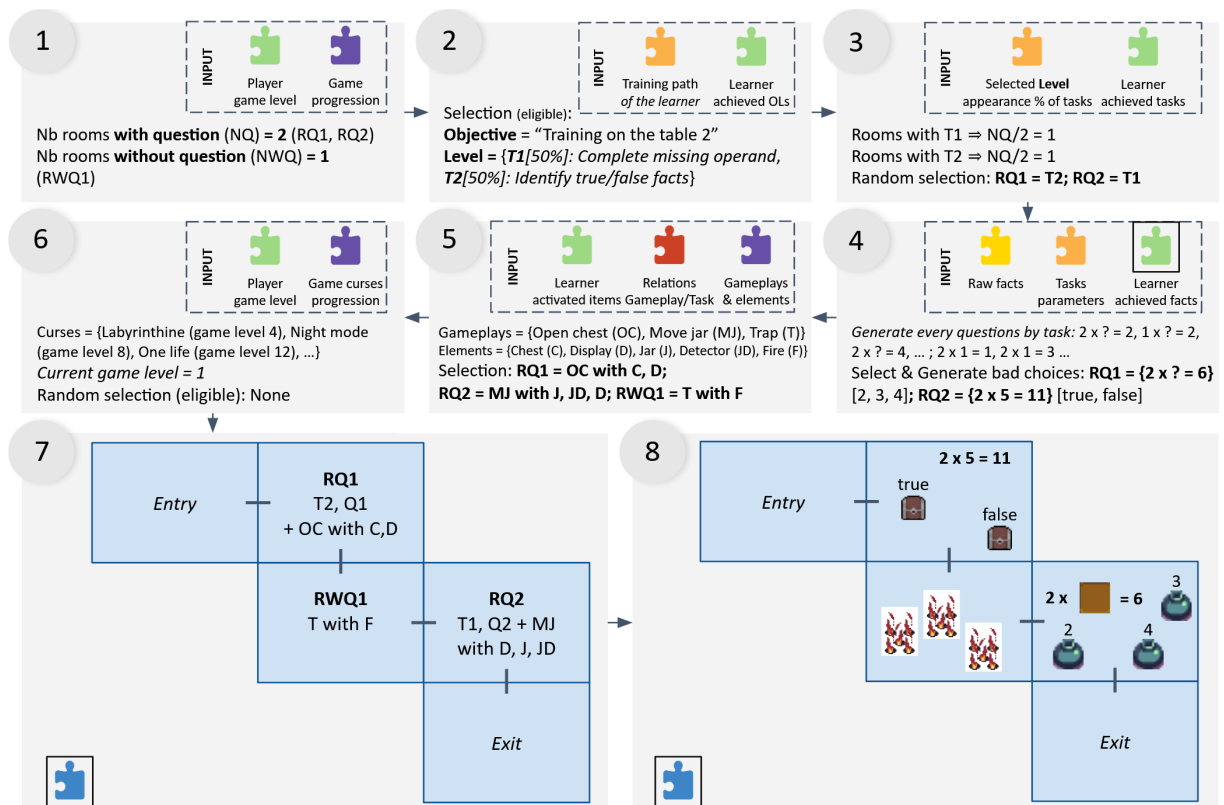


Figure 8.11 – Step-by-step example of the generation algorithm. Puzzle pieces colours correspond to Figure 8.1 and puzzle pieces with borders present data modified or created by the algorithm

Then, the next step consists in initialising the content of the dungeon in terms of game

elements based on every previously made choices. The aim is to correctly instantiate the elements selected for each room of the dungeon according to the gameplays and facts questioned. To this end, the values of the game elements (e.g., texts to be displayed, proposals carried by the objects) are defined in order to allow the questioning of a specific fact by associating the parameters of the facts being questioned with those described in the gameplays (see Section 7.2.3).

Figure 8.11 presents a step-by-step example of the algorithm. Finally, an operational step (not presented in Figure 8.11) consists of transforming the XMI model of the dungeon into an XML file. The main benefits of this transformation are to remove references to other XMI models/files (i.e., to flatten the model) and to make the system more portable. (i.e., many plugins interpret XML, but not XMI). This transformation is performed using the *Epsilon Transformation Language* (ETL), see transformation source code in Appendix D.

8.2.2 Algorithm for Generating Questions about Facts

As previously explained, raw facts have parameters that depend on the didactic domain targeted. For example, a fact representing a multiplication can be modelled as a class containing three integers x (operand), y (operand) and res (result). Whereas, an historical date would be modelled by a class containing a string (event) and a date or period (integers). Furthermore, questionable facts (i.e., intermediate form of question about facts present in the learner’s model) are also domain-dependent since only questioned facts (i.e., question about facts present in the activity model) have a generic form.

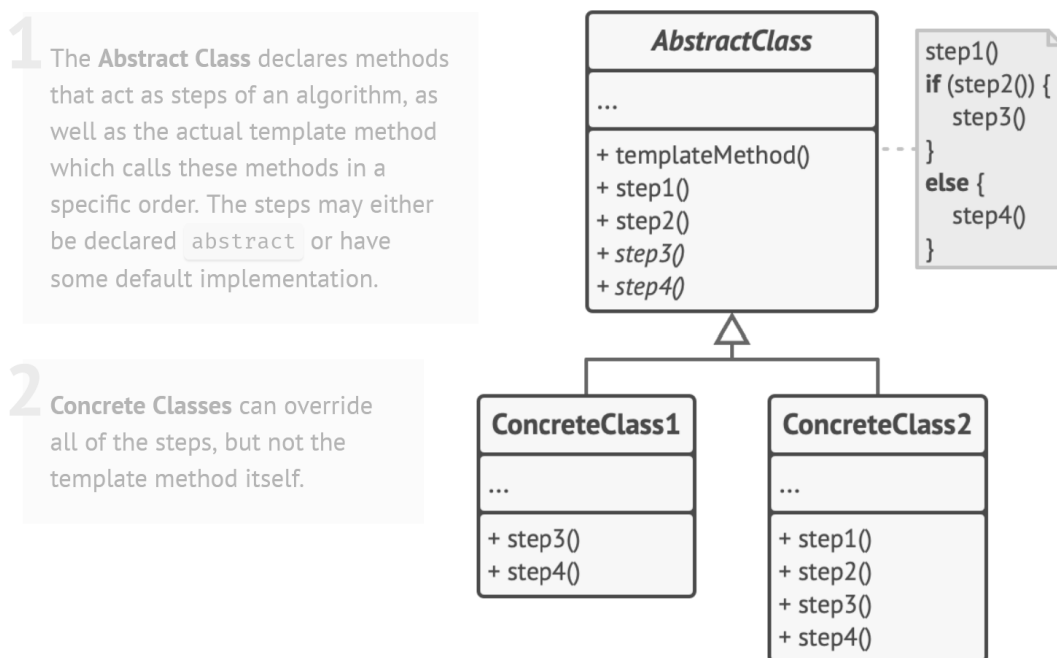


Figure 8.12 – Structure of the template method design pattern (Shvets 2018)

Note that it would be possible to make a transition directly from the raw facts to the questioned facts, but it would involve an additional challenge to understand the generic form of the questions. Using this intermediate form allows the definition of your own object to manipulate (i.e., questionable fact), which makes it easier to create methods for instantiating the questioned facts, since you are handling an object that you have modelled.

As a result, questionable fact generation algorithms and the methods allowing activity generators to instantiate questioned facts must be implemented during the extension of the framework. In order to guide this extension, the fact generation algorithm of the framework follows a design pattern called *template method* (see Figure 8.12) which allows for a domain-independent main part and domain-dependent extensions.

In the main part (i.e., corresponding to the **AbstractClass** in Figure 8.12), the skeleton of the generation code is captured, i.e., all the invariants relative to generation. By contrast, the extensions (i.e., corresponding to the **ConcreteClass(es)** in Figure 8.12) focus on the implementation of the methods used for the creation of the questionable facts and the instantiation of the parameters of the questioned facts (i.e., only the necessary methods are implemented depending on the task and the questionable fact concerned). Consequently, for each type of questionable fact (i.e., one for each training task of the didactic domain), a concrete class of the abstract main class is created.

8.3 Extension Rules

In order to build a domain-specific activity generator, the framework must be extended at different levels, in particular: metamodels have to be specified to declare domain-specific parameters, accordingly models have to be designed to describe domain-related information and the code for generating questions about the facts has to be developed.

Regarding the metamodels introduced, extension points (i.e., areas that must be extended in order to build a generator) are represented as abstract classes that do not already have concrete subclasses. Three metamodels are targeted by this extension: the knowledge metamodel, the training metamodel and the learner-player metamodel. Raw facts, wrong data, levels, training tasks, and questionable facts have different structures/-parameters depending on the didactic domain targeted. For instance, the method for building tables (operand \times table or table \times operand) depends solely on the mathematics discipline. Moreover, in a completion task (e.g., finding the historical date, or the result of a multiplication), the element to be sought depends on the domain (e.g., result, operand or table for multiplication, or event or date for history). Therefore, these elements have to be modelled according to the domain.

Moreover, as questionable facts are domain-dependent, the same applies to their generation (see the previous section). Finally, to be able to generate activities, a model conforming with each metamodel has to be created: a knowledge model describing the raw facts, a game model describing the game elements available to the generator, a relations model enabling the generator to maintain the coherence of the activities, a training

model describing the training path and a learner-player model. The learner-player model requires only minimal information: identifier, training path, and creation of empty progressions (i.e., game and training), since this model must be updated according to the learner-player’s results in the game. In addition, certain models are defined by default in the framework, such as the relation model and a version of a game model ⁵, and are therefore reusable. Table 8.1 summarises for each model: who creates it, when it is created, if it is modified, when it is modified and by whom.

MODEL	WHO CREATES IT?	WHEN IS IT CREATED?	IS IT MODIFIED?	WHEN IS IT MODIFIED?	WHO MODIFIES IT?
Knowledge model	Engineer based on experts input	When developing the generator	–	When errors are present	Engineer
Training model	Teachers through authoring tools or engineer based on experts input	When developing the generator	✓	When creating/modifying/removing a training path	Teachers or engineer
Learner-Player model	Engineer based on experts input (only basic information)	When developing the generator	✓	Each time the player completes a dungeon	The game
Game model	Engineer	When developing the generator	✓	When adding new game elements	Engineer
Relation model	Engineer	When developing the generator	–	In case of disagreement or errors	Engineer
Activity model	Generation algorithm	When the generator is running	✗		

Table 8.1 – Summary of the creation and use of the models required for activity generation (✗ = not modified, ✓ = modified, – = can be modified, but should not be)

5. A *game engine* has been developed to interpret and deliver playable dungeons, thus a game model related to this engine (i.e., the elements available to the engine) is provided in the framework.

In conclusion, extending the framework in order to design a domain-specific activity generator entails three main steps:

1. Extending the metamodels according to the targeted domain at the level of:
 - (a) the raw facts (i.e., *AbstractFact*)
 - (b) the questionable facts (i.e., *AQuestionableFact*)
 - (c) the levels (i.e., *Level*)
 - (d) the tasks (i.e., *CompletionTask*, *IdentificationTask*, *MembershipTask*, *Ordering-Task*)
2. Specifying models that conform to the metamodels (i.e., XMI files):
 - (a) a knowledge model
 - (b) a training model describing a training path
 - (c) a learner-player model (i.e., with minimal information)
 - (d) a game model (*optional*, possibility of using the default one)
 - (e) a relations model (*optional*, possibility of using the default one)
3. Implementing the generators of questions about facts for each task specific to the didactic domain by following the *template* provided.

Appendix E presents a step-by-step guide for extending the framework.

8.4 Synthesis

In this chapter, the final aspect of our contribution has been presented: an extensible software infrastructure, implemented in Java, allowing the implementation of *RogueLite* oriented generators of varied game activities adapted to learner-players for declarative knowledge training. This infrastructure encompasses the conceptual approach presented in Chapter 7, which is based on the elements and methods presented in Chapters 5 and 6.

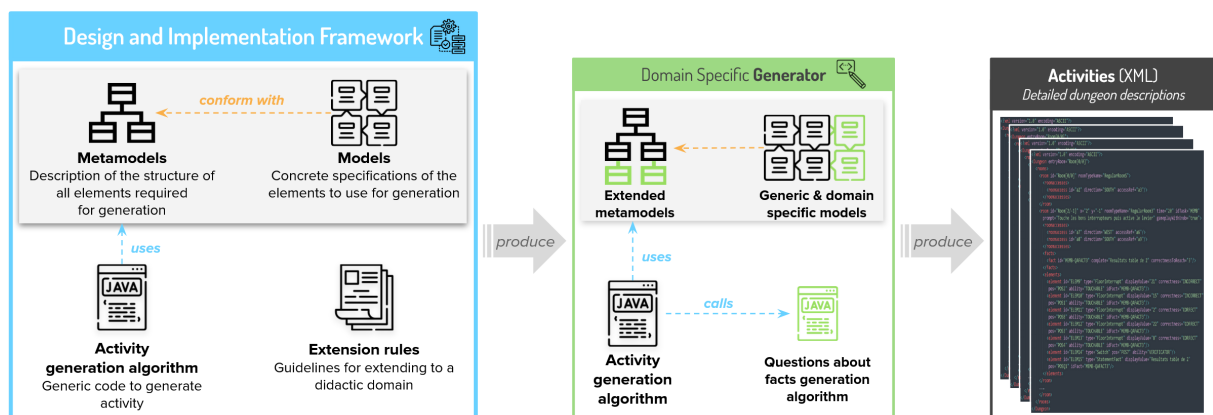


Figure 8.13 – Design framework and generators components overview

PART III

Application & Evaluation

EXTENSIONS OF THE FRAMEWORK

Contents

9.1	Generator for Multiplication Tables Training	130
9.2	Generator for History-Geography Facts Training	134
9.3	Generator for Judo Facts Training	137
9.4	Generator for Solar System Facts Training	141
9.5	Discussion	144

Since the proposed framework is founded on a conceptual approach integrated within a software architecture that is resting on various previously defined properties (see Section 5.1), its assessment relies on verifying that these properties are respected. Therefore, **the objective is not to evaluate the learning effect provided, especially since the research object is activity generators and not educational games¹, but rather the ability to produce generators that respect various constraints.**

Underpinning the framework are two key properties: (*FP1*) possibility of expressing different didactic domains, and (*FP2*) possibility of expressing teachers' views on individual learners' training (see Section 5.1). Assessing *FP1* requires a focus on the ability of the framework to be extended to different didactic domains. To that end, the framework has to be extended to at least two different didactic domains, such as multiplication tables and history-geography facts. Assessing *FP2* involves highlighting the ability to specify different training paths proposed by different teachers. This involves modelling at least two training paths proposed by two different teachers. It should be noted that to evaluate *FP2*, variations in the didactic domain are not relevant as the creation of training paths is similar across domains, therefore the domain has no influence on the ability to model different training paths.

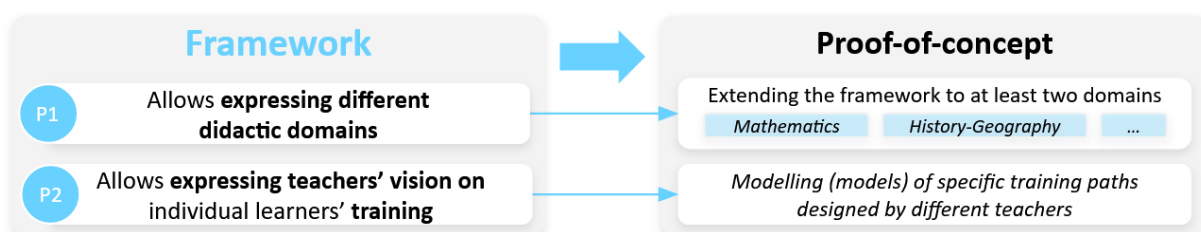


Figure 9.1 – Overview of the evaluation of the framework through proof-of-concept

1. Generators produce activities that have to be interpreted by another software component, the *game engine*, and cannot be directly played by players.

This chapter aims to evaluate these two properties by presenting different applications of the framework (see Figure 9.1). First, we present an extension to the context of AdapTABLES (i.e., multiplication tables). Then, we depict an extension to the context of history-geography facts required for the *Diplôme National du Brevet des Collèges* (i.e., French exam taken in 9th grade). Next, we present an extension for judo facts, i.e., techniques and referees gestures. Then, an extension for two training tasks for solar system facts is presented. Finally, a discussion about the framework is provided.

9.1 Generator for Multiplication Tables Training

As part of the exploratory study conducted within the AdapTABLES project, five training tasks for multiplication tables have been defined with mathematics experts (see Section 5.3): COMPLETION 1, COMPLETION 2, RECONSTRUCTION, IDENTIFICATION, MEMBERSHIP IDENTIFICATION. The study has also enabled us to specify different parameters for building questions about facts (i.e., multiplication can be questioned in different ways), such as the method for building tables with the “operand \times table or table \times operand”, the position of the equal symbol “on the left or on the right”, the min/max range of multiplications questioned (i.e., a teacher may only want to work on the table of three from 1 to 5: $3 \times 1, 3 \times 2, 3 \times 3, 3 \times 4, 3 \times 5$).

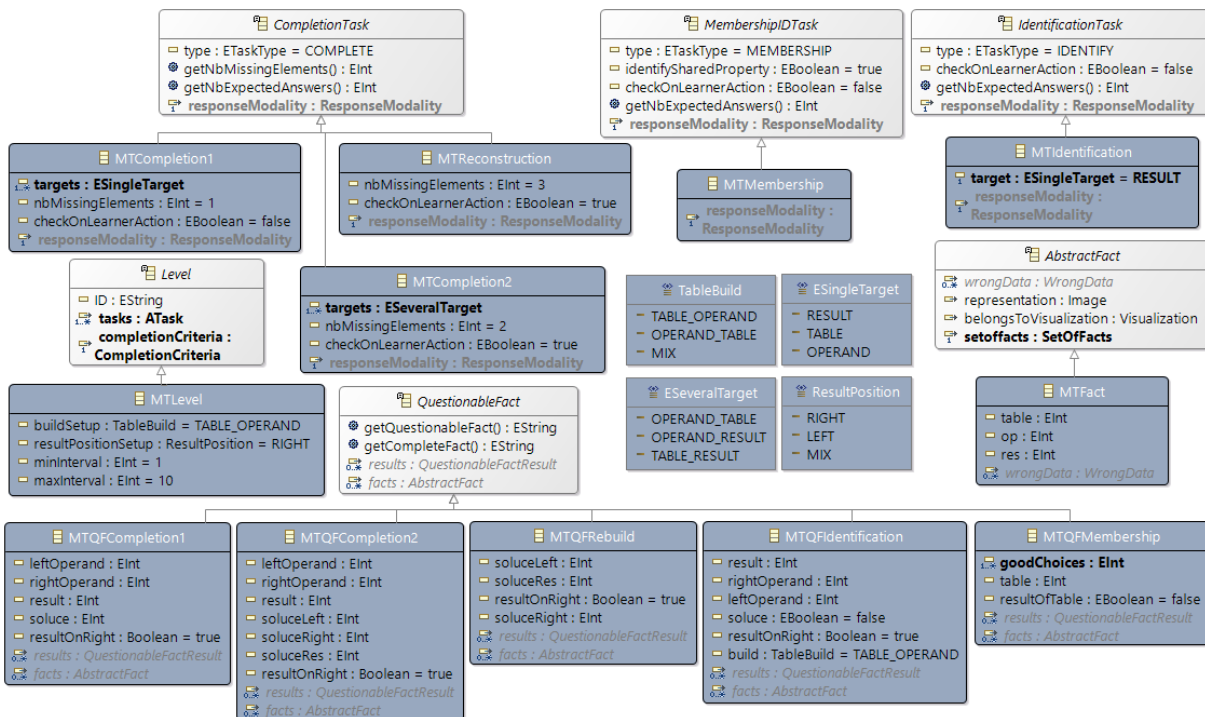


Figure 9.2 – Extension (in blue) of the metamodels for multiplication tables

Once the specificities of the domain have been defined, we followed the rules of the ex-

tension guide (see Appendix E) to design and implement a dedicated generator. First, the necessary metamodel elements have been specified (see Figure 9.2), namely:

- a raw fact type that represents multiplications (i.e., *MTFact*) composed of three integers: the table, the operand and the result.
- the five task types (i.e., *MTCompletion1*, *MTCompletion2*, *MTReconstruction*, *MT-Membership*, *MTIdentification*) being subclasses of the generic types (i.e., *CompletionTask*, *MembershipIDTask*, *IdentificationTask*).
- a level (i.e., present in the training paths) composed of the different building parameters (e.g., *resultPositionSetup* corresponding to the position of the equal) of the questions on the facts (i.e., *MTLevel*).
- for each task, a specific type of questionable fact (i.e., subclasses of *Questionable-Fact*) has been created. Usually, one or more questionable facts are constructed from a single raw fact, depending on the table construction parameters. However, in the case of the membership identification task type (i.e., identifying the results of a table), questionable facts are built from several raw facts, depending on the task settings (i.e., expected number of choices).

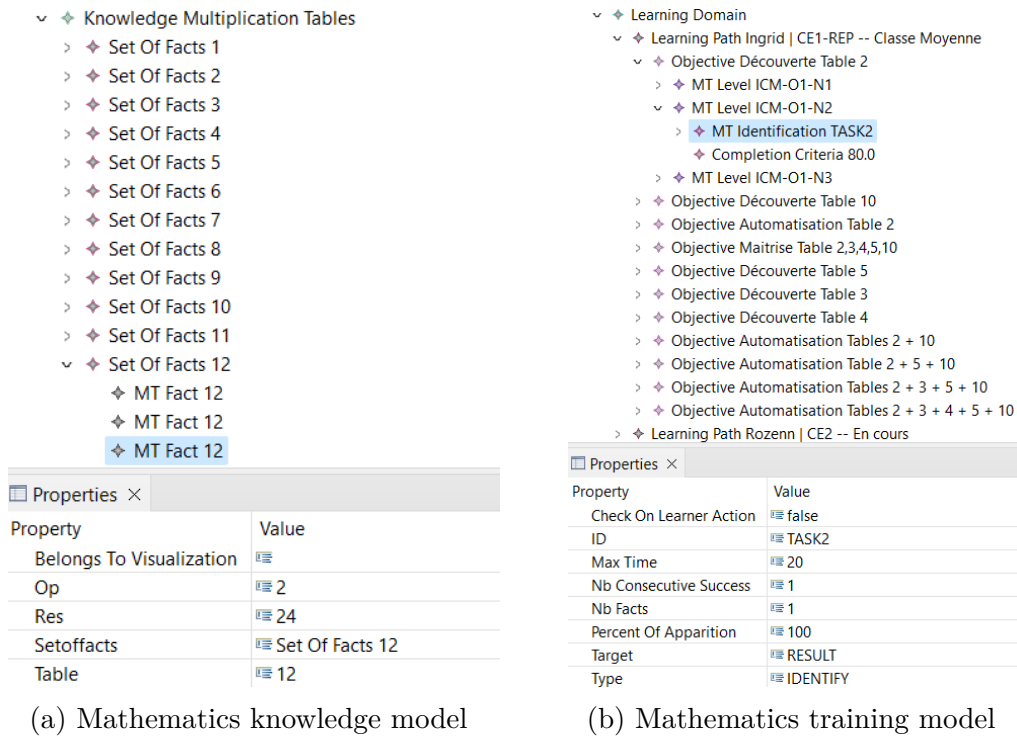


Figure 9.3 – Tree-based EMF views of mathematic models

Then, we specified the knowledge model (see Figure 9.3a) and developed the fact question generators for each of the five tasks by following the “template method”. Listing 9.1 presents an extract of the implemented code by displaying the method required to instantiate the questionable facts for the RECONSTRUCTION task (i.e., *MTQFRReconstruction*).

Moreover, we specified training paths with two teachers, one from CE1 (2nd grade) and one from CE2 (3rd grade), in order to model different groups of learner levels in their respective classrooms. Both training paths have been modelled as shown in Figure 9.3b.

Listing 9.1 – Example of generateQuestionableFactsOf implementation for the RECONSTRUCTION task

```

1  @Override
2  protected Set<AQuestionableFact> generateQuestionableFactsOf(ATask task, AbstractFact
   fact) {
3      if(fact instanceof MTFact) {
4          MTFact mF = (MTFact) fact;
5          int min = ((MTLevel) dungeonElements.getChosenLevel()).getMinInterval();
6          int max = ((MTLevel) dungeonElements.getChosenLevel()).getMaxInterval();
7          if(min <= mF.getOp() && mF.getOp() <= max){
8              Set<AQuestionableFact> qfs = new HashSet<>();
9              TableBuild build = ((MTLevel) dungeonElements.getChosenLevel()).getBuildSetup
   ();
10             ResultPosition equalPos = ((MTLevel) dungeonElements.getChosenLevel()).
   getResultPositionSetup();
11             if(build.equals(TableBuild.MIX)) {
12                 if(equalPos.equals(ResultPosition.MIX)) {
13                     qfs.add(buildQF(mF, ResultPosition.LEFT, TableBuild.OPERAND_TABLE));
14                     qfs.add(buildQF(mF, ResultPosition.RIGHT, TableBuild.OPERAND_TABLE));
15                     qfs.add(buildQF(mF, ResultPosition.LEFT, TableBuild.TABLE_OPERAND));
16                     qfs.add(buildQF(mF, ResultPosition.RIGHT, TableBuild.TABLE_OPERAND));
17                 } else {
18                     qfs.add(buildQF(mF, equalPos, TableBuild.TABLE_OPERAND));
19                     qfs.add(buildQF(mF, equalPos, TableBuild.OPERAND_TABLE));
20                 }
21             } else {
22                 if(equalPos.equals(ResultPosition.MIX)) {
23                     qfs.add(buildQF(mF, ResultPosition.LEFT, build));
24                     qfs.add(buildQF(mF, ResultPosition.RIGHT, build));
25                 } else {
26                     qfs.add(buildQF(mF, equalPos, build));
27                     qfs.add(buildQF(mF, equalPos, build));
28                 }
29             }
30             return qfs;
31         }
32     }
33     return new HashSet<>();
34 }
35
36 private MTQFRebuild buildQF(MTFact fact, ResultPosition resPos, TableBuild build) {
37     MTQFRebuild qf = new MTQFRebuildImpl();
38     qf.setID(taskID+"-QAFAC"+factsCounter); factsCounter++;
39     if(build.equals(TableBuild.OPERAND_TABLE)) {
40         qf.setSoluceLeft(fact.getOp());
41         qf.setSoluceRight(fact.getTable());
42     }else {
43         qf.setSoluceRight(fact.getOp());
44         qf.setSoluceLeft(fact.getTable());
45     }
46     qf.setSoluceRes(fact.getRes());
47     qf.setResultOnRight(resPos.equals(ResultPosition.RIGHT));
48     qf.getFacts().add(fact);
49     return qf;
50 }

```

As part of the AdapTABLES project, a *game engine* has been developed, using the Unity game engine and employing C# scripts, to interpret game levels (i.e., activities/departments) as 2D representations. This *game engine* allows:

- 1) the translation and interpretation of XML generated in playable dungeons;
- 2) the visualisation of the dungeon structure (i.e., maps);
- 3) to play the game levels.

Figure 9.4 displays screenshots of dungeon rooms presenting an example of gameplay for each multiplication tables training task. The generator has been implemented with French prompts/proposals, therefore, texts present in the screenshots are in French.



(a) COMPLETION 1



(b) COMPLETION 2



(c) RECONSTRUCTION



(d) IDENTIFICATION



(e) MEMBERSHIP IDENTIFICATION

Figure 9.4 – Examples of gameplays for multiplication tables training tasks interpreted by the *game engine*

9.2 Generator for History-Geography Facts Training

The *Diplôme National du Brevet des Collèges* (DNB) is an exam taken at the end of the 9th Grade in France, and certifies that learners have mastered the common base of knowledge, skills and culture defined by the government. In the context of history and geography, a number of declarative knowledge items, known as *repères*, are required to support higher learning levels (e.g., understanding, application, analysis of Bloom (1956)'s taxonomy) in these didactic domains².

Discussions with two history-geography teachers and an examination of previous exam papers (i.e., the questions asked on these *repères*) have led to the identification of six training tasks: ASSOCIATION, LEGENDING A MAP, IDENTIFICATION, TIMELINE, NAME AND LOCATE, MEMBERSHIP IDENTIFICATION.

Similarly to mathematics, once the tasks and their parameters have been defined, we followed the rules of the extension guide (see Appendix E) to design and implement a dedicated generator. First, the necessary metamodel elements have been specified (see Figure 9.5), namely:

- three types of raw facts:
 - a type representing historical events (i.e., *HistoryFact*) composed of a string corresponding to the event, a time (i.e., *Time* corresponding to a date or a period that have a position on a timeline), and a possible visual representation (e.g., image of people destroying Berlin's Wall to represent its fall).
 - a type representing map legends (*GeographyLegendFact*) composed of a text and a visual representation (i.e., symbol) both having a position on a map.
 - a type representing geographical elements on maps (*GeographyFact*) composed of a type representing the type of elements targeted (e.g., cities, countries, regions, etc.), a string representing the corresponding value, a category (i.e., members of European Union, France regions), and a position on a map.
- the six task types (i.e., *LocateOnAMap*, *GeographyMembership*, *HistoryIdentification*, *HistoricalEventAssociation*, *LegendAMap*, *HistoricalChronology*³) being subclasses of the generic types.
- a level without any specific parameters (i.e., *HGLevel*).
- like for mathematics, a specific type of questionable fact (i.e., subclasses of *QuestionableFact* for textual questions or *AVisualizationQuestionableFact* for questions based on a *Visualization*) has been created for each task.

Then, we specified the knowledge model (see Figure 9.6a) representing the principal *repères* and developed the fact question generators for each of the six tasks by following the “template method”. Listing 9.2 presents an extract of the implemented code by displaying the method required to instantiate the questionable facts for the LEGEND task (i.e., *LegendQuestionableFact*).

2. The common reference proposed by the government for history-geography is available at <https://www.education.gouv.fr/bo/13/Hebdo42/MENE1327027N.htm>.

3. Note that TIMELINE is named `HistoricalChronology` in the model.

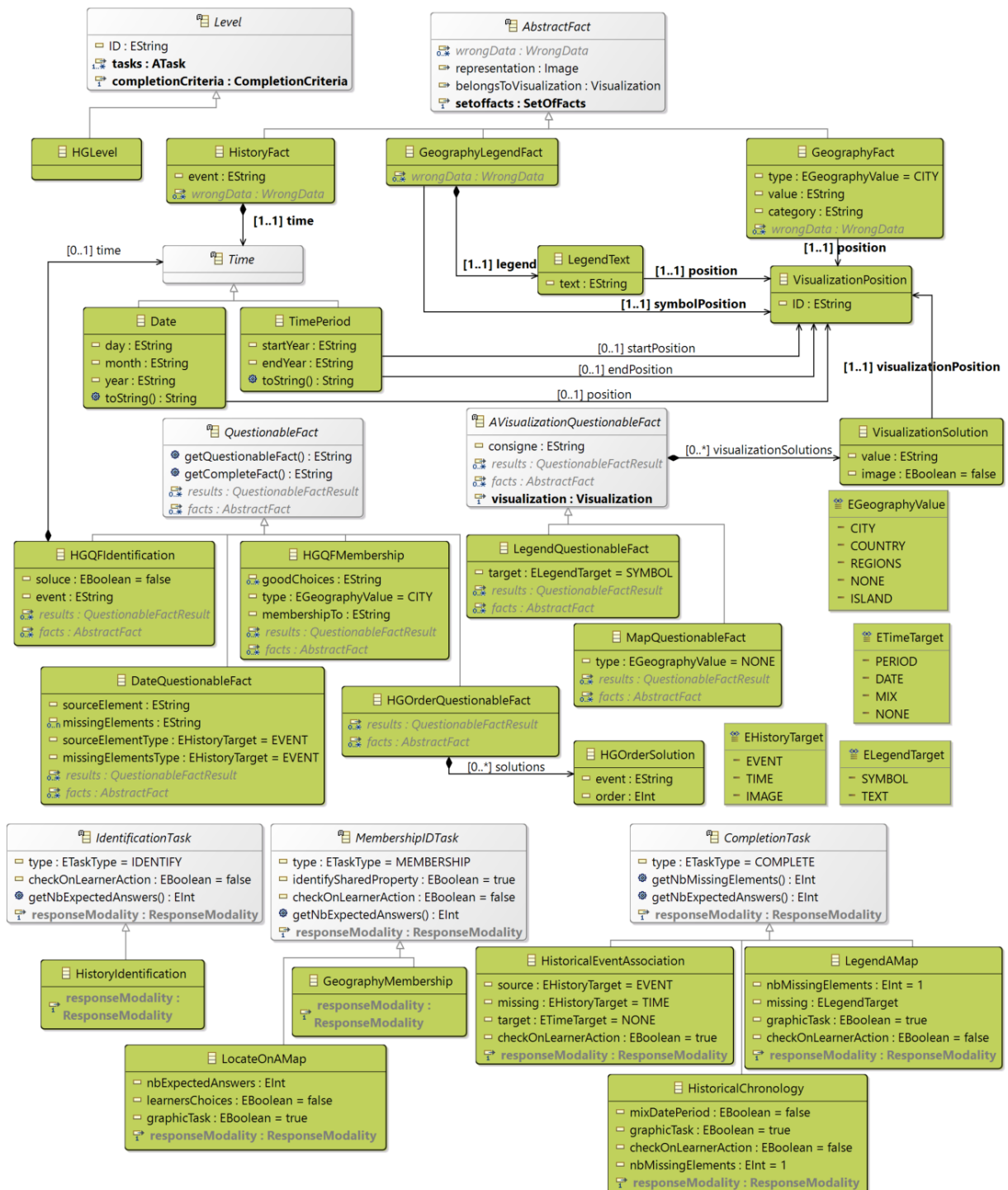


Figure 9.5 – Extension (in green) of the metamodels for history-geography facts

Additionally, a training path composed of a single objective/level pair targeting every fact and proposing every task has been modelled to test the generator (see Figure 9.6b).

However, due to a lack of time to exchange, no training path has been developed with teachers. This would be an interesting and relevant perspective.

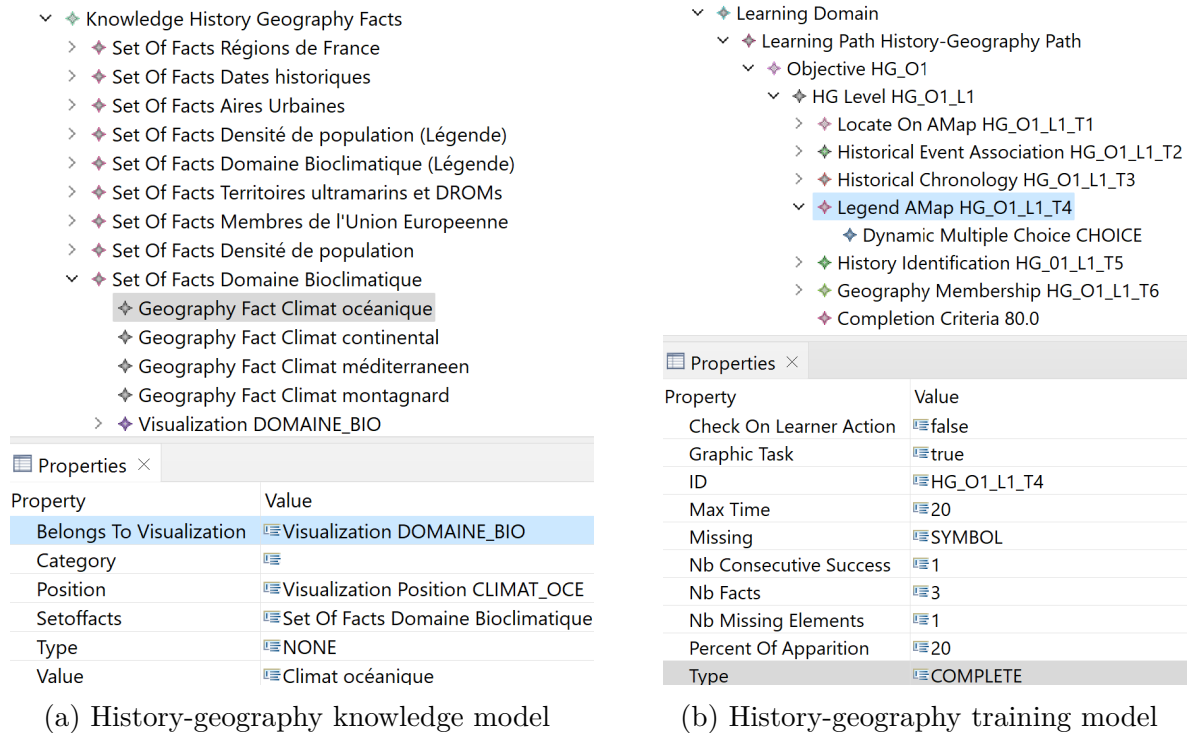


Figure 9.6 – Tree-based EMF views of history-geography models

Listing 9.2 – Example of generateQuestionableFactsOf implementation for the LEGEND task

```

1  @Override
2  protected Set<AQuestionableFact> generateQuestionableFactsOf(ATask task, AbstractFact
   fact) {
3      Set<AQuestionableFact> questionableFacts = new HashSet<>();
4      if(fact instanceof GeographyLegendFact && fact.getBelongsToVisualization() != null) {
5          questionableFacts.add(buildQF((LegendAMap) task, (GeographyLegendFact) fact));
6      }
7      return questionableFacts;
8  }
9
10 private AQuestionableFact buildQF(LegendAMap task, GeographyLegendFact fact) {
11     AVisualizationQuestionableFact qf = new LegendQuestionableFactImpl();
12     qf.setID(taskID+"-QFACT"+factsCounter); factsCounter++;
13     qf.setVisualization(fact.getBelongsToVisualization());
14     if(task.getMissing().equals(ELegendTarget.SYMBOL)) {
15         qf.getVisualizationSolutions().add(buildVisualizationSolution(fact.
16             getRepresentation().getID(), fact.getSymbolPosition(), true));
17     } else {
18         qf.getVisualizationSolutions().add(buildVisualizationSolution(fact.getLegend().
19             getText(), fact.getLegend().getPosition(), false));
20     }
21     qf.getFacts().add(fact);

```

```

20     return qf;
21 }
    
```

Finally, the *game engine* has been completed to add objects called *prefabs* allowing the interpretation of pictures/images such as maps. Therefore, Figure 9.7 displays interpreted dungeon rooms presenting examples of gameplay for each history-geography training task.

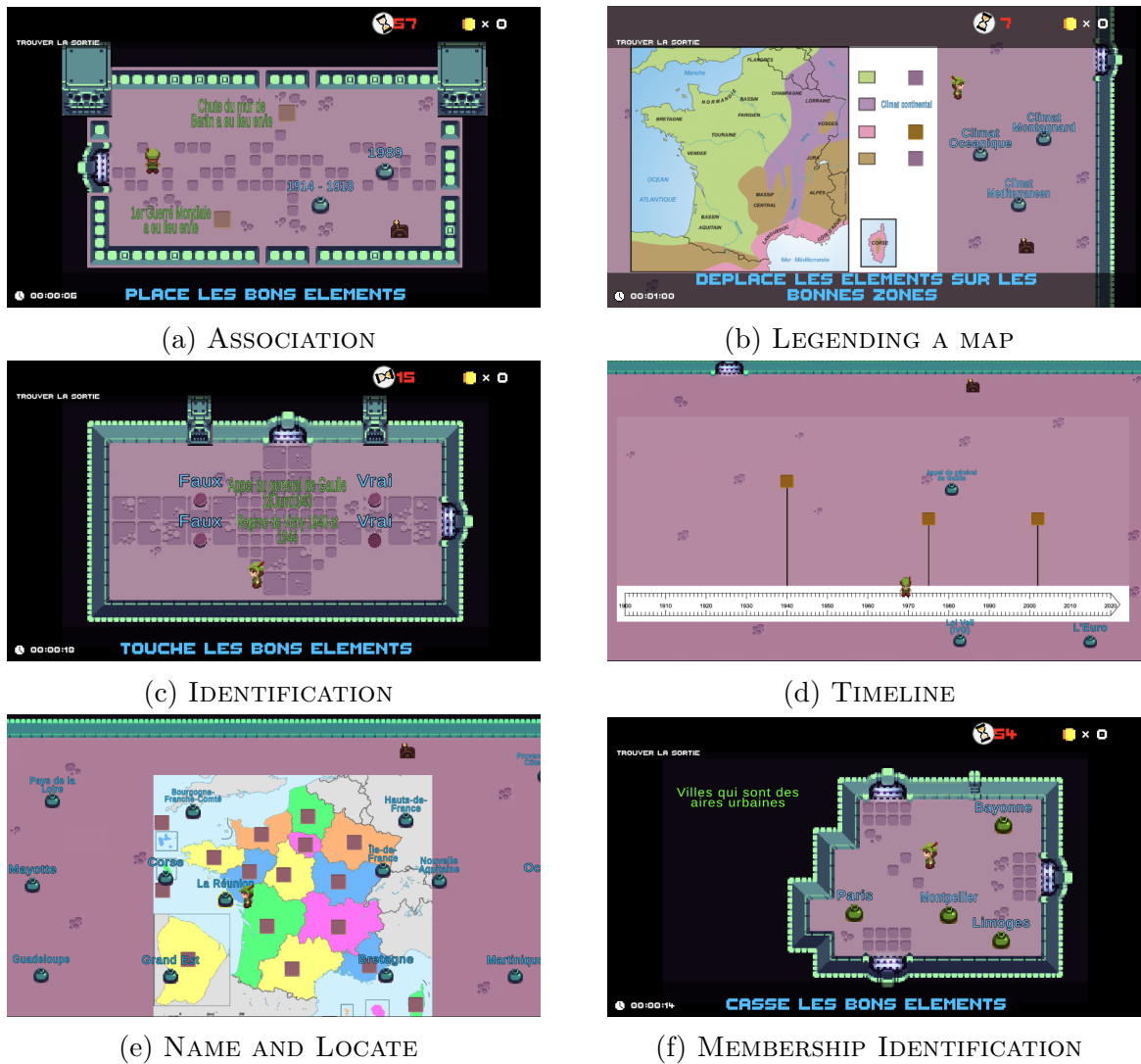


Figure 9.7 – Examples of gameplays for history-geography facts training tasks interpreted by the *game engine*

9.3 Generator for Judo Facts Training

Judo is a Japanese martial art created in 1882 by Jigorō Kanō. The progression is achieved by learning and mastering techniques specific to each level. The judokas evolve

through a system of coloured belts (i.e., white, yellow, orange, green, blue, brown, black) whereby each belt represents a level of skill and knowledge acquired. Obviously, the skills to be acquired cannot be taught through our approach, as they are know-how. However, the knowledge to be acquired is declarative: naming/identifying techniques and knowing refereeing gestures (i.e., essential for any fighter). Accordingly, our proposal is to extend the framework to provide training for the knowledge required in judo.

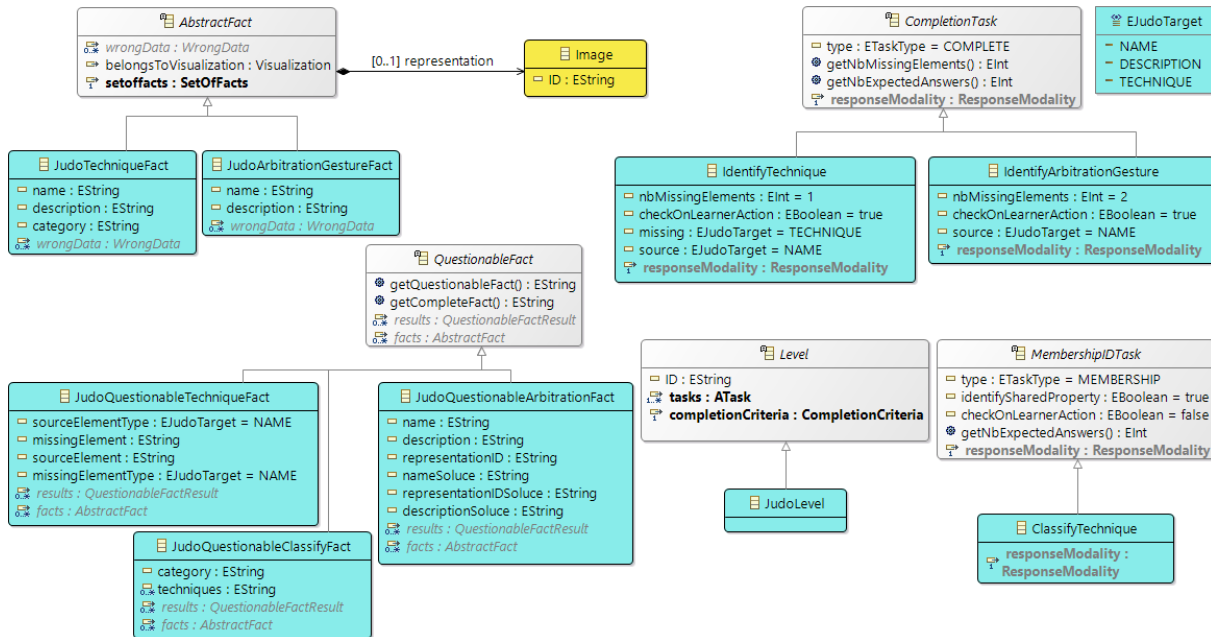


Figure 9.8 – Extension (in cyan) of the metamodels for judo facts

On the basis of the needs for knowledge in the context of judo, we have identified three training tasks:

- **TECHNIQUE IDENTIFICATION**, i.e., associate two items of a judo technique together from name of technique, representative image, description⁴ (e.g., “Hiza-Guruma is?”, with a set of picture proposals corresponding to judo techniques).
- **REFEREEING GESTURES IDENTIFICATION**, i.e., complete the two missing elements of a fact composed of the referee’s announcement, the meaning of his announcement and the associated gesture (e.g., “HAJIME – ? – ?” with a set of proposals like [start of the fight, image “hand-alongside-body”]). Usually, this task involves interrogating several facts simultaneously, and none of the questioned facts have “incorrect choices”, as these choices are those of the other questioned facts.
- **TECHNIQUE CLASSIFICATION**, i.e., identify the techniques belonging to a specific category such as sacrifice or immobilisation (e.g., [O-soto-gari, Uki-waza, Hiza-guruma] “Which are Ashi-waza (legs) techniques?”).

4. A description is often given by instructors to describe the techniques, e.g., *Ippon-seoi-nage*: shoulder throw from one side.

Like before, once the tasks have been defined, we followed the rules of the extension guide (see Appendix E) to design and implement a dedicated generator. First, the necessary metamodel elements have been specified (see Figure 9.8), namely:

- two types of raw facts:
 - a type representing judo technique composed of three strings and an image: the name, the description, the category, and a visual representation of the technique (i.e., *JudoTechniqueFact*).
 - a type representing judo referee gestures composed of two strings and an image: the name, the description, and a visual representation of the gesture (i.e., *JudoGestureFact*).
- the three task types (i.e., *IdentifyTechnique*, *IdentifyArbitrationGesture*, *ClassifyTechnique*) being subclasses of the generic types.
- a level without any specific parameters (i.e., *JudoLevel*).
- a specific type of questionable fact (i.e., subclasses of *QuestionableFact*) has been created for each task.

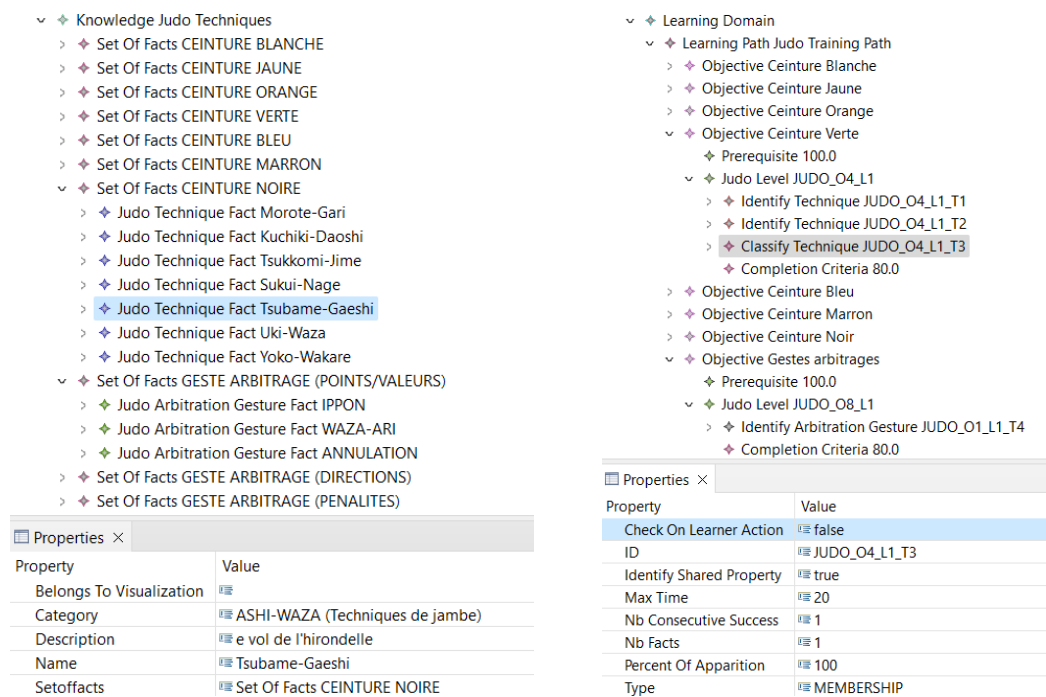


Figure 9.9 – Tree-based EMF views of judo models

Then, we specified the knowledge model (see Figure 9.9a) representing the principal judo techniques grouped according to the colour of the belt (i.e., gradual learning depending on the colour of the belt in judo) and developed the fact question generators for each of the three tasks. Additionally, since the training path is usually the same for all learners, our approach has been to model a single training path consisting of one objective for each

coloured belt (see Figure 9.9b) and one for referee gestures. Each objective has a single level whose prerequisite is the level of the lower belt (e.g., the yellow belt objective has as its prerequisite the level of the white belt objective), i.e., except the referee gesture objective which does not have any prerequisites. Finally, each level has three tasks: two technique identification tasks (i.e., one in which the description must be associated to the name, and the other in which the corresponding image must be associated to the name), and a classification task. However, the objective regarding referee gestures only has a single referee gestures identification task. Obviously, other paths can be created, but this path allows us to carry out tests and serves mainly as a proof-of-concept.

Then, the extension for generating questions about facts for each of the tasks has been implemented by following the “template method”. Listing 9.3 presents an extract of the implemented code by displaying the method required to instantiate the questionable facts for the TECHNIQUE CLASSIFICATION task (i.e., *JudoQuestionableClassifyFact*).

Listing 9.3 – Example of `generateQuestionableFactsOf` implementation for the TECHNIQUE CLASSIFICATION task

```

1  @Override
2  protected AQuestionableFact generateQuestionableFactOf(ATask task, List<AbstractFact>
   facts) {
3      JudoQuestionableClassifyFact qf = new JudoQuestionableClassifyFactImpl();
4      qf.setID(taskID+"-QAFAC"+factsCounter); factsCounter++;
5      qf.setCategory(((JudoTechniqueFact) facts.get(0)).getCategory());
6      for (AbstractFact judofact : facts) {
7          qf.getTechniques().add(((JudoTechniqueFact) judofact).getName());
8          qf.getFacts().add(judofact);
9      }
10     return (AQuestionableFact) qf;
11 }

```

Finally, the *game engine* has been completed to add objects called *prefabs* allowing the interpretation of images, i.e., techniques and gestures. Figure 9.10 displays interpreted dungeon rooms presenting examples of gameplay for each history-geography training task.



(a) TECHNIQUE IDENTIFICATION



(b) REFEREEING GESTURES IDENTIFICATION



(c) TECHNIQUE CLASSIFICATION

Figure 9.10 – Examples of gameplays for judo facts training tasks interpreted by the *game engine*

9.4 Generator for Solar System Facts Training

In the next chapter, an experiment involving the extension of the framework by an external engineer is presented (see Chapter 10). In order to guarantee the feasibility of the instructions provided to the engineer, we have extended beforehand the framework to the solar system domain. This extension has been chosen as a test case for the experiment with the engineer.

The solar system is a system formed by eight planets, including the Earth, orbiting around a star, the Sun. Planets are characterised by their distance from the sun and their orbit (i.e., the curved trajectory formed as it moves through space). Many characteristics could be explored in this domain, but our modelling focuses solely on these two.

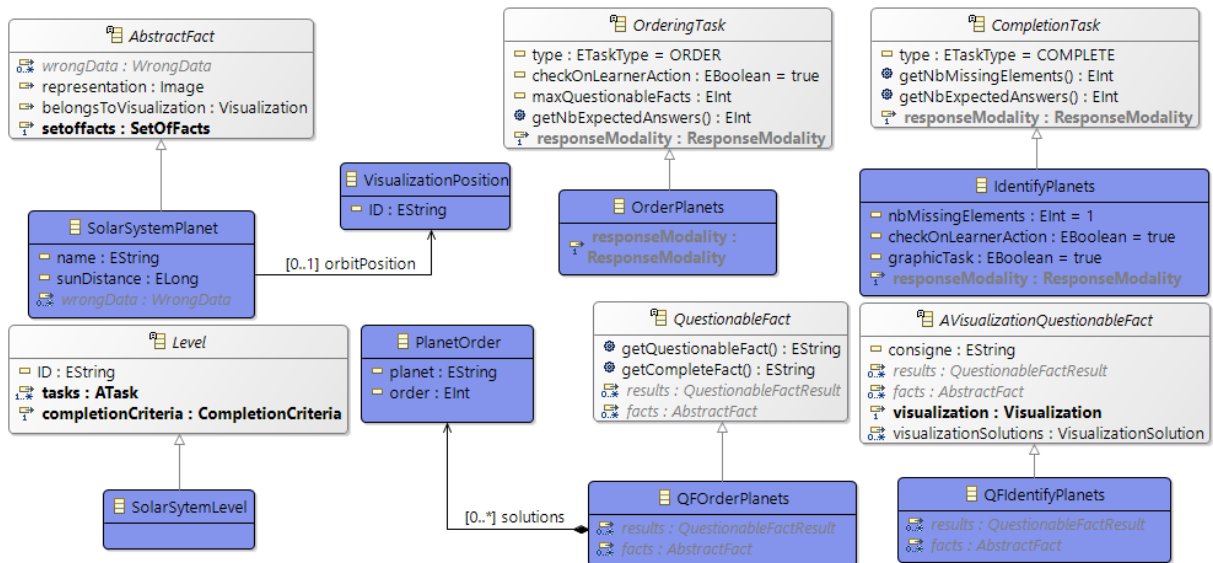


Figure 9.11 – Extension (in violet) of the metamodels for solar system facts

For the experiment, our decision has been to only define two training tasks to limit the time required for extending the framework. These two tasks are the following:

- ORDER PLANETS, i.e., order planets according to their distance from the sun (e.g., [Mars, Mercure, Jupiter, Saturn] “Order from closest to farthest from the sun.”).
- IDENTIFY PLANETS, i.e., fill-in the map of orbits by positioning the planets (e.g., a map of the orbits in which the planets must be correctly positioned).

These tasks, to be implemented, have been strategically chosen to cover the definition of a subclass of ordering task (i.e., *OrderingTask*), which was not covered by the other implemented domains, and a graphical task in order to impose the creation of two different types of questionable facts (i.e., *AQuestionableFact* and *AVisualizationQuestionableFact*).

As before, the rules of the extension guide (see Appendix E) have been followed to design and implement a dedicated generator. First, the necessary metamodel elements have been specified (see Figure 9.8), namely:

- a single type of raw facts representing planets of the solar system composed of a String, a long integer, and the position on a map of its orbit (i.e., *SolarSystemPlanet*).
- both task types (i.e., *IdentifyPlanets*, *OrderPlanets*) being subclasses of the generic types (i.e., *CompletionTask*, *OrderingTask*).
- a level without any specific parameters (i.e., *SolarSystemLevel*).
- a specific type of questionable fact (i.e., subclasses of *QuestionableFact* or *AVisualizationQuestionableFact*) has been created for each task.

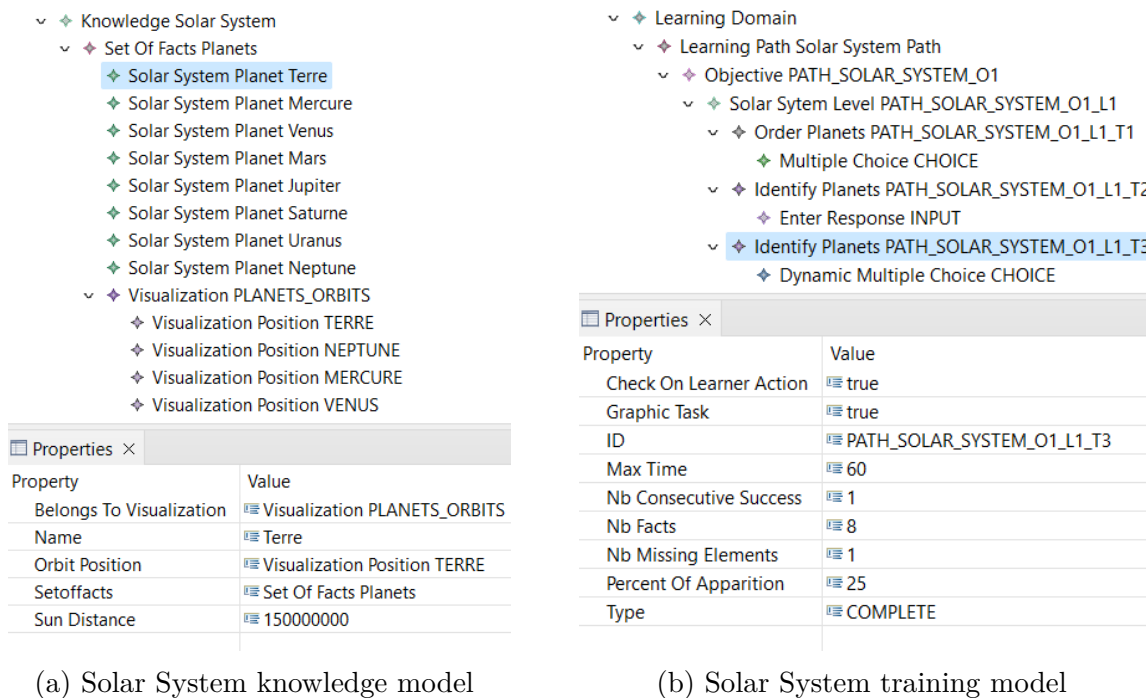


Figure 9.12 – Tree-based EMF views of solar system models

Then, we specified the knowledge model (see Figure 9.12a) representing the planets and developed the fact question generators for both tasks by following the “template method”. Listing 9.4 presents an extract of the implemented code by displaying the method required to instantiate the questionable facts for the ORDER PLANETS task (i.e., *QFOrderPlanets*).

Additionally, a training path composed of a single objective/level pair proposing an *OrderPlanets* task and two *IdentifyPlanets* tasks (i.e., one with input, the other with choice) has been modelled to test the generator (see Figure 9.12b).

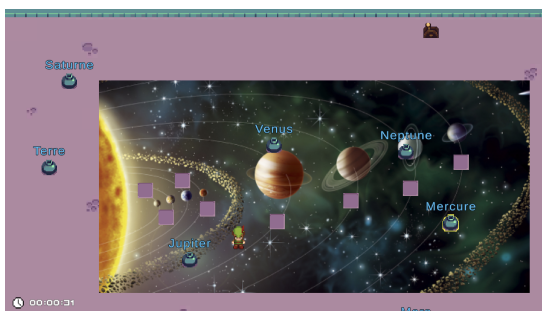
Listing 9.4 – Example of `generateQuestionableFactsOf` implementation for the ORDER PLANETS task

```

1  @Override
2  protected AQuestionableFact generateQuestionableFactOf(ATask task, List<AbstractFact>
   facts) {
3      QFOrderPlanets qf = new QFOrderPlanetsImpl();
4      qf.setID(taskID+"-QAFAC"+factsCounter); factsCounter++; int i = 1;
5      for(SolarSystemPlanet fact: orderFactsAscendingly(facts)) {
6          PlanetOrder solution = new PlanetOrderImpl();
7          solution.setPlanet(fact.getName());
8          solution.setOrder(i); i++;
9          qf.getSolutions().add(solution);
10         qf.getFacts().add(fact);
11     }
12     return qf;
13 }
14
15 private List<SolarSystemPlanet> orderFactsAscendingly(List<AbstractFact> facts) {
16     List<SolarSystemPlanet> sfacts = new ArrayList<>();
17     for(AbstractFact fact: facts) {
18         sfacts.add((SolarSystemPlanet) fact);
19     }
20     Collections.sort(sfacts, (o1, o2) ->
21         (((Long) o1.getSunDistance()).compareTo((Long) o2.getSunDistance())));
22     return sfacts;
23 }

```

Finally, the *game engine* has been completed to add *prefabs* allowing the interpretation of pictures, i.e., map of orbits. Figure 9.13 displays interpreted dungeon rooms presenting examples of gameplay for each solar system training task.



(a) IDENTIFY PLANETS



(b) ORDER PLANETS

Figure 9.13 – Examples of gameplays for solar system facts training tasks interpreted by the *game engine*

9.5 Discussion

Throughout this section, various extensions/applications of the framework to several didactic domains have been presented. These extensions represent proofs-of-concepts, of the genericness of the general approach and the tool that implements it, i.e., in particular thanks to the use of MDE, which guarantees the conformity and the coherence between the conceptual and software approaches. In addition, the training models defined with mathematics teachers provide a means to evaluate the ability to consider different teachers' viewpoints on training for different learners.

Even though the process of creating extensions has been simplified and reduced to the design and implementation of elements related to the didactic domain, it still requires considerable skills, particularly in the capacity to understand and get to grips the existing elements in order to exploit them. Part of the difficulty is inherent to the MDE context, which requires an understanding of the principle and interest of metamodels and models: what they represent in the environment/what they are used for and how they are used/how to use them. Another difficulty lies in the ability to grasp the concepts captured in the existing models/metamodels in order to manipulate, extend or use them (e.g., type of training task, abstract/raw fact, questionable fact). Even though the extension guide attempts to reduce this complexity, it cannot completely remove it.

Another observation concerns the close relationship between the raw facts and the types of training task. Indeed, in the case where several forms of raw facts are modelled (i.e., history-geography, judo), it can be noticed that each type of raw fact corresponds to one or more specific tasks. In our examples, there is a correlation between the form of the facts and the way in which they are questioned. This information has an impact on the modelling of training paths, since an objective questioning a specific task must therefore target facts that are compatible with that task, otherwise no questions can be generated for that task. As an example, the GEOGRAPHY LEGEND FACTS are typically related to the task of LEGENDING A MAP and no other. Since this constraint (i.e., each objective contains the facts corresponding to the tasks present in their levels) depends on the didactic domain and its modelling, it cannot be verified by the algorithm. However, it would be possible to verify this constraint through model validation rules defined specifically in relation to the didactic domain, i.e., possibility of adding them after creating an extension.

Furthermore, through these different extensions, it is possible to observe that some elements are specified exactly the same way, e.g., the mathematics completion 1 task (i.e., *MTCompletion1*) and the solar system planets identification task (i.e., *IdentifyPlanets*). Therefore, it seems possible to factorise some elements in order to define an additional layer of abstraction. More precisely, subclasses of our task types (e.g. *CompletionTask*) could be defined to represent different variants of our generic classes. For example, a subclass of *CompletionTask* could be created to represent tasks where only one element is missing. Hence, *MTCompletion1* and *IdentifyPlanets* would extend this subclass.

A further discussion point regards the current generation algorithm. The latter is based on certain random pedagogical choices, e.g., selection of the objective/level to work on.

An interesting improvement would be to implement a sort of “template-method” design pattern for specifying pedagogical strategies, as defined by [Melero *et al.* \(2016\)](#). This would allow the teacher’s viewpoint on training to be considered further.

Finally, declarative knowledge is theoretical knowledge stated in the form of declarations or propositions including facts, rules, laws, etc. Facts are objective/verifiable information considered as basic knowledge. In our context, the term fact has been used interchangeably with declarative knowledge. However, in retrospect, this view seems rather simplistic, as the proposed model allows declarative knowledge to be considered in a broader sense. Indeed, it would be possible to model sentences to present definitions (e.g., two strings of characters, one for the word, the other for the definition) or laws (e.g., two strings of characters, a number of laws, and a text) or even to model verbs to be conjugated and question them through sentences. Accordingly, it appears that using the term fact is reductive and that another term would have been more appropriate to describe the knowledge being used.

This chapter has presented several extensions of the framework that represent proof-of-concepts of the properties of the framework. The following chapter will describe the tests carried out to evaluate the properties associated to the generators produced by the framework.

TESTS AND VALIDATION OF THE FRAMEWORK

Contents

10.1 Framework Properties Evaluation through Tests	148
10.1.1 Learner Adaptation of the Generated Activities	149
10.1.2 Player Adaptation of the Generated Activities	152
10.1.3 Variety of the Generated Activities	153
10.2 Validation of Static Properties of Models	156
10.3 Framework Evaluation with an Engineer	157
10.4 Use of a Generator in Ecological Conditions	160
10.5 Synthesis	161

In the previous chapter, the ability of the framework to produce generators for different didactic domains (i.e., genericness) and to consider the teacher’s viewpoint on training by allowing the definition of training paths supplied by different teachers have been evaluated. As already mentioned, the aim of the evaluation is to verify the ability of the framework to produce generators respecting specific constraints.

The produced generators are underpinned by three previously defined properties (see Section 5.1):

- GP1) generated activities must be adapted to the learner’s level and results in their training path;
- GP2) generated activities must be adapted to the player’s game preferences;
- GP3) generated activities must be varied in terms of both education and game elements.

The assessment of these properties entails verifying that the generated activities are consistent with the training path and the learner’s progress in this path, that game preferences are respected (i.e., items purchased/activated) and that there is a variety of elements in the dungeon rooms.

In addition, other test methods can be used to evaluate the framework, such as:

- model validation, i.e., testing that the models satisfy the semantic constraints not captured in the metamodels;
- experimentation with engineers, unfamiliar with the framework, aiming at the creation of an extension of the framework;
- proof-of-concept involving the use of generators in games under ecological conditions.

Although the design choices linked to the game (i.e., purchase/activation approach)

or the effective retention of learners could be evaluated, this evaluation requires the development of a game (i.e., interface linking the player to the generated activities). As the design of concrete games are not our research object, **such evaluation is outside the scope of this thesis.**

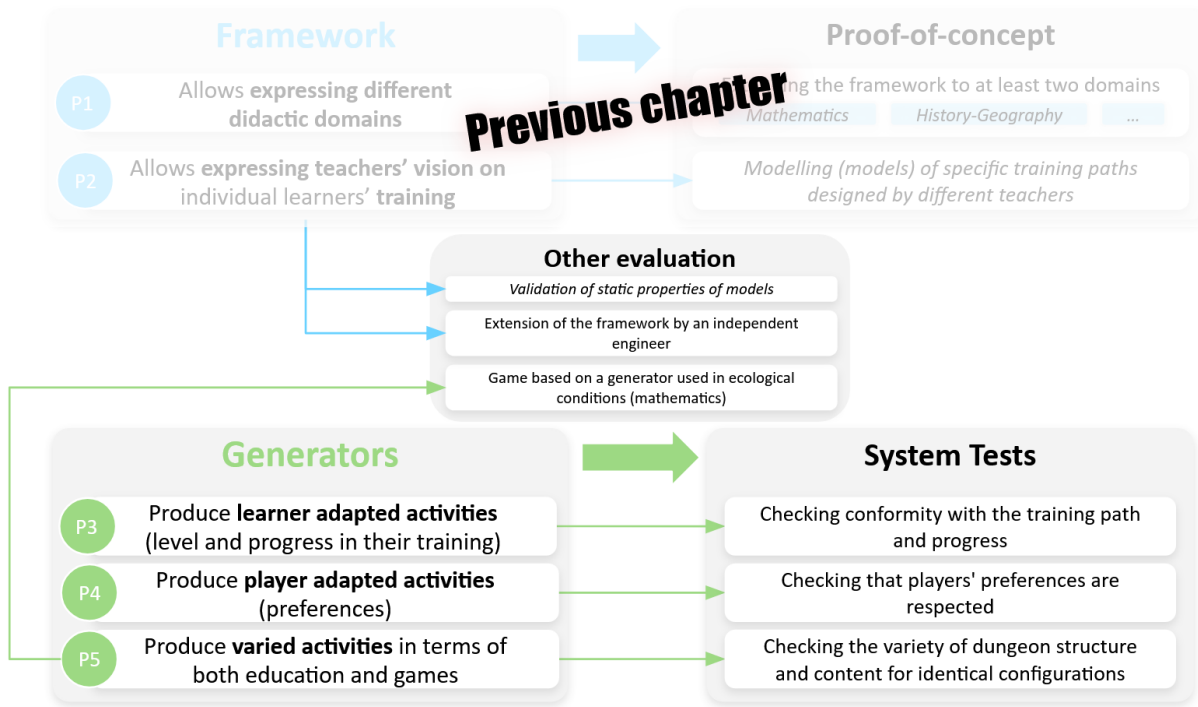


Figure 10.1 – Overview of the overall evaluation of the framework

This chapter aims to present the evaluation of the properties of the produced generators, as well as deepening the evaluation of the framework (see Figure 10.1). First, we present the system tests implemented for the three properties of the generators. Then, we describe the model validation rules implemented to verify the input models supplied to the generators. Next, we detail an experiment carried out with an engineer to extend the framework. Finally, we present a game for multiplication tables training, used in ecological conditions.

10.1 Framework Properties Evaluation through Tests

Generators are designed to be specific to a particular didactic domain. In our context, the system tests have been performed using the mathematics generator. However, as the algorithm is generic (i.e., only the generation of questions on facts is domain-specific), the results can be generalised (e.g., the objective selection algorithm does not change from one domain to another).

10.1.1 Learner Adaptation of the Generated Activities

Learner adaptation is based on the respect of the training path defined by their teacher and on their progress in this path. Evaluating such adaptation involves verifying that the generated activities are consistent with the predictions that can be deduced from an analysis of the learner’s training path. Activity consistency is based on two levels of analysis:

- A) the selection of the objective/level pair to work on;
- B) the tasks present in the dungeon according to the selected level, the learner’s progress and the desired percentage of appearance for each task.

A) Selection of the objective/level pair. According to each learner’s training path and their progress in that path, the generation algorithm must select an objective/level pair among those eligible (i.e., an objective/level pair whose prerequisites have been reached). In order to evaluate the objective/level pair selection algorithm, various edge cases have to be verified, particularly:

- that the objective/level pair selected for the dungeon is eligible;
- that all eligible pairs are selected at least once at some point;
- that none of the ineligible pairs are selected;
- that levels whose percentage of questions about facts encountered and percentage of successes have been reached are considered ineligible and therefore are never selected.

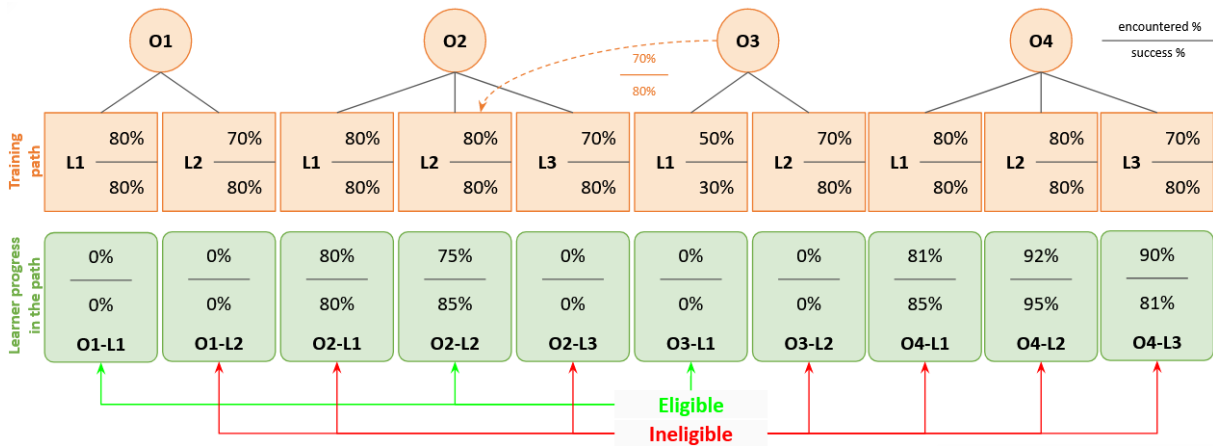


Figure 10.2 – Training path and learner’s progress used for testing the objective/level pair selection

In order to evaluate this aspect of the dungeon generation algorithm, a fictive training path and a learner-player model which have been defined to implement these edge cases are illustrated in Figure 10.2 (i.e., in orange the training path, and in green the learner progress). The proposed training path is composed of four objectives, either having two or three progressive levels (i.e., L_i with $i \in [1, 3]$). Only the third objective (O_3) has a

prerequisite that is achieved as the expected percentages of encountered questions about facts and success have been reached by the learner. Additionally, as the learner reached the expected percentage of encountered questions about facts and success of $O_2 - L_1$, $O_4 - L_1$, $O_4 - L_2$, $O_4 - L_3$ these levels are ineligible, so is the objective O_4 . However, for O_1 , O_3 the first level is eligible whereas for O_2 the second level is eligible. Any other level is considered ineligible.

On the basis of these test models, a test method has been created, in Java using the JUnit framework¹, for each of the edge cases, namely:

- for one generated dungeon, the objective/level pair selected is eligible (i.e., belongs to $[O_1 - L_1, O_2 - L_2, O_3 - L_1]$);
- any eligible objective/level pair (i.e., $O_1 - L_1, O_2 - L_2, O_3 - L_1$) are selected at least once over 150 generated dungeons;
- none of the unstarted and ineligible objective/level pairs (i.e., $O_1 - L_2, O_2 - L_3, O_3 - L_2$) are selected over 150 generated dungeons;
- $O_2 - L_1, O_4 - L_1, O_4 - L_2, O_4 - L_3$ whose percentages of questions about facts encountered and successes are achieved are considered ineligible, and do not appear in any dungeons over 150 generations.

B) Task allocation. The generation algorithm must allocate tasks in the dungeons according to:

1. their percentage of appearance;
2. the number of rooms in the dungeon based on the player's progress in the game (i.e., the player's current game level);
3. the achievement of the task (i.e., a task with a percentage of encountered questions about facts and success to 100% are considered completed).

For instance, a task with a percentage of appearance of 20%, not achieved by the learner, must appear in 2 rooms of a dungeon requiring 10 rooms with questions. It should be noted that when the size of a dungeon and the percentage of appearance of a task are small (e.g., five rooms with questions and 5% of appearance), the task may not appear in the dungeon (i.e., favouritism of the tasks with a higher percentage of appearance). Therefore, testing should be carried out on two different game levels (i.e., different dungeon sizes): one with few rooms with questions, another with many. Presently, when a task is completed, the algorithm distributes its percentage of appearance proportionally to the percentages of appearance of the remaining tasks, e.g., given T1[50%], T2[20%], T3[20%], T4[10%] where T1 is completed, then the percentages of appearance the other tasks become T2[40%], T3[40%], T4[20%].

Depending on the learner's progress, four edge cases must be tested in order to evaluate the correct allocation of tasks in the dungeons:

- when the learner has not started the level, progress (i.e., success and encountered questions about facts) is at 0% for all the tasks of the level;
- when the learner has started the level, but none of the tasks are completed, progress

1. <https://junit.org/junit5/>

- is greater than 0% but less than 100%;
- when the learner has achieved a single task of the level, progress of that task reaches 100% while the others are below 100%;
- when the learner has completed all but one of the tasks of the level, progress of these tasks is at 100% except for one.

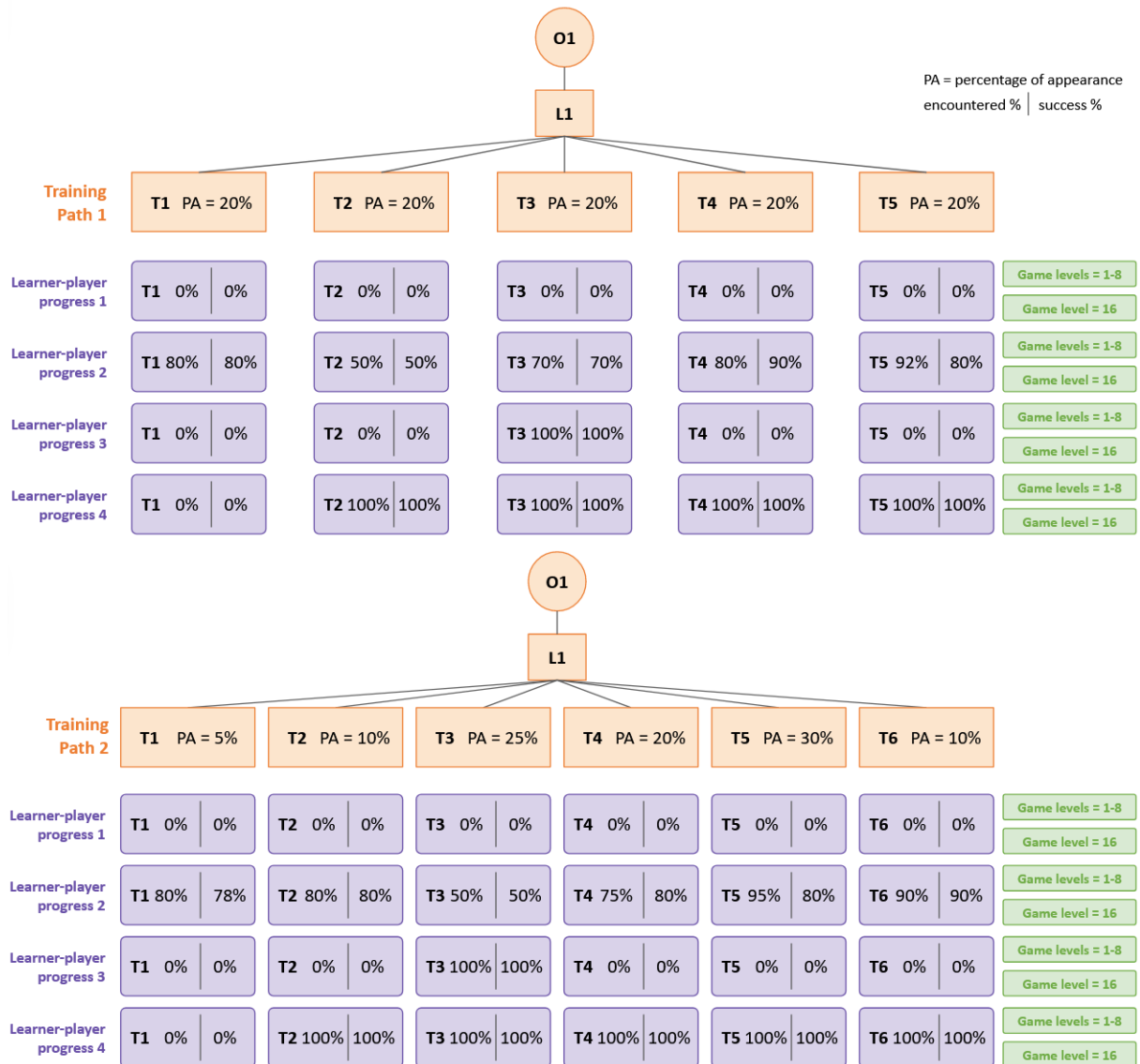


Figure 10.3 – Training paths and learner-player’s progress used for testing the allocation of the tasks

In order to evaluate the allocation of the tasks, two training paths for testing have been defined. The first one consists of five tasks with an identical percentage of appearance (20%), i.e., none of the tasks predominates over the others. The second consists of six tasks

having different percentages of appearance, i.e., some tasks are predominant. Evaluating the allocation of tasks on the basis of these two training paths seems sufficient, as it covers both case scenarios: equal allocation of tasks and unequal allocation of tasks. For both paths, the four edge cases must be evaluated over two different dungeon sizes. Therefore, four learner-player models, one for each edge case, have to be created for each training path. Additionally, the repartition of tasks has to be tested on linear and labyrinthine dungeons². The minimal game level for having labyrinthine dungeons (i.e., in our game context) is the level 8. Based on our defined game progression, the level 8 provides 12 rooms with questions. Therefore, tests on labyrinthine dungeons are performed for each learner-player model at the game level 8 whereas tests for linear dungeons are performed at the game level 1, which provides 5 rooms with questions. In order to try the task repartition on bigger dungeon size, tests are performed for both linear and labyrinthine dungeons at the game level 16, which provides 20 rooms with questions. Therefore, variants of the four learner-player models (i.e., for both training paths) have to be created considering different game levels: 1, 8, and 16. Figure 10.3 illustrates both training paths and the several learner-player models used for testing.

Afterwards, for each edge case, a test method verifying that for each learner-player model, a generated linear or labyrinthine dungeon has the right number of rooms per task has been implemented. For instance, in the case of the first training path and the first learner-player model (i.e., zero progression), and given a linear dungeon (i.e., 5 rooms with a question), the test verifies that the dungeon contains one room with T1, one with T2, one with T3, one with T4, and one with T5.

10.1.2 Player Adaptation of the Generated Activities

Similarly to the evaluation of adaptation to the learner, the evaluation of adaptation to the player consists of verifying that the generated activities are conforming to the deductible predictions from an analysis of the game preferences associated with the player. In our context, the player's game preferences are described in terms of purchasable and activatable equipments. These equipments unlock abilities and therefore new gameplays. A player can activate/deactivate an equipment item, thus locking/unlocking the associated abilities and gameplays. Naturally, default gameplays are defined and cannot be deactivated, otherwise the activities could not be correctly generated (i.e., no gameplay can be found for a given task because all the compatible gameplays have been deactivated).

Several edge cases need to be tested in order to evaluate the adaptation of activities according to the player's game preferences:

- the learner-player has made no purchase, therefore none of the gameplays featuring an ability locked by an equipment are present in the dungeons (i.e., only those with default abilities such as MOVABLE);
- the learner-player has purchased and activated all possible equipments, therefore all the abilities locked by an equipment must appear at least once in a dungeon (i.e., a

2. A specific parameter for tests can be used to force the generation of labyrinthine dungeons.

minimum of one gameplay featuring each ability);

- the learner-player has purchased and activated some equipments, the abilities associated with these equipments appear at least once in a dungeon (i.e., a minimum of one gameplay featuring each ability) while the others never appear;
- the learner-player has purchased but not activated all possible equipments, none of the gameplays containing an equipment-locked ability are present in the dungeons (i.e., only those with default abilities such as MOVABLE).

In order to evaluate the adaptation of activities to players, these edge cases have been simulated in learner-player models featuring different game preferences. Figure 10.4 shows the four learner-player game preferences models, and the game model in terms of equipments locking abilities used for testing. The game model (i.e., equipment and abilities) used is the one specified for the *game engine* allowing the interpretation of dungeons. Seven abilities are available, locked by six equipments. Learner-player preferences describe the purchase/activation of equipments.

Game equipments	Belt BREAKABLE	Glove ROTABLE	Bracelet PUSHABLE	Keys OPENABLE FOLLOWABLE	Coat CROSSABLE	Carrot CATCHABLE
	Purchased Activated	Purchased Activated	Purchased Activated	Purchased Activated	Purchased Activated	Purchased Activated
Learner-player preferences 1	✗ ✗	✗ ✗	✗ ✗	✗ ✗	✗ ✗	✗ ✗
Learner-player preferences 2	✓ ✓	✓ ✓	✓ ✓	✓ ✓	✓ ✓	✓ ✓
Learner-player preferences 3	✗ ✗	✗ ✗	✗ ✗	✓ ✗	✗ ✗	✓ ✓
Learner-player preferences 4	✓ ✗	✓ ✗	✓ ✗	✓ ✗	✓ ✗	✓ ✗

Figure 10.4 – Player’s game preferences for testing based on the equipments unlocking abilities used in our *game engine*

For each edge case (i.e., each learner-player model with the preferences illustrated in Figure 10.4), a test method has been created to ensure that for up to 100 generated dungeons, the conditions are always met. Similarly to the evaluation of the adaptation to learners, these test methods have been implemented in Java using the JUnit5 framework.

10.1.3 Variety of the Generated Activities

Dungeon variety is based on both the training and the game content. Many aspects of dungeons can be varied (e.g., general shape of dungeons, questions, elements and their position, shape of rooms, gameplays). In our context, the variety in terms of training depends on the path defined by the teacher. However, the evaluation of the compliance of the generated activities with the training paths has already been presented in Sec-

tion 10.1. As a result, our main interest lies in the variety of activities in terms of: A) the structure/layout of the dungeons and B) the elements contained in the dungeons.

A) Structure/layout of dungeons. *Roguelite* uses procedural generation to build dungeons that are unique, i.e., the arrangement of the rooms and the positions of the objects are organised differently each time. The game engine developed as part of the *AdapTABLES* project, besides allowing the interpretation and play of generated dungeons, provides a map-based visualisation of these dungeons without requiring any exploration. Maps display each room in a different colour depending on the task to be performed, and small lines indicate access between rooms (i.e., to differentiate between linear and labyrinthine dungeons). Accordingly, an initial evaluation consisted of a manual verification of the layout of the dungeons using the map-visualisation offered by the game engine. More precisely, the idea was to generate several dungeons for the same game level and the same learner-player and to compare their maps. Figure 10.5 displays four maps of generated dungeons (i.e., three labyrinthine dungeons and a linear dungeon) for the same objective/level and the same learner-player (i.e., with unchanged progression). It can be observed that these dungeons have different structures and are differently distributed.

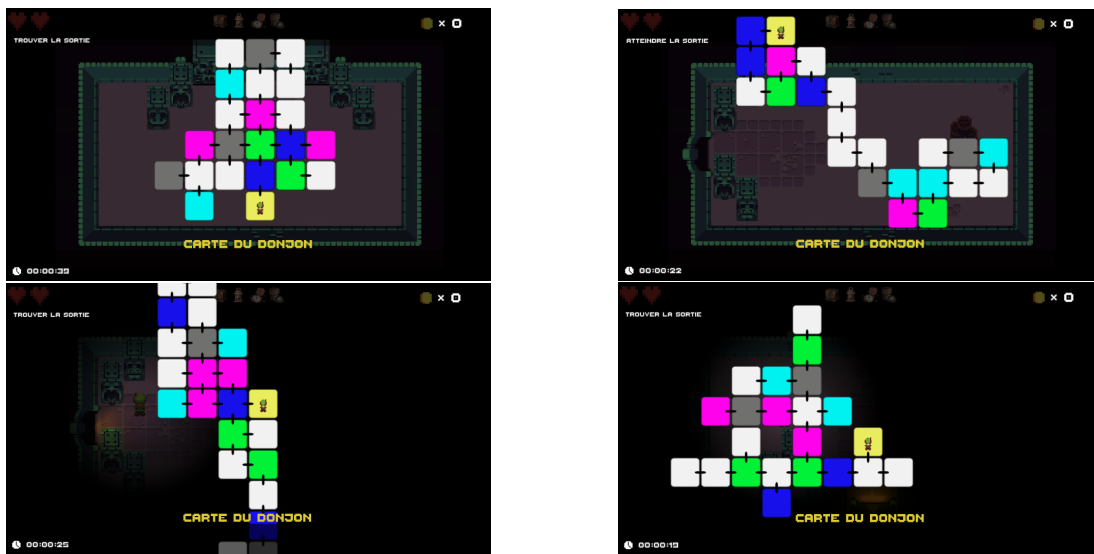


Figure 10.5 – Maps of four generated dungeons for the same objective/level pair and learner-player

B) Elements of the dungeons. The elements of the dungeons can vary at different levels:

1. the gameplays selected for each task;
2. the elements selected for each gameplay;
3. the selected room types (i.e., *RoomType*);
4. the position of the elements in the rooms;

5. the questions and the wrong choice of questions about facts.

First, the variety of room types and element positions depends on the modelling done in the game model (i.e., *RoomType* specification, the more room type variants are created, the more variety there is). Second, the variety of questions depends on the training path specified by the teacher. Regarding the variety of wrong choices for the questioned facts, it depends on the targeted didactic domain since the method created to generate these wrong choices is part of the extension (i.e., it can also depend on the “*WrongData*” modelled in the knowledge model). Furthermore, the variety of gameplays selected for each task depends on two factors:

1. the variety of gameplays described in the game model;
2. the purchases (i.e., game preferences) made by the player.

For each of the abilities (i.e., default and lockable abilities), different gameplays and gameplay variants have been defined (e.g., a variant of pushing elements to the right is pushing elements to the left) so that each task appears with at least two different gameplays or two gameplay variants. This aspect (i.e., selection of at least two different gameplays for each task out of 100 dungeon generations) has been evaluated for:

- a learner-player having made no purchases;
- a learner-player having purchased and activated everything.

Finally, the variety of elements for each gameplay mainly depends on the variety of game elements described in the game model. For example, let’s take an ability called *CATCHABLE* for which there are two associated game elements: a rabbit and a cow. In this case, the gameplay related to the ability *CATCHABLE* should appear in dungeons (i.e., not necessarily the same dungeon) with both game elements different (see Figure 10.6). This aspect has been formally evaluated by means of a test method, that has been implemented using the Junit5 framework (see Appendix F), verifying that for each ability having several compatible game elements (i.e., in the game model), each of these game elements is selected at least once after an undefined number of dungeon generations.

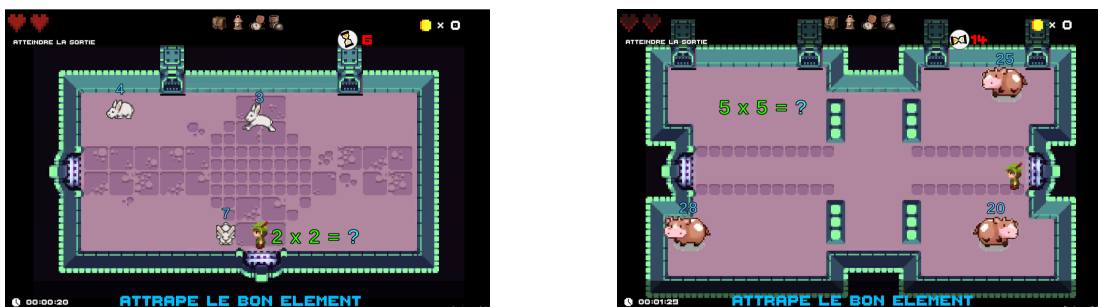


Figure 10.6 – Example of two variants of the same gameplay

10.2 Validation of Static Properties of Models

A further step in verifying the framework involved to validate the models used by the generators. Although a great deal of information is captured by the metamodels, they do not capture all the semantic properties required for a given domain. For instance, in our current training model (see Figure 8.3) nothing prevents the definition of an objective whose prerequisite is its own level. This makes no semantic sense. The validation of static properties of models is a semiformal method for the automatic verification of models, which consists of defining a set of constraints and validating or not the models according to these constraints. In order to guarantee the correct semantics of the (generic) models supplied as input to the generators, we have defined a set of semantic constraints on the models using *Epsilon Validation Language* (EVL).

In our context, the validation of static properties applies mainly to the two following models: the training model (i.e., provided as input to the generator) and the activity model (i.e., provided as output of the generator). Other models are not targeted, as we did not find any semantic constraints not captured. For the training model, the following rules have been implemented:

- the facts of an objective belong to the knowledge associated with the training path (Figure 10.7 illustrates this rule in EVL);
- the level (i.e., *requiredLevel*) of a prerequisite of an objective O, is not a level of the objective O;
- the percentages of success and encountered questions about facts (i.e., *successPercent* and *encountersPercent*) of the prerequisites and completion criteria have values between 0 and 100);
- the percentages of success and encountered questions about facts of the prerequisites must be less than or equal to the percentages for the completion criteria (i.e., else it is not achievable);
- the number of bad choices chosen in a multiple choice response modality is inferior to the total number of choices.

```

/*
Checks if every SetOfFacts of an Objective, belongs to the
corresponding Knowledge of the objective's path.
*/
constraint objectiveFactsBelongsToPathKnowledge {
  check {
    return self.objectives.forAll(obj |
      obj.setofacts.forAll(facts |
        self.knowledge.knowledgefacts.contains(facts)));
  }

  message {
    return "SetOfFacts does not belong to the correct knowledge";
  }
}

```

Figure 10.7 – An example of model validation rule written in EVL that verifies whether the facts associated to an objective belong to the knowledge model associated to the path

For the dungeon model, the following rules have been implemented:

- the chosen objective belongs to the learner-player path;
- the chosen level belongs to the chosen objective;
- each task present in the dungeon rooms belongs to the chosen level;
- the position of the elements of the room belongs to the room type of the room or to a composite element.

Appendix G present the full EVL source code that implements the semantic constraints for both the training model and the activity model. The validation of models has not been automated, therefore it must be run manually by an engineer when necessary. However, the execution of the model validation code did not indicate any problems on the tested models.

10.3 Framework Evaluation with an Engineer

Another step to evaluate the framework consisted of conducting an experiment with an engineer (see Figure 10.8). This experiment had two main objectives:

1. evaluating the usability of the framework, i.e., can the framework be extended by an engineer not involved in the project?
2. improving the clarity of the provided guidelines, i.e., are the guidelines self-sufficient for creating an extension.

For the experiment, the engineer had to extend the framework to the solar system domain presented in Section 9.4 by creating the necessary components (i.e., metamodel, models, generators of questions about facts). Several resources were available:

- a printed and numerical version of the extension guidelines document;
- an MDE expert answering their questions and guiding the uses of the EMF plugin, as the engineer had zero notions of MDE;
- a document presenting the specification of the raw fact (i.e., planets) and solar system training task to implement.

Since an engineer is not an expert in didactic or pedagogy, our choice has been to provide the specification of data related to the didactic domain to the engineer. The experiment has been realised over a day and a half. Encountered problems with the guidelines or incomprehension have been discussed directly.

Once the extension had been realised, the engineer had to answer a short questionnaire presented in Appendix H. The questionnaire is composed of:

- three general questions regarding the difficulty of extension, MDE, and the extension guide, namely:
 - did you find the guide self-sufficient to allow the extension of the framework?
 - do you think that a strong expertise in MDE is required for extending the framework?
 - on a scale from really easy to very hard, how do you estimate the difficulty of extending the framework?
- a grid asking to evaluate, on a scale from really easy to very hard, every step of

the extension (i.e., comprehension of the instructions/domain/framework, creations of the metamodels, models, generators of questions about facts, integration of the created to the existing one, debugging).

- a comment box.



Figure 10.8 – Experimentation of the framework with an engineer

Overall, the engineer found the extension to be well guided and quite easy to realise. Moreover, the engineer considered that a strong expertise in MDE is not required for extending the framework. Therefore, it would appear that the only requirement is the ability to manipulate the EMF framework and understand the basic mechanism of a model being an ‘instance’ of a metamodel. In addition, some necessary clarifications and improvements for the extension guidelines have been highlighted during the experiment, notably for the creation of generators of questions about facts and the debugging (i.e., understanding and correcting the errors created from the added code) parts. The engineer had trouble understanding how the generators of questions about facts work and what they did. There are two probable reasons for that: 1) the short amount of time for discovering the framework (i.e., the experiment lasted over a day and a half only), and 2) the lack of diagrams explaining the functioning of the components (i.e., main code and extensions). In order to improve that aspect, a diagram has been added to the extension guidelines. Furthermore, the debugging part has been considered quite hard by the engineer. In order to reduce the difficulty for this part, several possible “common” errors and their solutions or possible solutions have been identified and added to the extension guidelines (i.e., errors we made as well as the ones made during the experiment by the expert).

Note that although the extension, particularly in terms of the metamodel produced by the engineer, is similar to our modelling (see Section 9.4), some differences are noticeable, such as the different names used and the choice of certain types of element, e.g. in our case, the distance to the sun is represented as a Long, but the engineer has chosen to represent it as a Float. This demonstrates that modelling is an activity that involves a level of subjectivity.

Additionally, an extension has been realised by a researcher and engineer familiar with the framework and MDE in less formal conditions. The expert specified a completion task for a different domain: Spanish. The task consisted of completing sentences with verbs conjugated in the present subjunctive in Spanish. Then the expert modelled the task by following the guide and asking me questions in case there was any problem. Figures 10.9 and 10.10 present the metamodel and knowledge model created.

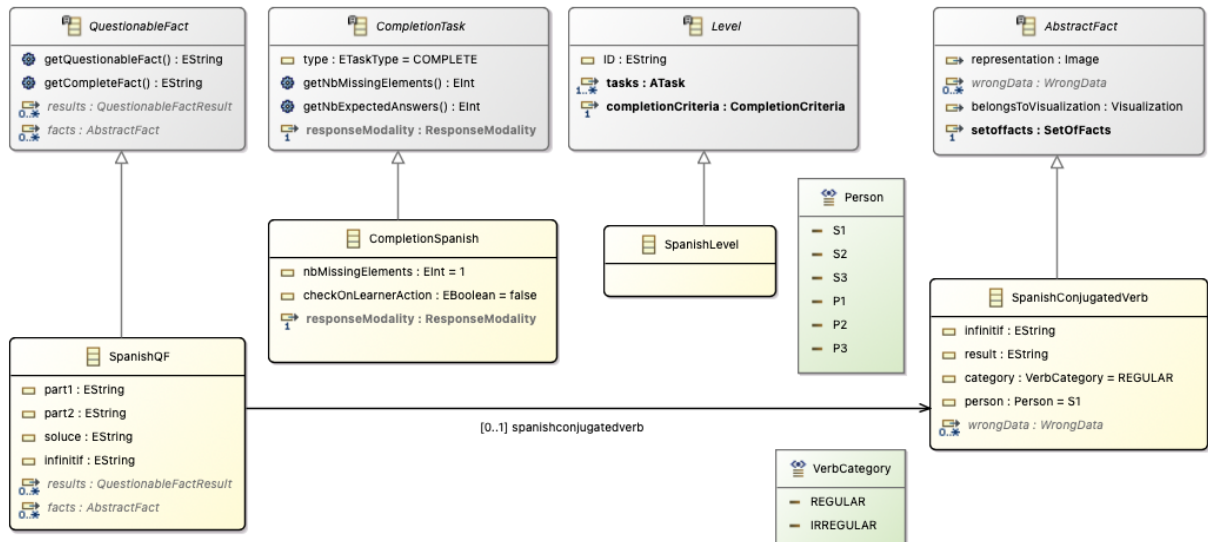


Figure 10.9 – Spanish metamodel created by an engineer familiar with the framework

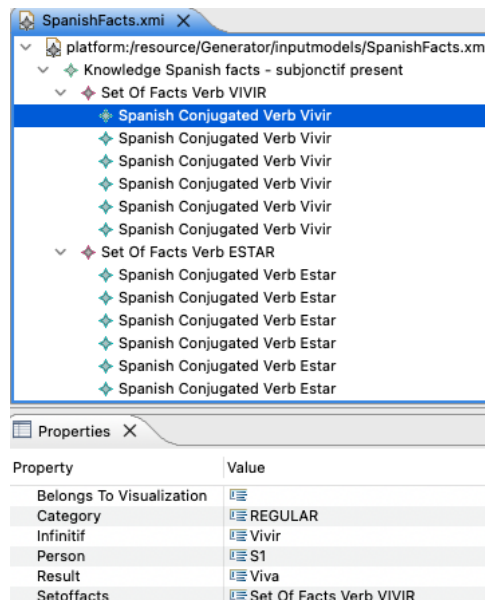


Figure 10.10 – Spanish knowledge model created by an engineer familiar with the framework

In conclusion, the proposed framework has been extended by two engineers, one external to the research work and one internal. The external engineer followed the extension guide to implement a generator for the solar system domain that had previously been implemented in order to verify the feasibility of the requested task. The internal engineer extended the framework to a previously unimplemented and undefined domain, Spanish verbs. Overall, the extension mechanism has been deemed as moderately complex, particularly regarding the comprehension of the framework in the broader sense.

10.4 Use of a Generator in Ecological Conditions

In the context of *AdapTABLES*, an educational game for training on multiplication tables has been developed. This game uses the mathematics generator (see Section 9.1) produced by our framework³. This game has been used at The Science Festival 2023, known as “la fête de la science” in France, which is an annual event aimed at promoting scientific knowledge and discovery among the public. Additionally, it has also been used in ecological conditions with a classroom of 11 students of the 2nd grade (see Figure 10.11). The objective was to collect informal feedbacks and gather potential bugs or issues about the game or the generator.



Figure 10.11 – Use of the *AdapTABLES* in ecological conditions

It is important to note that the design of a game is not necessary for the evaluation of a generator, and that these tests have been carried out as part of the project. However, these tests are particularly interesting as they have highlighted a problem of pedagogical alignment related to the response time given to the learner-players. Depending on the gameplay, this time should be adjusted. For this purpose, experiments could be carried out with learners. Learners should be classified according to their gaming experience.

3. Data exchanges are done through a REST API.

If they are used to playing games, manipulating a computer or a controller, this could explain a faster response time than people who are not used to manipulating computer tools. As a result, gameplays could be proposed to them and their response times recorded. Then, a formula (e.g., the average of the members of a regular/non-regular player group) would determine the response time to be associated with each specific gameplay. Finally, a parameter could be added to the learner model to adapt these times to them.

10.5 Synthesis

In conclusion, the tests have been carried out using the generator dedicated to multiplication tables. However, the use of a domain-specific generator has not invalidated the validation of the properties, since the algorithm for selecting an objective/level pair of a training path or for allocating tasks is independent of the targeted didactic domain, i.e., the creation of an extension by adding domain-specific tasks does not influence in the way in which the level of an objective is selected or the way in which tasks are distributed in a dungeon. Furthermore, the evaluation of the variety of elements in the dungeons is also validated despite the unique use of the mathematics generator, since the algorithm for generating the dungeons (i.e., training tasks translated into gameplays) is also generic. It should be noted that in order to use another generator, all the learner tests would have to be rewritten, whereas those for the player would be reusable with learner-player models having a training path related to the correct didactic domain. Moreover, it is important to note that the tests regarding adaptation to player preferences are based on the design choices made (i.e., purchase/activation mechanism) and that any other choice would require other tests.

CONCLUSION

Contents

11.1 Synthesis	163
11.1.1 Contributions to TEL Research Domain	165
11.1.2 Limitations	166
11.2 Perspectives	166

This last chapter provides an overview of the work accomplished during this PhD thesis, as well as its limitations. Then, several more or less coarse-grained perspectives related to this work are introduced.

11.1 Synthesis

The research presented in this thesis falls within the field of TEL (Technology-Enhanced Learning). More precisely, it presents an exploratory work that addresses a research in engineering problem by attempting to characterise and guide the design of activity generators adapted to learner-players in the context of training games. Due to the shortcomings observed in the state of the art (see Chapters 2 and 3), our interest has been focused on two objectives: 1) to propose an approach that abstracts as many elements as possible to provide reusable elements for the creation of one domain-specific generator to another, and 2) to take into account the adaptation of both the educational and game dimensions. Activity generation is a wide subject, therefore to reduce the scope of our research our focus has been on declarative knowledge training. Therefore, our interest has been on *how to guide the design and implementation of generators of learner-player adapted and varied game activities for declarative knowledge training* (i.e., research problem, see Chapter 4).

Declarative knowledge training requires repetition for its retention, and variety to reduce the boredom caused by repetition. In order to create game activities for training purposes, game design choices have to be made, such as selecting a compatible game genre. *Roguelite* are often dungeon-like games that provide repetition, variety, and a sense of progress. As a means of satisfying the training requirements, *Roguelite* has been shown to be a suitable game genre based on a repetitive mechanic that offers varied and generated game levels. Accordingly, the scope of our research has been narrowed down to the generation of declarative knowledge training activities in the context of *Roguelite* games (see Chapter 4).

To answer our first research question – **How to propose an approach sufficiently**

generic to consider declarative knowledge independently of a specific didactic domain? – our proposal consisted in modelling concepts at a higher level of abstraction (e.g., categories of gameplays, types of tasks) in order to provide (see Section 5.1):

1. an algorithm for generating activities that is independent of any didactic domain, i.e., the main part of the code does not have to be modified when changing the didactic domain.
2. an extension mechanism allowing the specification of domain-specific data that will be used, through generic concepts, by the algorithm.

Regarding our second research question – **What is a learner-player adapted and varied game training activity?** *Which educational and game elements constitute these activities? How to combine the game and educational elements coherently?* – our proposal has been to use activity theory to define the structure of our training game activities (see Chapter 5). Then, each training and game element composing an activity has been specified at a fine-grained level (see Chapter 5). Additionally, a bi-dimensional analysis framework guiding the specification of Roguelite games for educational purposes has been developed (Lemoine *et al.* 2023a; 2024c), in the context of the AdapTABLES project, which helped us in making the game design choices required for the generators (see Section 5.4.1). Finally, a relatively well-known research question in serious games design, that is also present when designing serious games activities, is the problem of alignment between game and educational elements. In order to tackle this problem, our proposal is a systematic method based on the use of numerical questionnaire formats as a pivot for specifying machine-readable relationships between training task types and gameplay categories (see Chapter 6) (Lemoine *et al.* 2023b; 2024b).

To answer our third question – **How to structure the required data and their relations in order to drive the generation of coherent activities?** – our proposal consists of a conceptual modelling approach (Lemoine and Laforcade 2023a). Six interconnected models are at the centre of our solution (i.e., the knowledge model, the training model, the game model, the relation model, the learner-player model, and the activity model). These models have been created and thought to be extensible, i.e., use of abstract concepts that can be extended, though the addition of sub-concepts, to domain specific information in order to be generically manipulated by the algorithm (see Chapter 7). Because of our genericness need, the mapping problem addressed before also appeared at a lower level: between questions about facts and gameplay elements. In order to solve that issue, our proposal is a generic modelling of questions about facts and a set of parameters in the game model (i.e., in the description of gameplays) (Lemoine and Laforcade 2023b), allowing the creation of a generic algorithm to instantiate task-oriented gameplays (see Section 7.2).

Finally, regarding our last research question – **How to specify every information required for generation to enable computer interpretation for the development of activity generators?** – our proposal consists in using EMF to develop a tool (i.e., framework) embedding our conceptual approach that guides and facilitates the creation of activity generators (Lemoine *et al.* 2023c; 2024d). This framework includes metamodels

(i.e., computerised version of our conceptual models), an activity generation algorithm, and an extension mechanism allowing the specification of domain specific information (see Chapter 8).

11.1.1 Contributions to TEL Research Domain

Regarding the knowledge produced, this PhD thesis has enabled the definition of a design approach of activity generators, in particular for declarative knowledge training, by characterising the necessary requirements of a generator. Therefore, this is a contribution to the field of research in engineering of TEL systems. Several general insights can be drawn from this research.

First, the **creation of activities requires the selection of a game genre compatible** with the type of knowledge and the didactic and pedagogical intention targeted.

Second, for **declarative knowledge training** as defined in this thesis, **Roguelite is a theoretically compatible genre** because it offers mechanisms that favour training needs: repetition, variety, and progression.

Furthermore, **it is possible to take into account adaptations on several dimensions for the generation of activities through the separation of concerns** (i.e., approach induced by our MDE context). However, it is important to note that some dimensions may have a slight priority than others, e.g., favouring the training over the game dimension in the event of conflict. As a result, although the concerns are viewed independently, they should not be thought of entirely autonomously.

Moreover, the main knowledge provided by this thesis is that **the automatic creation of activities** (i.e., generation) **for declarative knowledge training can be achieved through a “generic” approach** since it is possible to propose a domain-independent algorithm for generating activities based on an extension mechanism that only provides the knowledge related to the targeted didactic domain. Additionally, this **extension mechanism makes it possible to reduce the effort and guide the specification of the specifics of the didactic domain** under consideration. In particular, by defining educational and game concepts in a generic way, and by specifying the relationships that unite them through a machine-readable model (or other relationships).

Additionally, **activity generation can be seen as a model transformation** having several models as inputs and a single model as output, as defined in MDE.

Furthermore, **MDE is particularly well suited to the specification, manipulation and generation of data because of its principles** (i.e., abstraction, modelling/productivity of models, separation of concerns). It is also **well suited to the creation of a generic approach**, as one of its principles regards the reusability of models (i.e., *capitalisation*).

Finally, the **EMF framework provides a useful theoretical and practical framework for the creation of research prototypes**, limiting the amount of code required to develop such generators.

11.1.2 Limitations

The research conducted throughout this thesis has several limitations.

First, the scope of our research can be seen as the initial limitation of our work, since our contributions are limited to the training of declarative knowledge in the context of *Roguelite* games. Changing the context would require rethinking, redesigning and redeveloping every component, however the process (i.e., choosing a genre, defining the learning path, abstracting game and learning elements, mapping the elements) could probably be followed.

Second, the evaluation of the framework has only been performed by a single engineer independent of our research. In order to improve the extension guidelines and evaluate the usability of the framework, different extensions should be realised by several engineers unfamiliar with the research work.

Third, our proposed framework is based on justified and argued design choices, however these choices have not been evaluated with players. Therefore, in order to evaluate those choices, the use of the game developed in the *AdapTABLES* could be a possibility.

Fourth, our proposal only relies on player's game preferences (i.e., activation/deactivation of bought items) to adapt the activity by restricting the available gameplays. However, multiple type of data (e.g., players' expressions, players' actions/traces, information that players provided) can be used to generate adapted content that improves players' experience (Yannakakis and Togelius 2011).

Fifth, the software infrastructure provided in the framework is based on the use of EMF (i.e., an *eclipse* framework providing tools for MDE use). However, our approach and the use of a dedicated framework demand a minimum of knowledge and expertise in MDE and (meta-)modelling. Additionally, modelling and meta-modelling are subjective activities since they depend on the modeller's interpretation of the domain and the conventions and styles they are accustomed to. Therefore, another modeller could and probably would have modelled the concepts differently. Furthermore, the models and their embedded concepts are influenced by the experts with whom they have been developed. As a result, exchanges with other didactic domain experts could lead to variations in the proposed models.

Finally, our work specified task types and gameplay categories that do not claim to be exhaustive. Even though the proposed concepts have been sufficient for the domains tested, there is no guarantee that they will be sufficient for other domains. However, it should be noted that in the mapping approach (see Chapter 6), our task types have been sufficient to establish a relationship with each exercise extracted from the numerical questionnaire formats. Furthermore, each task type has at least one associated gameplay category (otherwise the approach would not work). Therefore, this can be seen as the first step in assessing the coverage of training task types and gameplay categories.

11.2 Perspectives

The research work carried out throughout this PhD thesis opens numerous prospects.

Additional experimentations. Several experiments should be carried out to deepen the validation already accomplished. First, the experiment with a single engineer should be reproduced with different engineers that are unfamiliar with the framework. The experimentation realised aimed to extend the framework to the domain of solar system facts and covered two training tasks (i.e., ordering and graphical completion). As a result, an experiment with more in-depth instructions, targeting all types of task, should be conducted.

Furthermore, although the design choices have been justified and argued, they have not been evaluated, in particular the purchase/activation mechanism for game preferences. An interesting experiment using the game developed in the context of the **AdapTABLES** project would be to evaluate the design choices with players. For example, the time taken to execute a gameplay actions differs according to the actions to be performed: moving a pot takes longer than opening a chest. As a result, an analysis of the time taken to execute gameplays needs to be conducted to ensure that the educational and game alignment is fine-tuned. For example, the players could be confronted with the same gameplays and depending on their level of knowledge and mastery of computer equipments (e.g., keyboard, joystick, etc.), their completion time on each equipment would be extracted to define several mean values. These mean values could be used to adapt the time allocated to each gameplay of the activity according to the players' level of knowledge and mastery of computer equipments.

Additional adaptations. Currently, the generation algorithm is based on the use of randomness to select the objective/level pair to train on in a dungeon. An interesting prospect would be to define and implement different pedagogical strategies for their selection, in the sense of [Melero *et al.* \(2016\)](#), which teachers could select. For example, the selection of an objective/level to train on could be done by always selecting the same eligible pair for the same run, i.e., during a run, all the dungeons played would allow training on the same objective/level, whereas currently, any eligible pair is selected each time. From this perspective, the following question needs to be addressed: Which are the pedagogical strategies, and how can they be implemented? In order to tackle that question, it is necessary to look at the literature to see if strategies exist and to exchange with teachers in order to define different training strategies. Therefore, the aim would be to define and characterise different strategies so that they can later be implemented. These strategies would be selectable by the teacher in the training path of each learner, and thus changeable at any time. From a coding perspective, these 'strategies' could be seen as heuristics that may or may not be applied.

At the beginning of this PhD thesis, our aim had also been to propose feedback adapted to the training situation (i.e., type of question) and to the learner's level, i.e., taking into account the error made or previous errors. Feedback can be defined as "information about the gap between the actual level and the reference level of a system parameter which is used to alter the gap in some way" ([Ramaprasad 1983](#)). Feedback is intended to reinforce knowledge, but also to guide learners in correcting their mistakes ([Bimba *et al.* 2017](#)). [Shute \(2008\)](#) has produced a thorough review of feedback forms and classified them by

increasing degree of complexity. In the context of declarative knowledge training using *Roguelite*, we have identified four types of feedback that appear to be relevant: Verification, Try again, Response contingent, and Topic contingent. Verification (i.e., “knowledge of the results”) informs the learner of the correctness of their answer. Try again informs the learner of an incorrect answer and allows one or more further attempts. Response contingent describes the reasons why the correct answer is correct or the incorrect answer is incorrect, e.g., Which is the capital of France: Paris or Berlin? Possible feedback (after answer): Berlin is in Germany, so it cannot be the French capital. Topic contingent presents information relative to content, e.g., “As a reminder, $3 \times 5 = 5 \times 3$, this is the commutativity principle” or “As a reminder, $3 \times 5 = 5 + 5 + 5 = 3 + 3 + 3 + 3 + 3$ ”. Therefore, an interesting prospect might be to determine the appropriate feedback for each learner. Adaptive feedback that considers prior knowledge, learning progress, and learning preferences has been shown to be effective within learning environments (Bimba *et al.* 2017). For example, hints could be provided for learners that are considered in difficulty (i.e., determined by an analysis of learners’ previous results, for example), such as giving an answer, giving information about the answer or removing incorrect suggestions. Therefore, the generator would have to manage this new adaptation mechanism and provide adapted feedback based on the question by using information on the learner.

Tooling. The training path is an interesting structure because it is quite modular. However, many parameters need to be set by the engineer to specify a training path. As a result, the creation of training paths is tedious and complex. To reduce the complexity of this task, two solutions are possible:

1. develop authoring tools to make it easier to specify training paths. Under the *AdapTABLES* project, an editor/dashboard has been developed. This tool enables the creation of training paths for mathematics, as well as allows the visualization of any learners’ progress.
2. automatically generate training paths.

The second item is more complex but also more interesting because, as in the case of activity generation, generating training paths removes this task from the hands of the teachers and engineers that have to model them. Generating learning paths has been the subject of several research works (see Chapter 3). Accordingly, one approach may be to define constraints and elements (e.g., knowledge to be worked on, level of difficulty, methods for working on it) to be provided by the teacher, in order to enable the program to generate training paths adapted to the learners. However, an additional difficulty with this problem is that of genericness. Specifying the parameters is great, but is it possible to have parameters that are independent of the didactic domains in order to offer a generic path generation? One possible solution would be to base the generators on a generic algorithm that takes into account a “path template” depending on the didactic domain in order to configure the right elements accordingly.

Modification of the context. Activity generation has received very little attention in TEL (Bezza *et al.* 2013). Our interest has focused on generating activities for declarative knowledge while proposing a generic approach. Therefore, the next most logical

question is: is it possible to generate activities for other types of knowledge (e.g., procedural knowledge) (Gorman 2002) while proposing a generic approach? In order to answer this question, the first step is to study the targeted knowledge and the learning objective (Bloom 1956) (i.e., remembering, understanding, applying, etc.). This should result in the identification of the requirements relating to this learning objective for this targeted knowledge (e.g., training on declarative knowledge requires repetition and variety). Then, a game genre that meets these needs has to be chosen.

Our interest has focused on *Roguelite* games. However, this type of game might not be motivating for everyone. In the context of declarative knowledge, the game genre target must provide repetition, variety, progression, and it should be based on generation (i.e., because of our research interest). Other game genres such as survival games (e.g., *Minecraft*) that are based on core gameplay loops revolving around gathering resources, building, and surviving with generated worlds and dynamic environment could provide the required elements for declarative knowledge training. A focus group or other relevant qualitative methods with game designers could be performed to identify the other possible game genres for declarative knowledge training. As a result, an interesting perspective would be to change the game model and the generator code to take into account other game genres. Changing the game genre requires modelling the new genre, defining new gameplay categories for that genre, matching the types of training tasks to the new gameplay categories (i.e., by following the proposed approach, see Chapter 6), possibly modifying the player adaptation mechanism, and probably rewriting the whole generation algorithm.

Although time-consuming to design, generation is a powerful process that can be used to build adapted and varied content. TEL research should focus more on the generation of pedagogical/learning activities and not just on the structuring of existing resources. Generating adapted and varied content facilitates the work of teachers, who need to build different activities according to the specific needs of learners.

Genericness of the game dimension. Our work has focused on just one game genre, *Roguelite*. However, since everyone is different, this genre is not necessarily appreciated by everybody. Furthermore, if several game genres are identified as compatible with declarative knowledge training, an interesting perspective would be to study their common features. Defining common features could possibly allow specifying, modelling and implementing the game dimension in a generic way. By doing so, it would avoid having to modify many of the components when changing game genres. However, having a generic modelling of the game dimension requires abstracting the game concepts to cover several game genres. If this abstraction is feasible, it would be possible to consider an extension mechanism such as the one already implemented for didactic domains. In this mechanism, the common parts of the generation algorithm would be shared and functions would allow the implementation of the variations.

BIBLIOGRAPHY

References for Chapter 1

- Anderson John R and Lebiere Christian J (2014), *The atomic components of thought*, Psychology Press, ISBN: 978-0-8058-2817-7.
- Bezza Assma, Balla Amar, and Marir Farhi (Sept. 2013), “An approach for personalizing learning content in e-learning systems: A review”, en, *in: 2013 Second International Conference on E-Learning and E-Technologies in Education (ICEEE)*, Lodz, Poland: IEEE, pp. 218–223, ISBN: 978-1-4673-5093-8, DOI: [10.1109/ICeLeTE.2013.6644377](https://doi.org/10.1109/ICeLeTE.2013.6644377), URL: <http://ieeexplore.ieee.org/document/6644377/> (visited on 05/30/2023).
- Brambilla Marco, Cabot Jordi, and Wimmer Manuel (2012), *Model-driven software engineering in practice*, en, Synthesis lectures on software engineering 1, San Rafael, Calif.: Morgan & Claypool, ISBN: 978-1-60845-882-0, DOI: [10.2200/S00441ED1V01Y201208SWE001](https://doi.org/10.2200/S00441ED1V01Y201208SWE001).
- Brame Cynthia J. and Biel Rachel (June 2015), “Test-Enhanced Learning: The Potential for Testing to Promote Greater Learning in Undergraduate Science Courses”, en, *in: CBE—Life Sciences Education* 14.2, ISSN: 1931-7913, DOI: [10.1187/cbe.14-11-0208](https://doi.org/10.1187/cbe.14-11-0208), URL: <https://www.lifescied.org/doi/10.1187/cbe.14-11-0208> (visited on 12/02/2022).
- Codish D. and Ravid G. (Nov. 2015), “Detecting playfulness in educational gamification through behavior patterns”, en, *in: IBM Journal of Research and Development* 59.6, pp. 1–14, ISSN: 0018-8646, 0018-8646, DOI: [10.1147/JRD.2015.2459651](https://doi.org/10.1147/JRD.2015.2459651), URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7330105> (visited on 10/25/2021).
- Kiili Kristian (2005), “Digital game-based learning: Towards an experiential gaming model”, *in: The Internet and Higher Education* 8.1, pp. 13–24, ISSN: 1096-7516, DOI: <https://doi.org/10.1016/j.iheduc.2004.12.001>, URL: <https://www.sciencedirect.com/science/article/pii/S1096751604000776>.
- Kim Jong W., Ritter Frank E., and Koubek Richard J. (Jan. 2013), “An integrated theory for improved skill acquisition and retention in the three stages of learning”, en, *in: Theoretical Issues in Ergonomics Science* 14.1, pp. 22–37, ISSN: 1463-922X, 1464-536X, DOI: [10.1080/1464536X.2011.573008](https://doi.org/10.1080/1464536X.2011.573008), URL: <http://www.tandfonline.com/doi/abs/10.1080/1464536X.2011.573008> (visited on 03/17/2022).
- Laforcade Pierre and Laghouaouta Youness (2018), “Generation of Adapted Learning Game Scenarios: A Model-Driven Engineering Approach”, *in: Computer Supported Education - 10th International Conference, CSEDU 2018, Funchal, Madeira, Portugal, March 15-17, 2018, Revised Selected Papers*, ed. by Bruce M. McLaren, Rob Reilly, Susan Zvacek, and James Uhomoihi, vol. 1022, Communications in Computer and

-
- Information Science, Springer, pp. 95–116, DOI: [10.1007/978-3-030-21151-6_6](https://doi.org/10.1007/978-3-030-21151-6_6), URL: https://doi.org/10.1007/978-3-030-21151-6%5C_6.
- Li Youling, Chen Di, and Deng Xinxia (Jan. 2024), “The impact of digital educational games on student’s motivation for learning: The mediating effect of learning engagement and the moderating effect of the digital environment”, en, in: *PLOS ONE* 19.1, ed. by José Gutiérrez-Pérez, e0294350, ISSN: 1932-6203, DOI: [10.1371/journal.pone.0294350](https://doi.org/10.1371/journal.pone.0294350), URL: <https://dx.plos.org/10.1371/journal.pone.0294350> (visited on 03/04/2024).
- Lopes Ricardo and Bidarra Rafael (Nov. 2011a), “A semantic generation framework for enabling adaptive game worlds”, en, in: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, Lisbon Portugal: ACM, pp. 1–8, ISBN: 978-1-4503-0827-4, DOI: [10.1145/2071423.2071431](https://doi.org/10.1145/2071423.2071431), URL: <https://dl.acm.org/doi/10.1145/2071423.2071431> (visited on 03/08/2024).
- Prensky Marc (2005), “Computer games and learning: Digital game-based learning”, in: *Handbook of computer game studies 18.2005*, pp. 97–122.
- Roediger Henry L. and Pyc Mary A. (Dec. 2012), “Inexpensive techniques to improve education: Applying cognitive psychology to enhance educational practice.”, en, in: *Journal of Applied Research in Memory and Cognition* 1.4, pp. 242–248, ISSN: 2211-369X, 2211-3681, DOI: [10.1016/j.jarmac.2012.09.002](https://doi.org/10.1016/j.jarmac.2012.09.002), URL: <http://doi.apa.org/getdoi.cfm?doi=10.1016/j.jarmac.2012.09.002> (visited on 12/05/2022).
- Smith Richard P (1981), “Boredom: A review”, in: *Human factors* 23.3, Publisher: SAGE Publications Sage CA: Los Angeles, CA, pp. 329–340, DOI: [10.1177/001872088102300308](https://doi.org/10.1177/001872088102300308).
- Sottet Jean-Sébastien, Ganneau Vincent, Calvary Gaëlle, Coutaz Joëlle, Demeure Alexandre, Favre Jean-Marie, and Demumieux Rachel (2007), “Model-Driven Adaptation for Plastic User Interfaces”, en, in: *Human-Computer Interaction – INTERACT 2007*, ed. by Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, vol. 4662, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 397–410, ISBN: 978-3-540-74794-9, DOI: [10.1007/978-3-540-74796-3_38](https://doi.org/10.1007/978-3-540-74796-3_38), URL: http://link.springer.com/10.1007/978-3-540-74796-3_38 (visited on 04/03/2024).
- Streicher Alexander and Smeddinck Jan D. (2016), “Personalized and Adaptive Serious Games”, en, in: *Entertainment Computing and Serious Games*, ed. by Ralf Dörner, Stefan Göbel, Michael Kickmeier-Rust, Maic Masuch, and Katharina Zweig, vol. 9970, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 332–377, ISBN: 978-3-319-46151-9, DOI: [10.1007/978-3-319-46152-6_14](https://doi.org/10.1007/978-3-319-46152-6_14), URL: http://link.springer.com/10.1007/978-3-319-46152-6_14 (visited on 09/17/2021).
- Susi Tarja, Johannesson Mikael, and Backlund Per (2007), “Serious games: An overview”, in: *Institutionen för kommunikation och information, Skövde*, Publisher: Institutionen för kommunikation och information.

Tchounikine Pierre, Mørch Anders I., and Bannon Liam J. (2009), “A Computer Science Perspective on Technology-Enhanced Learning Research”, en, *in: Technology-Enhanced Learning*, Dordrecht: Springer Netherlands, pp. 275–288, DOI: [10.1007/978-1-4020-9827-7_16](https://doi.org/10.1007/978-1-4020-9827-7_16), URL: http://link.springer.com/10.1007/978-1-4020-9827-7_16 (visited on 03/02/2022).

References for Chapter 2

- Abdul Jabbar Azita Iliya and Felicia Patrick (Mar. 2015), “Gameplay engagement and learning in game-based learning: A systematic review”, *in: Review of Educational Research* 85, DOI: [10.3102/0034654315577210](https://doi.org/10.3102/0034654315577210).
- Bakkes Sander, Tan Chek Tien, and Pisan Yusuf (July 2012), “Personalised gaming: a motivation and overview of literature”, en, *in: Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*, Auckland New Zealand: ACM, pp. 1–10, ISBN: 978-1-4503-1410-7, DOI: [10.1145/2336727.2336731](https://doi.org/10.1145/2336727.2336731), URL: <https://dl.acm.org/doi/10.1145/2336727.2336731> (visited on 05/30/2023).
- Bontchev Boyan Paskalev, Terzieva Valentina, and Paunova-Hubenova Elena (May 2021), “Personalization of serious games for learning”, en, *in: Interactive Technology and Smart Education* 18.1, pp. 50–68, ISSN: 1741-5659, 1741-5659, DOI: [10.1108/ITSE-05-2020-0069](https://doi.org/10.1108/ITSE-05-2020-0069), URL: <https://www.emerald.com/insight/content/doi/10.1108/ITSE-05-2020-0069/full/html> (visited on 04/21/2022).
- Brusilovsky Peter (1998), “Methods and Techniques of Adaptive Hypermedia”, en, *in: Adaptive Hypertext and Hypermedia*, ed. by Peter Brusilovsky, Alfred Kobsa, and Julita Vassileva, Dordrecht: Springer Netherlands, pp. 1–43, ISBN: 978-90-481-4944-5, DOI: [10.1007/978-94-017-0617-9_1](https://doi.org/10.1007/978-94-017-0617-9_1), URL: http://link.springer.com/10.1007/978-94-017-0617-9_1 (visited on 03/27/2024).
- Ćurčić Milenko, Milinković Dragica, and Radivojević Dragana (July 2018), “Educational Computer Software in the Function of Integrating and Individualization in Teaching of Mathematics and Knowledge of Nature”, en, *in: EURASIA Journal of Mathematics, Science and Technology Education* 14.12, ISSN: 13058223, DOI: [10.29333/ejmste/93808](https://doi.org/10.29333/ejmste/93808), URL: <https://www.ejmste.com/article/educational-computer-software-in-the-function-of-integrating-and-individualization-in-teaching-of-5571> (visited on 02/20/2024).
- De Freitas Sara (2018), “Are games effective learning tools? A review of educational games”, *in: Journal of Educational Technology & Society* 21.2, Publisher: JSTOR, pp. 74–84, URL: <http://www.jstor.org/stable/26388380>.
- Deterding Sebastian, Dixon Dan, Khaled Rilla, and Nacke Lennart (Sept. 2011), “From game design elements to gamefulness: defining "gamification"”, en, *in: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, Tampere Finland: ACM, pp. 9–15, ISBN: 978-1-4503-0816-8, DOI: [10.1145/2181037.2181040](https://doi.org/10.1145/2181037.2181040), URL: <https://dl.acm.org/doi/10.1145/2181037.2181040> (visited on 02/28/2024).

-
- Education United States Department of (2010), *Transforming American education: Learning powered by technology*.
- Fletcher J Dexter (1999), “Intelligent tutoring systems: then and now”, in: *NASA conference publication*, NASA, pp. 83–104.
- Göbel Stefan and Wendel Viktor (2016), “Personalization and Adaptation”, en, in: *Serious Games*, ed. by Ralf Dörner, Stefan Göbel, Wolfgang Effelsberg, and Josef Wiemeyer, Cham: Springer International Publishing, pp. 161–210, DOI: [10.1007/978-3-319-40612-1_7](https://doi.org/10.1007/978-3-319-40612-1_7), URL: http://link.springer.com/10.1007/978-3-319-40612-1_7 (visited on 03/25/2021).
- Goldberg Lewis R (1992), “The development of markers for the Big-Five factor structure.”, in: *Psychological assessment 4.1*, Publisher: American Psychological Association, p. 26, DOI: [10.1037/1040-3590.4.1.26](https://doi.org/10.1037/1040-3590.4.1.26).
- Guettat Belhassen, Chorfi Henda, and Jemni Mohamed (Jan. 2010), “Customized Learning Environment Based on Heterogeneous Traces”, en, in: *2010 International Conference on e-Education, e-Business, e-Management and e-Learning*, Sanya: IEEE, pp. 193–197, DOI: [10.1109/IC4E.2010.74](https://doi.org/10.1109/IC4E.2010.74), URL: <https://ieeexplore.ieee.org/document/5432420/> (visited on 02/21/2024).
- Ismail Heba and Belkhouche Boumediene (Nov. 2018), “A Reusable Software Architecture for Personalized Learning Systems”, en, in: *2018 International Conference on Innovations in Information Technology (IIT)*, Al Ain: IEEE, pp. 105–110, ISBN: 978-1-5386-6673-9, DOI: [10.1109/INNOVATIONS.2018.8605997](https://doi.org/10.1109/INNOVATIONS.2018.8605997), URL: <https://ieeexplore.ieee.org/document/8605997/> (visited on 05/27/2023).
- Klašnja-Miličević Aleksandra, Vesin Boban, Ivanović Mirjana, and Budimac Zoran (Apr. 2011), “E-Learning personalization based on hybrid recommendation strategy and learning style identification”, en, in: *Computers & Education 56.3*, pp. 885–899, ISSN: 03601315, DOI: [10.1016/j.compedu.2010.11.001](https://doi.org/10.1016/j.compedu.2010.11.001), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0360131510003222> (visited on 02/26/2024).
- Lefevre Marie, Jean-Daubias Stéphanie, and Guin Nathalie (2012), “An approach for unified personalization of learning”, in: *International workshop on personalization approaches in learning environments (PALE)-Conference user modeling, adaptation, and personalization*, pp. 5–10.
- Li Youling, Chen Di, and Deng Xinxia (Jan. 2024), “The impact of digital educational games on student’s motivation for learning: The mediating effect of learning engagement and the moderating effect of the digital environment”, en, in: *PLOS ONE 19.1*, ed. by José Gutiérrez-Pérez, e0294350, ISSN: 1932-6203, DOI: [10.1371/journal.pone.0294350](https://doi.org/10.1371/journal.pone.0294350), URL: <https://dx.plos.org/10.1371/journal.pone.0294350> (visited on 03/04/2024).
- Lopes Ricardo and Bidarra Rafael (June 2011b), “Adaptivity Challenges in Games and Simulations: A Survey”, en, in: *IEEE Transactions on Computational Intelligence and AI in Games 3.2*, pp. 85–99, ISSN: 1943-068X, 1943-0698, DOI: [10.1109/TCIAIG.2011.2152841](https://doi.org/10.1109/TCIAIG.2011.2152841), URL: <http://ieeexplore.ieee.org/document/5765665/> (visited on 03/08/2021).

-
- Marne Bertrand and Labat Jean Marc (2014), “Model and authoring tool to help teachers adapt serious games to their educational contexts”, en, in: *International Journal of Learning Technology* 9.2, p. 161, ISSN: 1477-8386, 1741-8119, DOI: [10.1504/IJLT.2014.064491](https://doi.org/10.1504/IJLT.2014.064491), URL: <http://www.inderscience.com/link.php?id=64491> (visited on 02/27/2024).
- Miraz Mahdi H., Ali Maaruf, and Excell Peter S. (May 2021), “Adaptive user interfaces and universal usability through plasticity of user interface design”, en, in: *Computer Science Review* 40, p. 100363, ISSN: 15740137, DOI: [10.1016/j.cosrev.2021.100363](https://doi.org/10.1016/j.cosrev.2021.100363), URL: <https://linkinghub.elsevier.com/retrieve/pii/S1574013721000034> (visited on 04/05/2024).
- Montserrat Baptiste, Lavoué Elise, and George Sébastien (2014), “Motivation for learning: Adaptive gamification for web-based learning environments”, in: *Proceedings of the 6th International Conference on Computer Supported Education*, Barcelona, Spain: SCITEPRESS - Science, pp. 117–125, DOI: [10.5220/0004848101170125](https://doi.org/10.5220/0004848101170125), URL: <http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0004848101170125> (visited on 02/26/2024).
- Montserrat Baptiste, Yessad Amel, Bouchet François, Lavoué Elise, and Luengo Vanda (2017), “MAGAM: A Multi-Aspect Generic Adaptation Model for Learning Environments”, en, in: *Data Driven Approaches in Digital Education*, vol. 10474, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 139–152, DOI: [10.1007/978-3-319-66610-5_11](https://doi.org/10.1007/978-3-319-66610-5_11), URL: http://link.springer.com/10.1007/978-3-319-66610-5_11 (visited on 03/31/2022).
- Nacke Lennart E., Bateman Chris, and Mandryk Regan L. (2014), “BrainHex: A neurobiological gamer typology survey”, in: *Entertainment Computing* 5.1, pp. 55–62, ISSN: 1875-9521, DOI: <https://doi.org/10.1016/j.entcom.2013.06.002>, URL: <https://www.sciencedirect.com/science/article/pii/S1875952113000086>.
- Natkin Stéphane, Yan Chen, Jumpertz Sylvie, and Marquet Bernard (2007), “Creating Multiplayer Ubiquitous Games using an adaptive narration model based on a user’s model”, in: *Proceedings of the 2007 DiGRA International Conference: Situated Play, DiGRA 2007, Tokyo, Japan, September 24-28, 2007*, ed. by Akira Baba, Digital Games Research Association, URL: <http://www.digra.org/digital-library/publications/creating-multiplayer-ubiquitous-games-using-an-adaptive-narration-model-based-on-a-users-model/>.
- Oppermann Reinhard and Rashev Rossen (1997), “Adaptability and adaptivity in learning systems”, in: *Knowledge transfer* 2, Publisher: Citeseer, pp. 173–179.
- Pham Xuan-Lam, Chen Gwo-Dong, Nguyen Thi-Huyen, and Hwang Wu-Yuin (July 2016), “Card-based design combined with spaced repetition: A new interface for displaying learning elements and improving active recall”, en, in: *Computers & Education* 98, pp. 142–156, ISSN: 03601315, DOI: [10.1016/j.compedu.2016.03.014](https://doi.org/10.1016/j.compedu.2016.03.014), URL: <https://linkinghub.elsevier.com/retrieve/pii/S036013151630077X> (visited on 12/16/2022).

-
- Plass Jan L. and Pawar Shashank (July 2020), “Toward a taxonomy of adaptivity for learning”, en, in: *Journal of Research on Technology in Education* 52.3, pp. 275–300, ISSN: 1539-1523, 1945-0818, DOI: [10.1080/15391523.2020.1719943](https://doi.org/10.1080/15391523.2020.1719943), URL: <https://www.tandfonline.com/doi/full/10.1080/15391523.2020.1719943> (visited on 09/06/2022).
- Roepke Rene, Drury Vincent, Schroeder Ulrik, and Meyer Ulrike (2021), “A modular architecture for personalized learning content in anti-phishing learning games”, in: *Software engineering (satellite events)*, DOI: [10.18154/RWTH-2021-02420](https://doi.org/10.18154/RWTH-2021-02420).
- Sablayrolles Louis, Lefevre Marie, Guin Nathalie, and Broisin Julien (2022), “Design and evaluation of a competency-based recommendation process”, in: *Intelligent tutoring systems*, ed. by Scott Crossley and Elvira Popescu, Cham: Springer International Publishing, pp. 148–160, ISBN: 978-3-031-09680-8, DOI: [10.1007/978-3-031-09680-8_14](https://doi.org/10.1007/978-3-031-09680-8_14).
- Sajjadi Pejman, Ewais Ahmed, and De Troyer Olga (Mar. 2022), “Individualization in serious games: A systematic review of the literature on the aspects of the players to adapt to”, en, in: *Entertainment Computing* 41, p. 100468, ISSN: 18759521, DOI: [10.1016/j.entcom.2021.100468](https://doi.org/10.1016/j.entcom.2021.100468), URL: <https://linkinghub.elsevier.com/retrieve/pii/S1875952121000653> (visited on 02/13/2024).
- Sehaba Karim (2014), “Adaptation dynamique des Environnements Informatiques pour l’Apprentissage Humain”, Habilitation à Diriger des Recherches (non publié), Lyon, France.
- Shemshack Atikah and Spector Jonathan Michael (Dec. 2020), “A systematic literature review of personalized learning terms”, en, in: *Smart Learning Environments* 7.1, p. 33, ISSN: 2196-7091, DOI: [10.1186/s40561-020-00140-9](https://doi.org/10.1186/s40561-020-00140-9), URL: <https://slejournal.springeropen.com/articles/10.1186/s40561-020-00140-9> (visited on 02/21/2024).
- Soflano Mario, Connolly Thomas M., and Hainey Thomas (Aug. 2015), “An application of adaptive games-based learning based on learning style to teach SQL”, en, in: *Computers & Education* 86, pp. 192–211, ISSN: 03601315, DOI: [10.1016/j.compedu.2015.03.015](https://doi.org/10.1016/j.compedu.2015.03.015), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0360131515000937> (visited on 02/26/2024).
- Streicher Alexander and Smeddinck Jan D. (2016), “Personalized and Adaptive Serious Games”, en, in: *Entertainment Computing and Serious Games*, ed. by Ralf Dörner, Stefan Göbel, Michael Kickmeier-Rust, Maic Masuch, and Katharina Zweig, vol. 9970, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 332–377, ISBN: 978-3-319-46151-9, DOI: [10.1007/978-3-319-46152-6_14](https://doi.org/10.1007/978-3-319-46152-6_14), URL: http://link.springer.com/10.1007/978-3-319-46152-6_14 (visited on 09/17/2021).
- Tondello Gustavo F., Wehbe Rina R., Diamond Lisa, Busch Marc, Marczewski Andrzej, and Nacke Lennart E. (Oct. 2016), “The Gamification User Types Hexad Scale”, en, in: *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, Austin Texas USA: ACM, pp. 229–243, ISBN: 978-1-4503-4456-2, DOI: [10.114](https://doi.org/10.114)

-
- 5/2967934.2968082, URL: <https://dl.acm.org/doi/10.1145/2967934.2968082> (visited on 07/09/2022).
- Vandewaetere Mieke, Desmet Piet, and Clarebout Geraldine (Jan. 2011), “The contribution of learner characteristics in the development of computer-based adaptive learning environments”, en, in: *Computers in Human Behavior* 27.1, pp. 118–130, ISSN: 07475632, DOI: [10.1016/j.chb.2010.07.038](https://doi.org/10.1016/j.chb.2010.07.038), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0747563210002347> (visited on 03/23/2022).
- Wilson Chunyu and Scott Bernard (Jan. 2017), “Adaptive systems in education: a review and conceptual unification”, en, in: *The International Journal of Information and Learning Technology* 34.1, pp. 2–19, ISSN: 2056-4880, DOI: [10.1108/IJILT-09-2016-0040](https://doi.org/10.1108/IJILT-09-2016-0040), URL: <https://www.emerald.com/insight/content/doi/10.1108/IJILT-09-2016-0040/full/html> (visited on 09/17/2021).

References for Chapter 3

- Amory Alan (Jan. 2007), “Game object model version II: a theoretical framework for educational game development”, en, in: *Educational Technology Research and Development* 55.1, pp. 51–77, ISSN: 1042-1629, 1556-6501, DOI: [10.1007/s11423-006-9001-x](https://doi.org/10.1007/s11423-006-9001-x), URL: <http://link.springer.com/10.1007/s11423-006-9001-x> (visited on 01/28/2022).
- Amory Alan, Naicker Kevin, Vincent Jacky, and Adams Claudia (Oct. 1999), “The use of computer games as an educational tool: identification of appropriate game types and game elements”, en, in: *British Journal of Educational Technology* 30.4, pp. 311–321, ISSN: 0007-1013, 1467-8535, DOI: [10.1111/1467-8535.00121](https://doi.org/10.1111/1467-8535.00121), URL: <https://onlinelibrary.wiley.com/doi/10.1111/1467-8535.00121> (visited on 09/20/2022).
- Barbosa André F. S., Pereira Pedro N. M., Dias João A. F. F., and Silva Frutuoso G. M. (2014), “A New Methodology of Design and Development of Serious Games”, en, in: *International Journal of Computer Games Technology* 2014, pp. 1–8, ISSN: 1687-7047, 1687-7055, DOI: [10.1155/2014/817167](https://doi.org/10.1155/2014/817167), URL: <http://www.hindawi.com/journals/ijcgt/2014/817167/> (visited on 01/28/2022).
- Bezza Assma, Balla Amar, and Marir Farhi (Sept. 2013), “An approach for personalizing learning content in e-learning systems: A review”, en, in: *2013 Second International Conference on E-Learning and E-Technologies in Education (ICEEE)*, Lodz, Poland: IEEE, pp. 218–223, ISBN: 978-1-4673-5093-8, DOI: [10.1109/ICeLeTE.2013.6644377](https://doi.org/10.1109/ICeLeTE.2013.6644377), URL: <http://ieeexplore.ieee.org/document/6644377/> (visited on 05/30/2023).
- Browne Cameron, Colton Simon, Cook Michael, Gow Jeremy, and Baumgarten Robin (Mar. 2014), “Toward the Adaptive Generation of Bespoke Game Content”, en, in: *Handbook of Digital Games*, ed. by Marios C. Angelides and Harry Agius, 1st ed., Wiley, pp. 15–61, ISBN: 978-1-118-79644-3, DOI: [10.1002/9781118796443.ch1](https://doi.org/10.1002/9781118796443.ch1), URL: <https://onlinelibrary.wiley.com/doi/10.1002/9781118796443.ch1> (visited on 03/08/2024).

-
- Callies Sophie (2016), “Architecture de génération automatique de scénarios pédagogiques de jeux sérieux éducatifs.”, fr, *in*: Publisher: Unpublished, DOI: [10.13140/RG.2.2.35521.76647](https://doi.org/10.13140/RG.2.2.35521.76647), URL: <http://rgdoi.net/10.13140/RG.2.2.35521.76647> (visited on 03/03/2021).
- Carpentier Kevin and Lourdeaux Domitile (2014), “Generation of Learning Situations According to the Learner’s Profile Within a Virtual Environment”, en, *in*: *Agents and Artificial Intelligence*, ed. by Joaquim Filipe and Ana Fred, vol. 449, Series Title: Communications in Computer and Information Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 245–260, ISBN: 978-3-662-44439-9, DOI: [10.1007/978-3-662-44440-5_15](https://doi.org/10.1007/978-3-662-44440-5_15), URL: http://link.springer.com/10.1007/978-3-662-44440-5_15 (visited on 03/08/2021).
- Carvalho Maira B., Bellotti Francesco, Berta Riccardo, De Gloria Alessandro, Sedano Carolina Islas, Hauge Jannicke Baalsrud, Hu Jun, and Rauterberg Matthias (Sept. 2015), “An activity theory-based model for serious games analysis and conceptual design”, en, *in*: *Computers & Education* 87, pp. 166–181, ISSN: 03601315, DOI: [10.1016/j.compedu.2015.03.023](https://doi.org/10.1016/j.compedu.2015.03.023), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0360131515001050> (visited on 09/17/2021).
- De Freitas Sara and Jarvis Steve (Jan. 2006), “A Framework for developing serious games to meet learner needs”, en, *in*: *Interservice/Industry Training, Simulation & Education Conference, I/ITSEC*, URL: <https://researchportal.murdoch.edu.au/esploro/outputs/conferencePaper/A-framework-for-developing-serious-games/991005544482907891>.
- Diwan Chaitali, Srinivasa Srinath, and Ram Prasad (2019), “Automatic Generation of Coherent Learning Pathways for Open Educational Resources”, en, *in*: *Transforming Learning with Meaningful Technologies*, ed. by Maren Scheffel, Julien Broisin, Viktoria Pammer-Schindler, Andri Ioannou, and Jan Schneider, vol. 11722, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, pp. 321–334, ISBN: 978-3-030-29735-0, DOI: [10.1007/978-3-030-29736-7_24](https://doi.org/10.1007/978-3-030-29736-7_24), URL: http://link.springer.com/10.1007/978-3-030-29736-7_24 (visited on 11/30/2021).
- Djaouti Damien, Alvarez Julian, Jessel Jean-Pierre, Methel Gilles, and Molinier P (2007), “Towards a classification of video games”, *in*: *Artificial and ambient intelligence convention (artificial societies for ambient intelligence)*.
- Dormans Joris and Bakkes Sander (2011), “Generating Missions and Spaces for Adaptable Play Experiences”, *in*: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3, pp. 216–228, DOI: [10.1109/TCIAIG.2011.2149523](https://doi.org/10.1109/TCIAIG.2011.2149523).
- Fujihira Keita, Hsueh Chu-Hsuan, and Ikeda Kokolo (2022), “Procedural Maze Generation Considering Difficulty from Human Players’ Perspectives”, *in*: *Advances in Computer Games*, Springer, pp. 165–175, DOI: [10.1007/978-3-031-11488-5_15](https://doi.org/10.1007/978-3-031-11488-5_15), URL: https://link.springer.com/chapter/10.1007/978-3-031-11488-5_15.
- Hall Joshua V., Wyeth Peta A., and Johnson Daniel (Oct. 2014), “Instructional objectives to core-gameplay: a serious game design technique”, en, *in*: *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, Toronto

-
- Ontario Canada: ACM, pp. 121–130, ISBN: 978-1-4503-3014-5, DOI: [10.1145/2658537.2658696](https://doi.org/10.1145/2658537.2658696), URL: <https://dl.acm.org/doi/10.1145/2658537.2658696> (visited on 12/15/2021).
- Hendriks Mark, Meijer Sebastiaan, Van Der Velden Joeri, and Iosup Alexandru (Feb. 2013), “Procedural content generation for games: A survey”, *in: ACM Trans. Multimedia Comput. Commun. Appl.* 9.1, Number of pages: 22 Place: New York, NY, USA Publisher: Association for Computing Machinery tex.articleno: 1 tex.issue_date: February 2013, ISSN: 1551-6857, DOI: [10.1145/2422956.2422957](https://doi.org/10.1145/2422956.2422957), URL: <https://doi.org/10.1145/2422956.2422957>.
- Holohan E., Melia M., McMullen D., and Pahl C. (2006), “The Generation of E-Learning Exercise Problems from Subject Ontologies”, *en, in: Sixth IEEE International Conference on Advanced Learning Technologies (ICALT’06)*, Kerkrade, The Netherlands: IEEE, pp. 967–969, ISBN: 978-0-7695-2632-4, DOI: [10.1109/ICALT.2006.1652605](https://doi.org/10.1109/ICALT.2006.1652605), URL: <http://ieeexplore.ieee.org/document/1652605/> (visited on 05/30/2023).
- Hunicke Robin, LeBlanc Marc, and Zubek Robert (2004), “MDA: A formal approach to game design and game research”, *in: Proceedings of the AAAI workshop on challenges in game AI*, vol. 4, Number: 1 tex.organization: San Jose, CA.
- Junior Rogério and Silva Frutuoso (Sept. 2021), “Redefining the MDA Framework—The Pursuit of a Game Design Ontology”, *en, in: Information* 12.10, ISSN: 2078-2489, DOI: [10.3390/info12100395](https://doi.org/10.3390/info12100395), URL: <https://www.mdpi.com/2078-2489/12/10/395> (visited on 01/21/2022).
- Khalifa Ahmed, Perez-Liebana Diego, Lucas Simon M., and Togelius Julian (July 2016), “General Video Game Level Generation”, *en, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016*, Denver Colorado USA: ACM, pp. 253–259, ISBN: 978-1-4503-4206-3, DOI: [10.1145/2908812.2908920](https://doi.org/10.1145/2908812.2908920), URL: <https://dl.acm.org/doi/10.1145/2908812.2908920> (visited on 03/21/2024).
- Kiili Kristian (2005), “Digital game-based learning: Towards an experiential gaming model”, *in: The Internet and Higher Education* 8.1, pp. 13–24, ISSN: 1096-7516, DOI: <https://doi.org/10.1016/j.iheduc.2004.12.001>, URL: <https://www.sciencedirect.com/science/article/pii/S1096751604000776>.
- Laforcade Pierre and Laghouaouta Youness (2018), “Generation of Adapted Learning Game Scenarios: A Model-Driven Engineering Approach”, *in: Computer Supported Education - 10th International Conference, CSEDU 2018, Funchal, Madeira, Portugal, March 15-17, 2018, Revised Selected Papers*, ed. by Bruce M. McLaren, Rob Reilly, Susan Zvacek, and James Uhomobhi, vol. 1022, Communications in Computer and Information Science, Springer, pp. 95–116, DOI: [10.1007/978-3-030-21151-6_6](https://doi.org/10.1007/978-3-030-21151-6_6), URL: https://doi.org/10.1007/978-3-030-21151-6_6.
- Lopes Ricardo and Bidarra Rafael (June 2011b), “Adaptivity Challenges in Games and Simulations: A Survey”, *en, in: IEEE Transactions on Computational Intelligence and AI in Games* 3.2, pp. 85–99, ISSN: 1943-068X, 1943-0698, DOI: [10.1109/TCAIG.2011.2152841](https://doi.org/10.1109/TCAIG.2011.2152841), URL: <http://ieeexplore.ieee.org/document/5765665/> (visited on 03/08/2021).

-
- Marne Bertrand, Wisdom John, Huynh-Kim-Bang Benjamin, and Labat Jean-Marc (2012), “The Six Facets of Serious Game Design: A Methodology Enhanced by Our Design Pattern Library”, en, in: *21st Century Learning for 21st Century Skills*, ed. by David Hutchison *et al.*, vol. 7563, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 208–221, ISBN: 978-3-642-33262-3, DOI: [10.1007/978-3-642-33263-0_17](https://doi.org/10.1007/978-3-642-33263-0_17), URL: http://link.springer.com/10.1007/978-3-642-33263-0_17 (visited on 01/28/2022).
- Melis Erica, Andres Eric, Budenbender Jochen, Frischauf Adrian, Goduadze George, Libbrecht Paul, Pollet Martin, and Ullrich Carsten (2001), “ActiveMath: A generic and adaptive web-based learning environment”, in: *International Journal of Artificial Intelligence in Education* 12, Publisher: Springer tex.hal_id: hal-00197329 tex.hal_version: v1, pp. 385–407, URL: <https://telearn.hal.science/hal-00197329>.
- Nakamura Jeanne and Csikszentmihalyi Mihaly (2009), “Flow theory and research”, in: *Handbook of positive psychology* 195, p. 206.
- Oliveira Sergio and Magalhaes Luis (Oct. 2017), “Adaptive content generation for games”, en, in: *2017 24^o Encontro Português de Computação Gráfica e Interação (EPCGI)*, Guimaraes: IEEE, pp. 1–8, ISBN: 978-1-5386-2080-9, DOI: [10.1109/EPCGI.2017.8124303](https://doi.org/10.1109/EPCGI.2017.8124303), URL: <http://ieeexplore.ieee.org/document/8124303/> (visited on 03/08/2024).
- Pereira Leonardo Tortoro, Prado Paulo Victor De Souza, Lopes Rafael Miranda, and Toledo Claudio Fabiano Motta (Oct. 2021), “Procedural generation of dungeons’ maps and locked-door missions through an evolutionary algorithm validated with players”, en, in: *Expert Systems with Applications* 180, p. 115009, ISSN: 09574174, DOI: [10.1016/j.eswa.2021.115009](https://doi.org/10.1016/j.eswa.2021.115009), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0957417421004504> (visited on 03/19/2024).
- Ripamonti Laura Anna, Mannalà Mattia, Gadia Davide, and Maggiorini Dario (Feb. 2017), “Procedural content generation for platformers: designing and testing FUN PLEdGE”, en, in: *Multimedia Tools and Applications* 76.4, pp. 5001–5050, ISSN: 1380-7501, 1573-7721, DOI: [10.1007/s11042-016-3636-3](https://doi.org/10.1007/s11042-016-3636-3), URL: <http://link.springer.com/10.1007/s11042-016-3636-3> (visited on 03/08/2024).
- Sehaba Karim and Hussaan Aarij Mahmood (Sept. 2013), “GOALS: Generator of adaptive learning scenarios”, in: *International Journal of Learning Technology*, 3rd ser. 8, Publisher: Inderscience, pp. 224–245, ISSN: 1477-8386, 1741-8119, DOI: [10.1504/IJLT.2013.057061](https://doi.org/10.1504/IJLT.2013.057061), URL: <https://hal.science/hal-01339255> (visited on 03/31/2022).
- Shaker Noor, Togelius Julian, and Nelson Mark J. (2016), *Procedural Content Generation in Games*, en, Computational Synthesis and Creative Systems, Cham: Springer International Publishing, DOI: [10.1007/978-3-319-42716-4](https://doi.org/10.1007/978-3-319-42716-4), URL: <http://link.springer.com/10.1007/978-3-319-42716-4> (visited on 03/08/2024).
- Sina Sigal, Rosenfeld Avi, and Kraus Sarit (Jan. 2014), “Generating content for scenario-based serious-games using crowdsourcing”, in: *Proceedings of the National Conference on Artificial Intelligence* 1, pp. 522–529, DOI: [10.1609/aaai.v28i1.8790](https://doi.org/10.1609/aaai.v28i1.8790).

-
- Soares De Lima Edirlei, Feijo Bruno, and Furtado Antonio L. (Oct. 2019), “Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning”, en, in: *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Rio de Janeiro, Brazil: IEEE, pp. 144–153, ISBN: 978-1-72814-637-9, DOI: [10.1109/SBGames.2019.00028](https://doi.org/10.1109/SBGames.2019.00028), URL: <https://ieeexplore.ieee.org/document/8924855/> (visited on 03/19/2024).
- Vassileva Julita (1995), “Dynamic courseware generation: at the cross point of CAL, ITS and authoring”, in: *Proceedings of ICCE*, vol. 95, pp. 290–297.
- Winn B.M. (2009), *Handbook of Research on Effective Electronic Gaming in Education*: en, ed. by Richard E. Ferdig, IGI Global, ISBN: 978-1-59904-808-6, DOI: [10.4018/978-1-59904-808-6](https://doi.org/10.4018/978-1-59904-808-6), URL: <http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59904-808-6> (visited on 01/28/2022).
- Yannakakis G. N. and Togelius J. (July 2011), “Experience-Driven Procedural Content Generation”, en, in: *IEEE Transactions on Affective Computing 2.3*, pp. 147–161, ISSN: 1949-3045, DOI: [10.1109/T-AFFC.2011.6](https://doi.org/10.1109/T-AFFC.2011.6), URL: <http://ieeexplore.ieee.org/document/5740836/> (visited on 03/08/2024).
- Yusoff Amri, Crowder Richard, Gilbert Lester, and Wills Gary (July 2009), “A Conceptual Framework for Serious Games”, en, in: *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, Riga, Latvia: IEEE, pp. 21–23, DOI: [10.1109/ICALT.2009.19](https://doi.org/10.1109/ICALT.2009.19), URL: <http://ieeexplore.ieee.org/document/5194153/> (visited on 01/28/2022).

References for Chapter 4

- Brambilla Marco, Cabot Jordi, and Wimmer Manuel (2012), *Model-driven software engineering in practice*, en, Synthesis lectures on software engineering 1, San Rafael, Calif.: Morgan & Claypool, ISBN: 978-1-60845-882-0, DOI: [10.2200/S00441ED1V01Y201208SWE001](https://doi.org/10.2200/S00441ED1V01Y201208SWE001).
- Harris John (Sept. 2020), “The Berlin Interpretation”, in: *Exploring Roguelike Games*, CRC Press, pp. 37–43, DOI: [10.1201/9781003053576-9](https://doi.org/10.1201/9781003053576-9), URL: <https://doi.org/10.1201/9781003053576-9>.
- Lemoine Bérénice and Laforcade Pierre (2023a), “Generator of personalised training games activities: A conceptual design approach”, in: *Games and learning alliance - 12th international conference, GALA 2023, dublin, ireland, november 29 - december 1, 2023, proceedings*, ed. by Pierpaolo Dondio, Mariana Rocha, Attracta Brennan, Avo Schönbohm, Francesca de Rosa, Antti Koskinen, and Francesco Bellotti, vol. 14475, Lecture notes in computer science, Springer, pp. 321–331, DOI: [10.1007/978-3-031-49065-1_31](https://doi.org/10.1007/978-3-031-49065-1_31), URL: https://doi.org/10.1007/978-3-031-49065-1_31.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (June 2023c), “Un framework de conception pour des générateurs d’activités de jeu variées et adaptées”, in: *11ème conférence sur les environnements informatiques pour l’Apprentissage humain*, Brest, France, pp. 88–99, URL: <https://hal.science/hal-04152004>.

-
- Pereira Leonardo Tortoro, Prado Paulo Victor De Souza, Lopes Rafael Miranda, and Toledo Claudio Fabiano Motta (Oct. 2021), “Procedural generation of dungeons’ maps and locked-door missions through an evolutionary algorithm validated with players”, en, in: *Expert Systems with Applications* 180, p. 115009, ISSN: 09574174, DOI: [10.1016/j.eswa.2021.115009](https://doi.org/10.1016/j.eswa.2021.115009), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0957417421004504> (visited on 03/19/2024).
- Plass Jan L. and Pawar Shashank (July 2020), “Toward a taxonomy of adaptivity for learning”, en, in: *Journal of Research on Technology in Education* 52.3, pp. 275–300, ISSN: 1539-1523, 1945-0818, DOI: [10.1080/15391523.2020.1719943](https://doi.org/10.1080/15391523.2020.1719943), URL: <https://www.tandfonline.com/doi/full/10.1080/15391523.2020.1719943> (visited on 09/06/2022).
- Sottet Jean-Sébastien, Ganneau Vincent, Calvary Gaëlle, Coutaz Joëlle, Demeure Alexandre, Favre Jean-Marie, and Demumieux Rachel (2007), “Model-Driven Adaptation for Plastic User Interfaces”, en, in: *Human-Computer Interaction – INTERACT 2007*, ed. by Cécilia Baranauskas, Philippe Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, vol. 4662, Series Title: Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 397–410, ISBN: 978-3-540-74794-9, DOI: [10.1007/978-3-540-74796-3_38](https://doi.org/10.1007/978-3-540-74796-3_38), URL: http://link.springer.com/10.1007/978-3-540-74796-3_38 (visited on 04/03/2024).
- Tchounikine Pierre, Mørch Anders I., and Bannon Liam J. (2009), “A Computer Science Perspective on Technology-Enhanced Learning Research”, en, in: *Technology-Enhanced Learning*, Dordrecht: Springer Netherlands, pp. 275–288, DOI: [10.1007/978-1-4020-9827-7_16](https://doi.org/10.1007/978-1-4020-9827-7_16), URL: http://link.springer.com/10.1007/978-1-4020-9827-7_16 (visited on 03/02/2022).
- Toy Michael, Wichman Glenn, Arnold Ken, and Lane Jon (1980), *Rogue*.
- Yannakakis G. N. and Togelius J. (July 2011), “Experience-Driven Procedural Content Generation”, en, in: *IEEE Transactions on Affective Computing* 2.3, pp. 147–161, ISSN: 1949-3045, DOI: [10.1109/T-AFFC.2011.6](https://doi.org/10.1109/T-AFFC.2011.6), URL: <http://ieeexplore.ieee.org/document/5740836/> (visited on 03/08/2024).

References for Chapter 5

- Carvalho Maira B., Bellotti Francesco, Berta Riccardo, De Gloria Alessandro, Sedano Carolina Islas, Hauge Jannicke Baalsrud, Hu Jun, and Rauterberg Matthias (Sept. 2015), “An activity theory-based model for serious games analysis and conceptual design”, en, in: *Computers & Education* 87, pp. 166–181, ISSN: 03601315, DOI: [10.1016/j.compedu.2015.03.023](https://doi.org/10.1016/j.compedu.2015.03.023), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0360131515001050> (visited on 09/17/2021).
- Djaouti Damien, Alvarez Julian, Jessel Jean-Pierre, Methel Gilles, and Molinier Pierre (2008), “A Gameplay Definition through Videogame Classification”, en, in: *International Journal of Computer Games Technology*, pp. 1–7, ISSN: 1687-7047, 1687-7055,

-
- DOI: [10.1155/2008/470350](https://doi.org/10.1155/2008/470350), URL: <http://www.hindawi.com/journals/ijcgt/2008/470350/> (visited on 07/11/2022).
- Kiili Kristian (2005), “Digital game-based learning: Towards an experiential gaming model”, *in: The Internet and Higher Education 8.1*, pp. 13–24, ISSN: 1096-7516, DOI: <https://doi.org/10.1016/j.iheduc.2004.12.001>, URL: <https://www.sciencedirect.com/science/article/pii/S1096751604000776>.
- Laforcade Pierre, Mottier Emeric, Jolivet Sébastien, and Lemoine Bérénice (Apr. 2022), “Expressing adaptations to take into account in generator-based exercisers: an exploratory study about multiplication facts”, *in: 14th International Conference on Computer Supported Education*, France, pp. 242–249, DOI: [10.5220/0011033100003182](https://doi.org/10.5220/0011033100003182), URL: <https://hal.archives-ouvertes.fr/hal-03711643>.
- Lemoine Bérénice and Laforcade Pierre (2023a), “Generator of personalised training games activities: A conceptual design approach”, *in: Games and learning alliance - 12th international conference, GALA 2023, dublin, ireland, november 29 - december 1, 2023, proceedings*, ed. by Pierpaolo Dondio, Mariana Rocha, Attracta Brennan, Avo Schönbohm, Francesca de Rosa, Antti Koskinen, and Francesco Bellotti, vol. 14475, Lecture notes in computer science, Springer, pp. 321–331, DOI: [10.1007/978-3-031-49065-1_31](https://doi.org/10.1007/978-3-031-49065-1_31), URL: https://doi.org/10.1007/978-3-031-49065-1_31.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2023a), “An analysis framework for designing declarative knowledge training games using roguelite genre”, *in: Proceedings of the 15th international conference on computer supported education, CSEDU 2023, volume 2, prague, czech republic, april 21-23*, ed. by Jelena Jovanovic, Irene-Angelica Chounta, James Uhomoibhi, and Bruce M. McLaren, SCITEPRESS, pp. 276–287, DOI: [10.5220/0011840200003470](https://doi.org/10.5220/0011840200003470), URL: <https://doi.org/10.5220/0011840200003470>.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (June 2023c), “Un framework de conception pour des générateurs d’activités de jeu variées et adaptées”, *in: 11ème conférence sur les environnements informatiques pour l’Apprentissage humain*, Brest, France, pp. 88–99, URL: <https://hal.science/hal-04152004>.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2024c), “Designing declarative knowledge training games: An analysis framework based on the roguelite genre”, *in: Computer supported education*, ed. by Bruce M. McLaren, James Uhomoibhi, Jelena Jovanovic, and Irene-Angelica Chounta, Cham: Springer Nature Switzerland, pp. 69–92, ISBN: 978-3-031-53656-4.
- Prensky Marc (2005), “Computer games and learning: Digital game-based learning”, *in: Handbook of computer game studies 18.2005*, pp. 97–122.

References for Chapter 6

- Bloom Benjamin Samuel (1956), “Taxonomy of educational objectives: The classification of educational goals”, *in: Cognitive domain*, Publisher: Longman.

-
- Chong Y, Wong M, and Thomson Fredrik E (2005), “The impact of learning styles on the effectiveness of digital games in education”, in: *Proceedings of the Symposium on Information Technology in Education, KDU College, Patailing Java, Malaysia*.
- Debabi Wassila and Champagnat Ronan (2017), “Towards architecture for pedagogical and game scenarios adaptation in serious games”, in: *International association for development of the information society (IADIS) conference on E-learning*, pp. 63–70.
- Degens Nick, Bril Ivo, and Braad Eelco (2015), “A three-dimensional model for educational game analysis & design”, en, in: *Foundations of Digital Games 2015*.
- Dondi Claudio and Moretti Michela (May 2007), “A methodological proposal for learning games selection and quality assessment”, en, in: *British Journal of Educational Technology* 38.3, pp. 502–512, ISSN: 0007-1013, 1467-8535, DOI: [10.1111/j.1467-8535.2007.00713.x](https://doi.org/10.1111/j.1467-8535.2007.00713.x), URL: <https://onlinelibrary.wiley.com/doi/10.1111/j.1467-8535.2007.00713.x> (visited on 03/16/2022).
- Gosper Maree and McNeill Margot (2012), “Implementing game-based learning: The MAPLET framework as a guide to learner-centred design and assessment”, English, in: *Assessment in Game-Based Learning*, United States: Springer, Springer Nature, pp. 217–233, ISBN: 978-1-4614-3545-7, DOI: [10.1007/978-1-4614-3546-4_12](https://doi.org/10.1007/978-1-4614-3546-4_12).
- Hall Joshua V., Wyeth Peta A., and Johnson Daniel (Oct. 2014), “Instructional objectives to core-gameplay: a serious game design technique”, en, in: *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, Toronto Ontario Canada: ACM, pp. 121–130, ISBN: 978-1-4503-3014-5, DOI: [10.1145/2658537.2658696](https://doi.org/10.1145/2658537.2658696), URL: <https://dl.acm.org/doi/10.1145/2658537.2658696> (visited on 12/15/2021).
- Jafari Seyed Mohammadbagher and Abdollahzade Zahra (Sept. 2019), “Investigating the relationship between learning style and game type in the game-based learning environment”, en, in: *Education and Information Technologies* 24.5, pp. 2841–2862, ISSN: 1360-2357, 1573-7608, DOI: [10.1007/s10639-019-09898-z](https://doi.org/10.1007/s10639-019-09898-z), URL: <http://link.springer.com/10.1007/s10639-019-09898-z> (visited on 01/29/2023).
- Kanaan Malak, Maillos Sébastien, and Muratet Mathieu (Sept. 2022), “Toward a learning game on Computational Thinking Driven by Competencies”, en, in: *European Conference on Games Based Learning* 16.1, pp. 288–296, ISSN: 2049-100X, 2049-0992, DOI: [10.34190/ecgbl.16.1.537](https://doi.org/10.34190/ecgbl.16.1.537), URL: <https://papers.academic-conferences.org/index.php/ecgbl/article/view/537> (visited on 02/02/2023).
- Khenissi Mohamed Ali, Essalmi Fathi, Jemni Mohamed, Kinshuk, Graf Sabine, and Chen Nian-Shing (Oct. 2016), “Relationship between learning styles and genres of games”, en, in: *Computers & Education* 101, pp. 1–14, ISSN: 03601315, DOI: [10.1016/j.compedu.2016.05.005](https://doi.org/10.1016/j.compedu.2016.05.005), URL: <https://linkinghub.elsevier.com/retrieve/pii/S0360131516301154> (visited on 01/29/2023).
- Kiili Kristian (2005), “Digital game-based learning: Towards an experiential gaming model”, in: *The Internet and Higher Education* 8.1, pp. 13–24, ISSN: 1096-7516, DOI: <https://doi.org/10.1016/j.iheduc.2004.12.001>, URL: <https://www.sciencedirect.com/science/article/pii/S1096751604000776>.

-
- Lameras Petros, Arnab Sylvester, Dunwell Ian, Stewart Craig, Clarke Samantha, and Petridis Panagiotis (June 2017), “Essential features of serious games design in higher education: Linking learning attributes to game mechanics: Essential features of serious games design”, en, in: *British Journal of Educational Technology* 48.4, pp. 972–994, ISSN: 00071013, DOI: [10.1111/bjet.12467](https://doi.org/10.1111/bjet.12467), URL: <https://onlinelibrary.wiley.com/doi/10.1111/bjet.12467> (visited on 02/20/2023).
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2023b), “Mapping task types and gameplay categories in the context of declarative knowledge training”, in: *Proceedings of the 15th international conference on computer supported education, CSEDU 2023, volume 2, prague, czech republic, april 21-23*, ed. by Jelena Jovanovic, Irene-Angelica Chounta, James Uhomoibhi, and Bruce M. McLaren, SCITEPRESS, pp. 264–275, DOI: [10.5220/0011840100003470](https://doi.org/10.5220/0011840100003470), URL: <https://doi.org/10.5220/0011840100003470>.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2024b), “An approach for mapping declarative knowledge training task types to gameplay categories”, in: *Computer supported education*, ed. by Bruce M. McLaren, James Uhomoibhi, Jelena Jovanovic, and Irene-Angelica Chounta, Cham: Springer Nature Switzerland, pp. 47–68, ISBN: 978-3-031-53656-4.
- Lim Theodore, Carvalho Maira B, Bellotti Francesco, Arnab Sylvester, de Freitas Sara, Louchart Sandy, Suttie Neil, Berta Riccardo, and Gloria Alessandro De (2013), “The LM-GM framework for Serious Games Analysis”, en, in: *Pittsburgh: University of Pittsburgh*.
- Prensky Marc (2005), “Computer games and learning: Digital game-based learning”, in: *Handbook of computer game studies* 18.2005, pp. 97–122.
- Rapeepisarn Kowit, Wong Kok Wai, Fung Chun Che, and Khine Myint Swe (2008), “The Relationship between Game Genres, Learning Techniques and Learning Styles in Educational Computer Games”, en, in: *Technologies for E-Learning and Digital Entertainment*, vol. 5093, Springer Berlin Heidelberg, pp. 497–508, ISBN: 978-3-540-69734-3, DOI: [10.1007/978-3-540-69736-7_53](https://doi.org/10.1007/978-3-540-69736-7_53), URL: http://link.springer.com/10.1007/978-3-540-69736-7_53 (visited on 09/23/2022).
- Sherry John L (2010), “Matching computer game genres to educational outcomes”, in: *Teaching and Learning with Technology*, Routledge, pp. 234–246.

References for Chapter 7

- Debabi Wassila and Champagnat Ronan (2017), “Towards architecture for pedagogical and game scenarios adaptation in serious games”, in: *International association for development of the information society (IADIS) conference on E-learning*, pp. 63–70.
- Gosper Maree (2011), “MAPLET—A Framework for Matching Aims, Processes, Learner Expertise and Technologies”, in: *Multiple perspectives on problem solving and learning in the digital age*, Springer, pp. 23–36.

-
- Lemoine Bérénice and Laforcade Pierre (2023a), “Generator of personalised training games activities: A conceptual design approach”, *in: Games and learning alliance - 12th international conference, GALA 2023, dublin, ireland, november 29 - december 1, 2023, proceedings*, ed. by Pierpaolo Dondio, Mariana Rocha, Attracta Brennan, Avo Schönbohm, Francesca de Rosa, Antti Koskinen, and Francesco Bellotti, vol. 14475, Lecture notes in computer science, Springer, pp. 321–331, DOI: [10.1007/978-3-031-49065-1_31](https://doi.org/10.1007/978-3-031-49065-1_31), URL: https://doi.org/10.1007/978-3-031-49065-1_31.
- Lemoine Bérénice and Laforcade Pierre (2023b), “Mapping facts to concrete game elements for generation purposes: A conceptual approach”, *in: Games and learning alliance - 12th international conference, GALA 2023, dublin, ireland, november 29 - december 1, 2023, proceedings*, ed. by Pierpaolo Dondio, Mariana Rocha, Attracta Brennan, Avo Schönbohm, Francesca de Rosa, Antti Koskinen, and Francesco Bellotti, vol. 14475, Lecture notes in computer science, Springer, pp. 342–352, DOI: [10.1007/978-3-031-49065-1_33](https://doi.org/10.1007/978-3-031-49065-1_33), URL: https://doi.org/10.1007/978-3-031-49065-1_33.
- Lim Theodore, Carvalho Maira B, Bellotti Francesco, Arnab Sylvester, de Freitas Sara, Louchart Sandy, Suttie Neil, Berta Riccardo, and Gloria Alessandro De (2013), “The LM-GM framework for Serious Games Analysis”, *en, in: Pittsburgh: University of Pittsburgh*.

References for Chapter 8

- Laforcade Pierre and Laghouaouta Youness (2018), “Generation of Adapted Learning Game Scenarios: A Model-Driven Engineering Approach”, *in: Computer Supported Education - 10th International Conference, CSEDU 2018, Funchal, Madeira, Portugal, March 15-17, 2018, Revised Selected Papers*, ed. by Bruce M. McLaren, Rob Reilly, Susan Zvacek, and James Uhomuibhi, vol. 1022, Communications in Computer and Information Science, Springer, pp. 95–116, DOI: [10.1007/978-3-030-21151-6_6](https://doi.org/10.1007/978-3-030-21151-6_6), URL: https://doi.org/10.1007/978-3-030-21151-6_6.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2024a), “A framework for generators of varied and adapted training game activities”, *in: Technology enhanced learning for inclusive and equitable quality education*, ed. by Rafael Ferreira Mello, Nikol Rummel, Ioana Jivet, Gerti Pishtari, and José A. Ruipérez Valiente, Cham: Springer Nature Switzerland, pp. 237–252, ISBN: 978-3-031-72315-5.
- Melero Javier, El-Kechai Naima, Yessad Amel, and Labat Jean-Marc (2016), “Adapting learning paths in serious games: an approach based on teachers’ requirements”, *in: Computer supported education - 7th international conference, CSEDU 2015, lisbon, portugal, may 23-25, 2015, revised selected papers*, vol. 583, Communications in computer and information science, Springer, pp. 376–394, DOI: [10.1007/978-3-319-29585-5_22](https://doi.org/10.1007/978-3-319-29585-5_22), URL: <https://hal.sorbonne-universite.fr/hal-01303795>.
- Sehaba Karim and Hussaan Aarij Mahmood (Sept. 2013), “GOALS: Generator of adaptive learning scenarios”, *in: International Journal of Learning Technology*, 3rd ser. 8, Pub-

-
- lisher: Inderscience, pp. 224–245, ISSN: 1477-8386, 1741-8119, DOI: [10.1504/IJLT.2013.057061](https://doi.org/10.1504/IJLT.2013.057061), URL: <https://hal.science/hal-01339255> (visited on 03/31/2022).
- Shvets Alexander (2018), *Dive Into Design Patterns*, Refactoring.Guru, URL: <https://refactoring.guru/design-patterns/template-method>.
- Steinberg David, Budinsky Frank, Paternostro Marcelo, and Merks Ed (2009), *EMF: Eclipse modeling framework 2.0*, 2nd ed., Addison-Wesley Professional, ISBN: 0-321-33188-5.

References for Chapter 9

- Bloom Benjamin Samuel (1956), “Taxonomy of educational objectives: The classification of educational goals”, in: *Cognitive domain*, Publisher: Longman.
- Melero Javier, El-Kechai Naima, Yessad Amel, and Labat Jean-Marc (2016), “Adapting learning paths in serious games: an approach based on teachers’ requirements”, in: *Computer supported education - 7th international conference, CSEDU 2015, lisbon, portugal, may 23-25, 2015, revised selected papers*, vol. 583, Communications in computer and information science, Springer, pp. 376–394, DOI: [10.1007/978-3-319-29585-5_22](https://doi.org/10.1007/978-3-319-29585-5_22), URL: <https://hal.sorbonne-universite.fr/hal-01303795>.

References for Chapter 11

- Bezza Assma, Balla Amar, and Marir Farhi (Sept. 2013), “An approach for personalizing learning content in e-learning systems: A review”, en, in: *2013 Second International Conference on E-Learning and E-Technologies in Education (ICEEE)*, Lodz, Poland: IEEE, pp. 218–223, ISBN: 978-1-4673-5093-8, DOI: [10.1109/ICeLeTE.2013.6644377](https://doi.org/10.1109/ICeLeTE.2013.6644377), URL: <http://ieeexplore.ieee.org/document/6644377/> (visited on 05/30/2023).
- Bimba Andrew Thomas, Idris Norisma, Al-Hunaiyyan Ahmed, Mahmud Rohana Binti, and Shuib Nor Liyana Bt Mohd (Oct. 2017), “Adaptive feedback in computer-based learning environments: a review”, en, in: *Adaptive Behavior 25.5*, pp. 217–234, ISSN: 1059-7123, 1741-2633, DOI: [10.1177/1059712317727590](https://doi.org/10.1177/1059712317727590), URL: <http://journals.sagepub.com/doi/10.1177/1059712317727590> (visited on 06/28/2024).
- Bloom Benjamin Samuel (1956), “Taxonomy of educational objectives: The classification of educational goals”, in: *Cognitive domain*, Publisher: Longman.
- Gorman Michael E (2002), “Types of Knowledge and Their Roles in Technology Transfer”, en, in: DOI: <https://doi.org/10.1023/A:1015672119590>.
- Lemoine Bérénice and Laforcade Pierre (2023a), “Generator of personalised training games activities: A conceptual design approach”, in: *Games and learning alliance - 12th international conference, GALA 2023, dublin, ireland, november 29 - december 1, 2023, proceedings*, ed. by Pierpaolo Dondio, Mariana Rocha, Attracta Brennan, Avo Schönbohm, Francesca de Rosa, Antti Koskinen, and Francesco Bellotti, vol. 14475,

-
- Lecture notes in computer science, Springer, pp. 321–331, DOI: [10.1007/978-3-031-49065-1_31](https://doi.org/10.1007/978-3-031-49065-1_31), URL: https://doi.org/10.1007/978-3-031-49065-1_31.
- Lemoine Bérénice and Laforcade Pierre (2023b), “Mapping facts to concrete game elements for generation purposes: A conceptual approach”, in: *Games and learning alliance - 12th international conference, GALA 2023, dublin, ireland, november 29 - december 1, 2023, proceedings*, ed. by Pierpaolo Dondio, Mariana Rocha, Attracta Brennan, Avo Schönbohm, Francesca de Rosa, Antti Koskinen, and Francesco Bellotti, vol. 14475, Lecture notes in computer science, Springer, pp. 342–352, DOI: [10.1007/978-3-031-49065-1_33](https://doi.org/10.1007/978-3-031-49065-1_33), URL: https://doi.org/10.1007/978-3-031-49065-1_33.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2023a), “An analysis framework for designing declarative knowledge training games using roguelite genre”, in: *Proceedings of the 15th international conference on computer supported education, CSEDU 2023, volume 2, prague, czech republic, april 21-23*, ed. by Jelena Jovanovic, Irene-Angelica Chounta, James Uhomobhi, and Bruce M. McLaren, SCITEPRESS, pp. 276–287, DOI: [10.5220/0011840200003470](https://doi.org/10.5220/0011840200003470), URL: <https://doi.org/10.5220/0011840200003470>.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2023b), “Mapping task types and gameplay categories in the context of declarative knowledge training”, in: *Proceedings of the 15th international conference on computer supported education, CSEDU 2023, volume 2, prague, czech republic, april 21-23*, ed. by Jelena Jovanovic, Irene-Angelica Chounta, James Uhomobhi, and Bruce M. McLaren, SCITEPRESS, pp. 264–275, DOI: [10.5220/0011840100003470](https://doi.org/10.5220/0011840100003470), URL: <https://doi.org/10.5220/0011840100003470>.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (June 2023c), “Un framework de conception pour des générateurs d’activités de jeu variées et adaptées”, in: *11ème conférence sur les environnements informatiques pour l’Apprentissage humain*, Brest, France, pp. 88–99, URL: <https://hal.science/hal-04152004>.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2024b), “An approach for mapping declarative knowledge training task types to gameplay categories”, in: *Computer supported education*, ed. by Bruce M. McLaren, James Uhomobhi, Jelena Jovanovic, and Irene-Angelica Chounta, Cham: Springer Nature Switzerland, pp. 47–68, ISBN: 978-3-031-53656-4.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (2024c), “Designing declarative knowledge training games: An analysis framework based on the roguelite genre”, in: *Computer supported education*, ed. by Bruce M. McLaren, James Uhomobhi, Jelena Jovanovic, and Irene-Angelica Chounta, Cham: Springer Nature Switzerland, pp. 69–92, ISBN: 978-3-031-53656-4.
- Lemoine Bérénice, Laforcade Pierre, and George Sébastien (Sept. 2024d), “A Framework for Generators of Varied and Adapted Training Game Activities”, en, in: *Nineteenth European Conference on Technology Enhanced Learning ECTEL 2024*, to be published, Lecture Notes in Computer Science.

-
- Melero Javier, El-Kechai Naima, Yessad Amel, and Labat Jean-Marc (2016), “Adapting learning paths in serious games: an approach based on teachers’ requirements”, in: *Computer supported education - 7th international conference, CSEDU 2015, lisbon, portugal, may 23-25, 2015, revised selected papers*, vol. 583, Communications in computer and information science, Springer, pp. 376–394, DOI: [10.1007/978-3-319-29585-5_22](https://doi.org/10.1007/978-3-319-29585-5_22), URL: <https://hal.sorbonne-universite.fr/hal-01303795>.
- Ramaprasad Arkalgud (Jan. 1983), “On the definition of feedback”, in: *Behavioral Science* 28, pp. 4–13, DOI: [10.1002/bs.3830280103](https://doi.org/10.1002/bs.3830280103).
- Shute Valerie J. (Mar. 2008), “Focus on Formative Feedback”, en, in: *Review of Educational Research* 78.1, pp. 153–189, ISSN: 0034-6543, 1935-1046, DOI: [10.3102/0034654307313795](https://doi.org/10.3102/0034654307313795), URL: <http://journals.sagepub.com/doi/10.3102/0034654307313795> (visited on 02/23/2022).
- Yannakakis G. N. and Togelius J. (July 2011), “Experience-Driven Procedural Content Generation”, en, in: *IEEE Transactions on Affective Computing* 2.3, pp. 147–161, ISSN: 1949-3045, DOI: [10.1109/T-AFFC.2011.6](https://doi.org/10.1109/T-AFFC.2011.6), URL: <http://ieeexplore.ieee.org/document/5740836/> (visited on 03/08/2024).

ANALYSIS OF EXISTING GAMES FOR MULTIPLICATION TABLES TRAINING

Table A.1: Analysis of existing games for Multiplication Table training

Platform	Name	Description	Education Settings	Game Settings	Persistence
Computer Online source	Friends & Figo	Destroy (keyboard + click) boxes in a limited time with the correct answers	Tables from 1 to 12		Yes
Computer Online source	Figo's Factory	Move (keyboard) to collect gifts with questions and place (keyboard + click) them in the boxes with the correct answer	Tables from 1 to 12	Avatar	No
Computer Online source	My Smart Horse / Rally v10	Choose (click) the correct answer between the 4 proposals and avoid obstacles by jumping (click)	Tables from 1 to 12	Avatar & Vehicle, Buyable objects, Game Mode (Training, Competition vs PCs)	Yes
Computer Online source	Happy Burger	Click on the correct burger ingredients based on clients commands (questions)	Tables from 1 to 12		No
Computer Online source	Calculs Sous-Marin	Reconstruction of the question and answering between 3 proposals by moving and catching (keyboard) the bubulles	Difficulty (Easy, Hard)	Avatar's vehicle	No
Computer Online source	Course de multiplication	Choose (click) the correct answer between the 4 proposals and avoid obstacles by jumping (click)	Tables from 1 to 12	Game Mode (Training, Competition vs PCs)	Yes
Computer Online source	Chat de multiplication	Jump (click) on the platform with the correct answer between the 4 proposals			No

Continued on next page

Table A.1: Analysis of existing games for Multiplication Table training (Continued)

Platform	Name	Description	Education Settings	Game Settings	Persistence
Computer Online source	Les chiffres plongeurs	Reconstruction of the question and answering between 3 proposals by moving and catching (keyboard) the bubulles	Tables from 1 to 10		No
Computer Online source	Jeu de mémoire	A memory game but with multiplication and results on the hidden side of the cards	Tables from 1 to 12	Game Mode (1-2 players)	No
Computer Online source	Jeu de tir	Shoot (click) on the target with the correct answer between 4 proposals	Tables from 1 to 10		No
Computer Online source	Spuq ballons	Shoot (click) on the balloon with the correct answer (lot of balloons present)	Tables from 1 to 12, Difficulty (Easy, Hard)		No
Computer Online source	Sauve les animaux	Jump (click at the right time) on the platform with the correct answer between the 4 proposals	Tables from 1 to 12, Difficulty (Easy, Hard)	Avatar	Yes
Computer Online source	Spuq	Enter the correct answer by clicking (virtual numpad)	Table from 1 to 12 (only one), Difficulty (Easy, Hard)		No
Computer Online source	Logiciel Educatif.fr	Enter (click) the correct answers using the keyboard for the horse to move and win	Tables from 2 to 12		No
Computer Online source	iEducat!f	Choose (click) the correct answer between 4 proposals	Tables from 1 to 12 (One only)		No
Computer Online source	Course aux tables de multiplication	Move the car by choosing (click) the correct answer between 100 proposals	Table from 1 to 10 (one or all), Difficulty (Discovery, Hard, Expert, Time Competition)		No
Computer Online source	Speedy Calculo	Engage the lever (click) to get the question, Determine the correct question between 4 proposals and choose (click) the correct answer between 4 proposals	Level (Primary Graduation Levels in France)		No
Computer Online source	Multiplication 4 in a Row	Choose (click) the correct answer between 4 proposals to place a pawn (Power 4 Board Game)		1 or 2 players	No

Continued on next page

Table A.1: Analysis of existing games for Multiplication Table training (Continued)

Platform	Name	Description	Education Settings	Game Settings	Persistence
Phone / Tablet Application	Learn Multiplication Table - Times Table Game	Enter the correct answer to win money	Time limit, Difficulty, Question type ($A \times B = ? \vee A \times ? = C$)	Buyable objects	Yes
	Le château des multiplications	Destroy the bricks with the correct answer on the way to win money (Mario type)	Table from 1 to 11 (only one), Multiplicand Position (Left \oplus Right)	Difficulty (Easy, Medium, Hard)	No
	Multiplication	Choose (click) on the right answer between 4 proposals multiple times for the avatar to keep going	Table from 1 to 10 or 12	Buyable objects	Yes

GAMEPLAY MOCK-UPS EVALUATION QUESTIONNAIRE

This questionnaire has been used to evaluate gameplay mock-ups for training tasks involving multiplication tables. The questionnaire is in French and presents gameplay mock-ups for each of the training tasks, as well as a description of how the gameplay is intended to work.

Évaluation de gameplays

Ce questionnaire a pour objectif d'évaluer la pertinence de gameplay (= fonctionnalité du jeu) vis-à-vis des 6 types de tâches définies avec vous :

- complétion de faits à 1 élément manquant (e.g., $3 \times ? = 15$) ;
- complétion de faits à 2 éléments manquants (e.g., $3 \times ? = ?$) ;
- reconstitution de faits (e.g., $? \times ? = ?$) ;
- validation de faits (e.g., $3 \times 5 = 15$, vrai ou faux ?) ;
- vérification d'appartenance (e.g., *Quels éléments parmi [3, 5, 9, 12, 15, 17, 23] ne sont pas des résultats de la table de 3 ?*).

Ce questionnaire devrait prendre environ 20 min.

Les différents gameplays imaginés pour réaliser ces six tâches ont été maquetés.

Nous allons vous présenter ces maquettes une à une en vous les expliquant afin que vous compreniez en quoi le gameplay consistera.

ATTENTION. Les consignes (en bas des maquettes), les énoncés (faits complets ou incomplets visibles dans la salle), le nombre d'éléments présents ne sont que fictifs et ne doivent pas faire l'objet de votre attention même si vous les trouvez mal formulés ou qu'il y a trop d'éléments. Nous attendons de vous un avis sur les actions que l'élève devra accomplir pour répondre (exemple : toucher l'élément portant la réponse qu'il choisit).

Nous vous remercions par avance de toute participation qui nous permet de faire avancer notre recherche.

* Indique une question obligatoire

Complétion à 1 élément manquant

Dans cette première section nous vous demandons d'évaluer des propositions de gameplays pour de la complétion à 1 élément manquant, exemples de question : $3 \times 5 = ?$, $3 \times ? = 15$, $? \times 5 = 15$, $? = 3 \times 5$, $15 = ? \times 5$ ou encore $15 = 3 \times ?$.

1. Les quatre images présentées sont quatre variations d'un même gameplay qui consiste à sélectionner l'objet portant la bonne réponse. *
- En cas de bonne réponse, la porte de sortie s'ouvre, sinon le joueur perd une vie (ou meurt et doit recommencer s'il n'avait plus de vie).
- Dans les deux images du haut, la sélection s'effectue par ouverture du coffre et cassage du pot respectivement. En bas à gauche, la sélection s'effectue par toucher (toucher la porte). Enfin, en bas à droite, la sélection s'effectue par le passage à travers le bon pont.

Trouvez-vous ces quatre variantes de gameplays pertinentes ?



Une seule réponse possible.

- OUI
- NON

2. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

3. L'image présentée représente un gameplay de sélection unique d'une réponse. *

Ici, un seul objet possède l'ensemble des choix possibles.

Le joueur doit orienter la lumière vers la bonne réponse puis s'éloigner de l'élément pour que la réponse soit "verrouillée" (la consigne en bas va être améliorée ⇒ NE JUGEZ PAS LES CONSIGNES SVP).

Si la réponse choisie est correcte, la porte s'ouvre, sinon le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
 NON

4. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

5. L'image présente le même gameplay qu'à la question précédente sauf que cette fois-ci, la vérification du choix s'effectue par action du joueur sur le levier. *

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

6. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

7. L'image ci-dessous présente deux gameplays de positionnement de l'avatar sur la bonne réponse. *
- L'avatar doit donc se positionner sur le lieu.
- Dans le premier cas, la réponse est validée lorsque l'avatar s'arrête de bouger sans attente.
- Dans le second, la validation s'effectue après un arrêt complet de l'avatar pendant un léger délai (le temps n'est pas encore défini et sera réfléchi et modifié après des tests).
- Comme les précédents, si la réponse est correcte, la porte s'ouvre, sinon le joueur perd une vie.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

OUI

NON

8. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

9. L'image présente trois gameplays de déplacement d'un objet. *
- Dans le premier, le joueur doit déplacer le pot pour le positionner sur la bonne dalle
- Dans le second, le joueur doit pousser la bonne statue vers la bonne dalle (déplacement vertical des statues).
- Dans le troisième, le joueur doit positionner le pot sur la bonne réponse de l'image (120 choix, nous pourrions aller jusqu'à 144-150 pour prendre en compte jusqu'à 12×12).
- Dans les trois cas, la réponse est validée et la porte de sortie s'ouvre lorsque la bonne réponse détecte la présence de la statue ou du pot. En cas d'erreur, le joueur perd une vie.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

- OUI
- NON

10. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

11. L'image présente deux gameplays de déplacement. *
Le premier consiste à déplacer le bon pot sur la dalle rose, le second à déplacer le bon pot sur la dalle grise (partie de la réponse manquante).
Dans les deux cas, la porte s'ouvre lorsque la dalle détecte la présence du pot avec la bonne réponse.
En cas d'erreur de détection d'un mauvais pot, le joueur perd une vie.

Trouvez-vous ces gameplay pertinents ?



Une seule réponse possible.

OUI

NON

12. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

13. L'image présente un gameplay de sélection d'un objet. *
Ici, le joueur doit tuer l'ennemi portant la bonne réponse.
Certains ennemis ne possèdent pas de numéro et sont juste présents pour la difficulté du jeu.

Comme pour les précédents, la porte s'ouvre si le bon ennemi a été tué. En cas d'erreur, le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

NON

14. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

15. L'image présente un gameplay de position & saisie. *

Le joueur doit se placer sur la dalle puis saisir avec un périphérique (ex : clavier) la réponse attendue.

Comme pour les précédents, la porte s'ouvre, la bonne réponse a été saisie. En cas d'erreur, le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

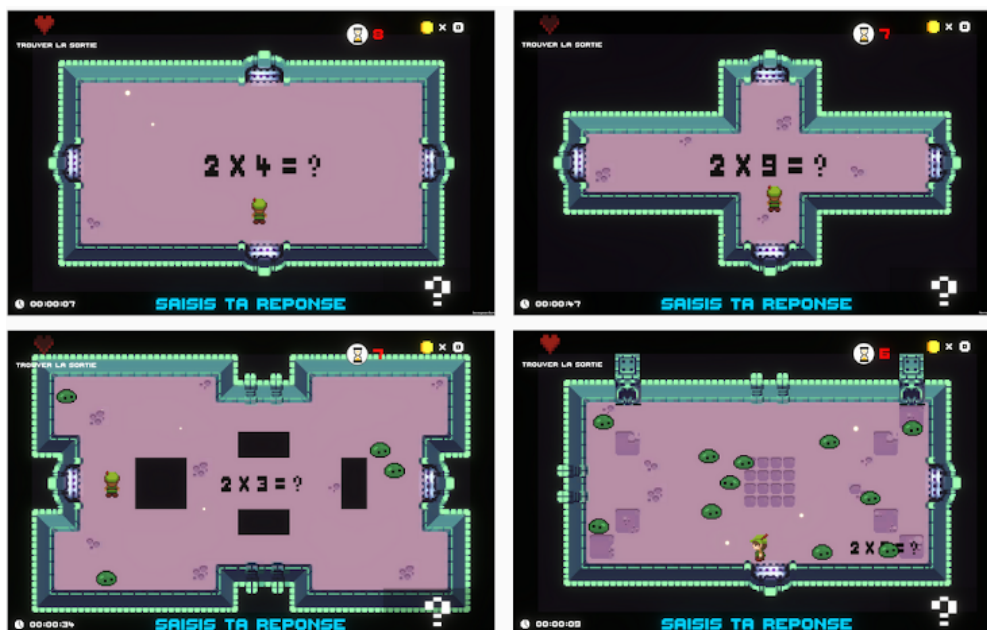
16. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

17. L'image présente des gameplays de saisie. *

Le joueur doit simplement saisir la bonne réponse à l'aide d'un périphérique (ex : clavier).
Dans certains cas, il peut y avoir des ennemis à éviter en plus.

Comme pour les précédents, la porte s'ouvre si la bonne réponse a été saisie. En cas d'erreur, le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

18. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

Complétion à 2 éléments manquants

Dans cette seconde section nous vous demandons

d'évaluer des propositions de gameplays pour de la complétion à 2

éléments manquants, exemples de question : $3 \times ? = ?$, $? \times ? = 15$, $? \times 5 = ?$, $? = ? \times 5$, $? = 3 \times ?$ ou encore $15 = ? \times ?$.

19. L'image présente trois gameplays de type sélection de multiples objets. *

Dans chacun, le joueur doit sélectionner les bonnes réponses.

La porte s'ouvre lorsque l'ensemble des bonnes réponses ont été données. En cas d'erreur, le joueur perd une vie.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

OUI

NON

20. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

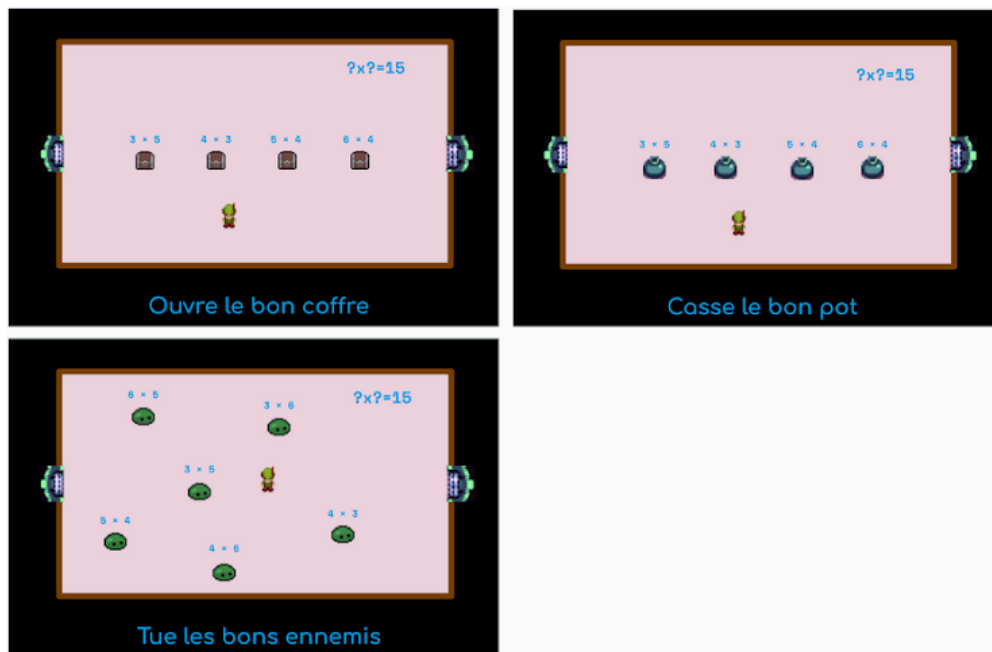
21. L'image présente trois gameplays de type sélection unique d'objet. *

Pour chacun, le joueur doit sélectionner la bonne réponse.

Chaque réponse présente les deux valeurs manquantes : $x * y$ (nous pourrions imaginer d'autres représentations ex : $[x, y]$).

La porte s'ouvre lorsque la bonne réponse a été donnée. En cas d'erreur, le joueur perd une vie.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

- OUI
 NON

22. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

23. L'image présente un gameplay de sélection multiple. *

Le joueur doit choisir, pour chaque image de l'énoncé (image de pot, coffre), l'élément physique (pot, coffre) correspondant.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

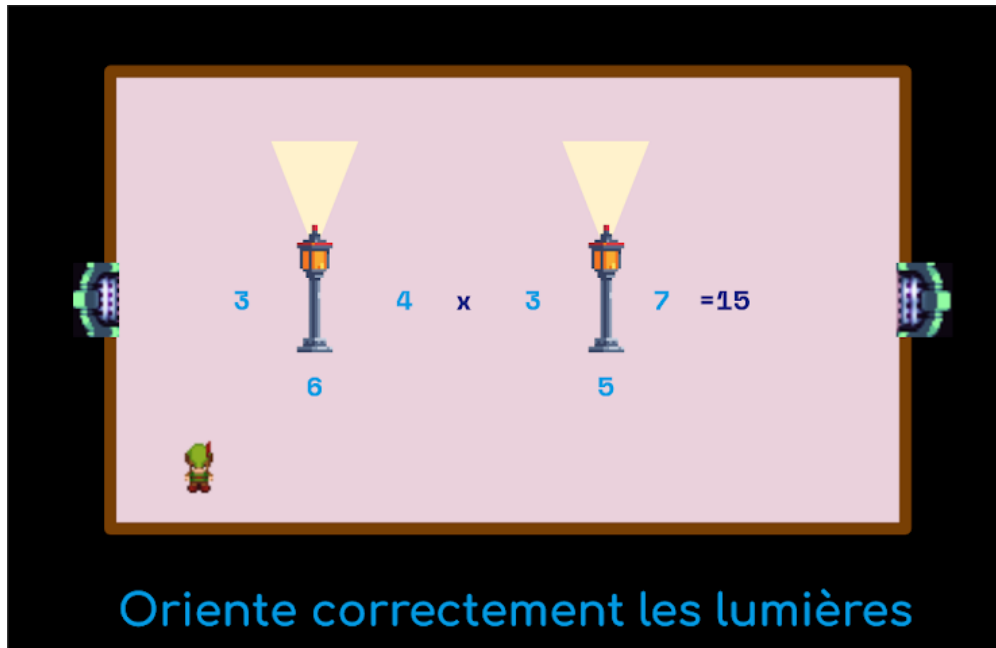
- OUI
- NON

24. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

25. L'image présente un gameplay de sélection multiple. *

Le joueur doit orienter chaque lumière sur la bonne position.
La validation se fait lorsque l'avatar s'éloigne des deux éléments.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

26. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

27. L'image présente un gameplay de déplacement. *

Le joueur doit positionner un pot sur chaque bonne réponse.
Si les deux bonnes réponses sont sélectionnées, la porte s'ouvre.
Lorsque le joueur place un pot sur une mauvaise réponse, il perd une vie.
(il ne sera pas proposé d'autres combinaison gagnante que celle attendue)
Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

NON

28. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

29. L'image présente un gameplay de déplacement multiple. *

Le joueur doit placer les bons pots sur les dalles dans l'ordre (= dalle de gauche → "?" de gauche et inversement).

Lorsque les deux dalles contiennent un pot, les valeurs sont vérifiées. Si le résultat est faux, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
 NON

30. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

31. L'image présente un gameplay de positionnement de l'avatar. *

Si la réponse est la bonne la porte s'ouvre, sinon le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
 NON

32. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

Reconstitution

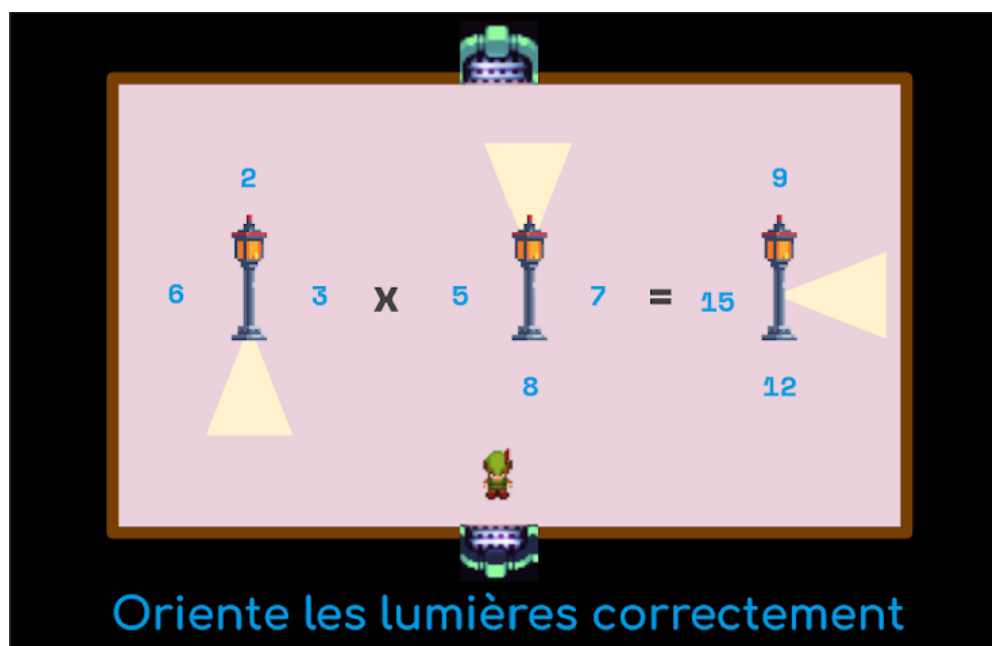
Dans cette troisième section nous vous demandons d'évaluer des propositions de gameplays pour de la reconstitution de faits, exemple de question : $? \times ? = ?$.

33. L'image présente un gameplay d'orientation. *

Pour chaque lumière, le joueur doit orienter cette dernière vers la bonne réponse.
La validation s'effectue une fois que l'avatar s'est éloigné et que chaque lumière est orientée vers un nombre.

Si les trois réponses sont bonnes, la porte s'ouvre, sinon le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



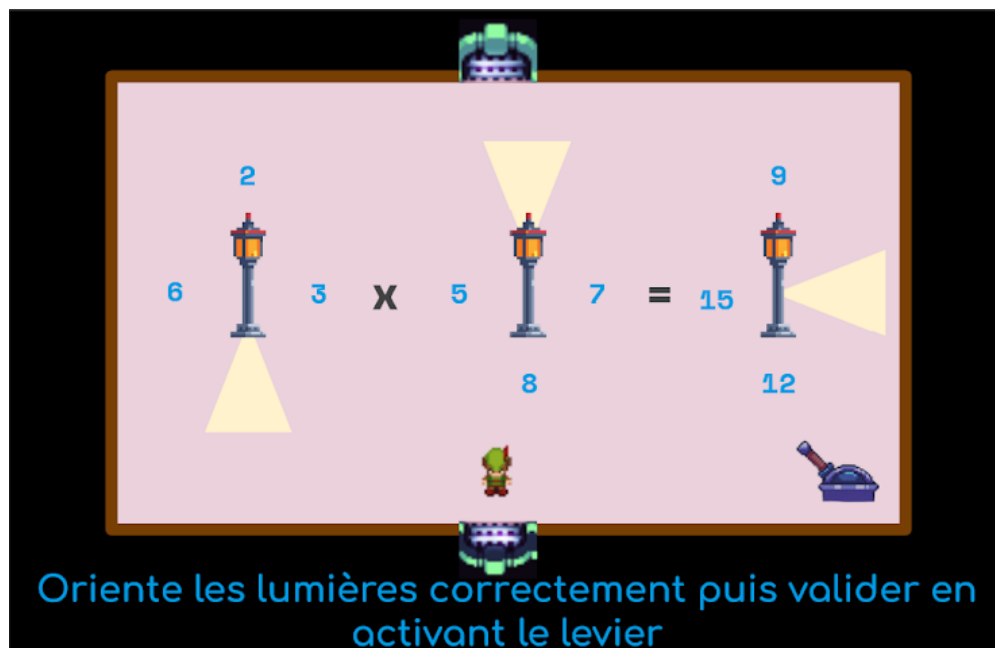
Une seule réponse possible.

- OUI
 NON

34. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

35. L'image présente le même gameplay. *
La différence est au niveau de la validation de la réponse qui s'effectue ici par action de l'avatar (activation du levier après l'orientation des 3 lumières).
L'ouverture de la porte s'effectue si la réponse validée est correcte sinon le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

36. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

37. L'image présente un gameplay de déplacement multiple. *

Le joueur dépose un pot dans chaque zone (carré gris) de la question.

Après le dépôt des 3 pots, si le résultat est faux, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

NON

38. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

39. L'image présente un gameplay de sélection multiple. *

En fonction de chaque type d'élément manquant (statue carrée, tonneau, statue), le joueur doit toucher l'élément de chaque type correspondant (la valeur s'affiche au-dessus de la position dans la question).

À la fin des 3 sélections, si le résultat est faux, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

NON

40. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

41. L'image présente le même gameplay que juste avant. *

La nuance est que le joueur doit valider la réponse en actionnant le levier. Puis, si le résultat (des sélections) est faux, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

42. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

Validation de faits

Dans cette quatrième section nous vous demandons
d'évaluer des propositions de gameplays pour de la validation de faits, exemple de question : $3 \times 5 = 15$, vrai
ou faux ?

43. L'image présente quatre gameplays de sélection unique. *

Le joueur doit choisir (en ouvrant, cassant, touchant ou passant par) la bonne réponse.
Si la réponse est fausse, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

- OUI
 NON

44. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

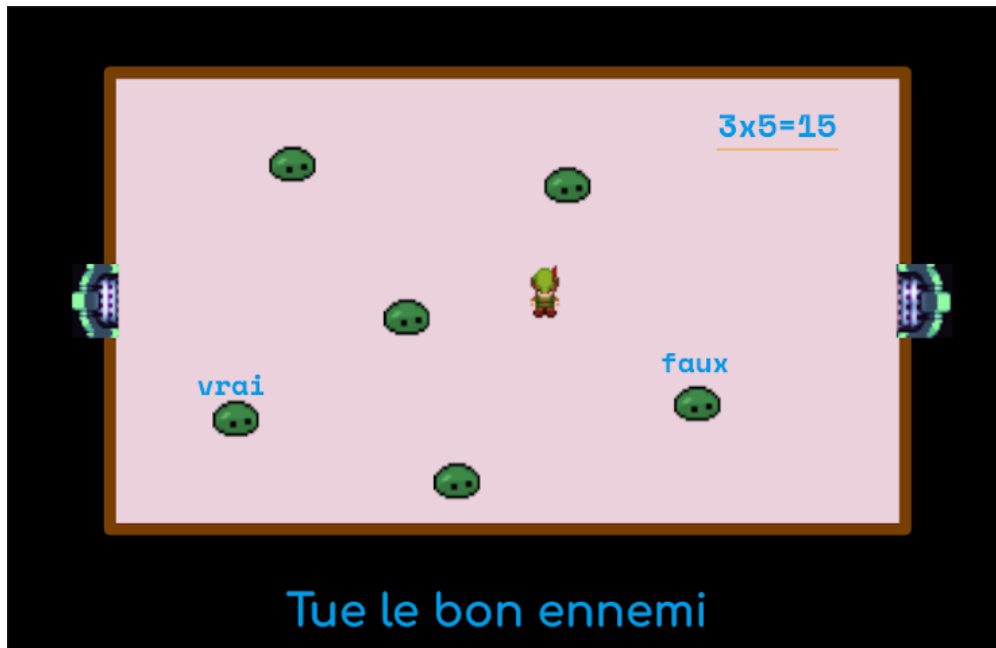
45. L'image présente un gameplay de sélection unique. *

Le joueur doit tuer l'ennemi portant la bonne réponse.

Certain ennemi sont présents uniquement pour la difficulté de jeu et ne portent pas de choix.

Si l'ennemi tué porte la mauvaise réponse, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

NON

46. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

47. L'image présente un gameplay de positionnement unique. *

Le joueur doit se positionner sur la dalle portant la bonne réponse.
S'il choisit la mauvaise dalle, il perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
 NON

48. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

49. L'image présente deux gameplays de déplacement unique. *

Le joueur doit déplacer un objet sur la dalle portant la bonne réponse.

Si la mauvaise dalle détecte un objet, le joueur perd une vie. Si la dalle portant la bonne réponse détecte un objet (pot, statue) la porte s'ouvre.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

- OUI
 NON

50. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

51. L'image présente deux gameplays de sélection multiple. *

Le joueur doit sélectionner (en ouvrant, en cassant) les objets portant une réponse valide.

En cas de mauvaise sélection, le joueur perd une vie.

Lorsque tous les bons éléments ont été sélectionnés, la porte s'ouvre.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

OUI

NON

52. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

53. L'image présente un gameplay de déplacement multiple. *

Le joueur doit pousser du bon côté chaque pot.

Lorsque tous les pots ont été poussés, les résultats sont vérifiés. En cas d'erreur, le joueur perd une vie.

Si tous les résultats sont bons, la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

54. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

55. L'image présente le même gameplay qu'avant. *

La nuance est que le joueur doit valider sa réponse en actionnant le levier pour lancer la vérification de la réponse (et donc la possible ouverture de la porte).

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

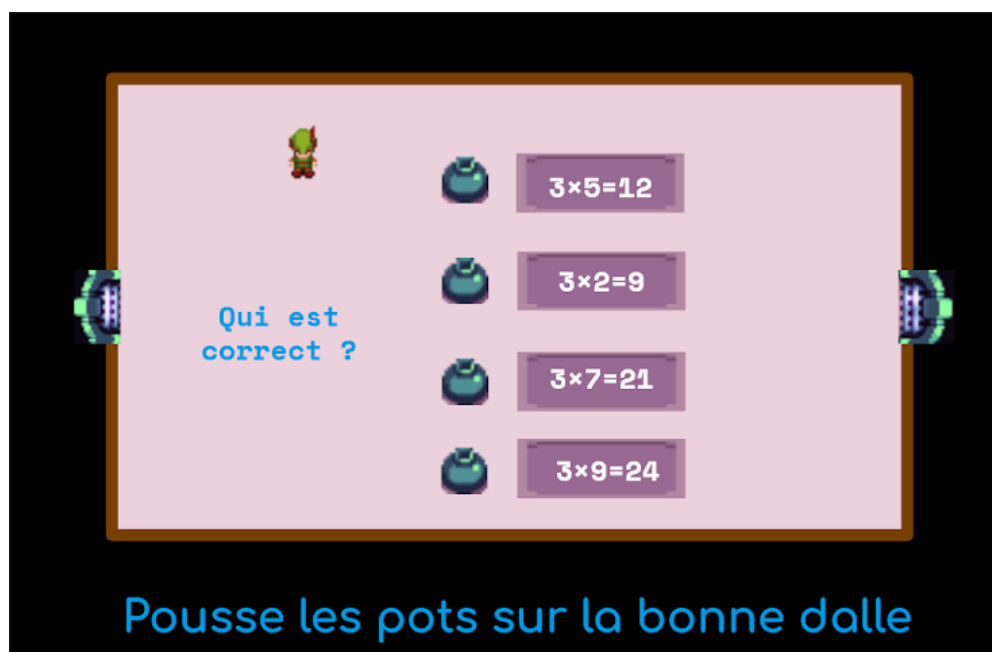
- OUI
- NON

56. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

57. L'image présente un gameplay de déplacement multiple. *

Le joueur doit pousser un pot sur les bonnes réponses uniquement.
Lorsque toutes les bonnes réponses auront un pot, la porte s'ouvrira.
Si un pot est poussé vers une mauvaise réponse, le joueur perd une vie.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

58. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

59. L'image présente le même gameplay qu'à la question précédente. *

Cependant, dans ce cas si, le joueur doit valider sa réponse après avoir déposé les pots.

La vérification n'a lieu qu'après que l'avatar ait actionné le levier.

En cas d'erreur, le joueur perd

Si le résultat est faux, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

NON

60. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

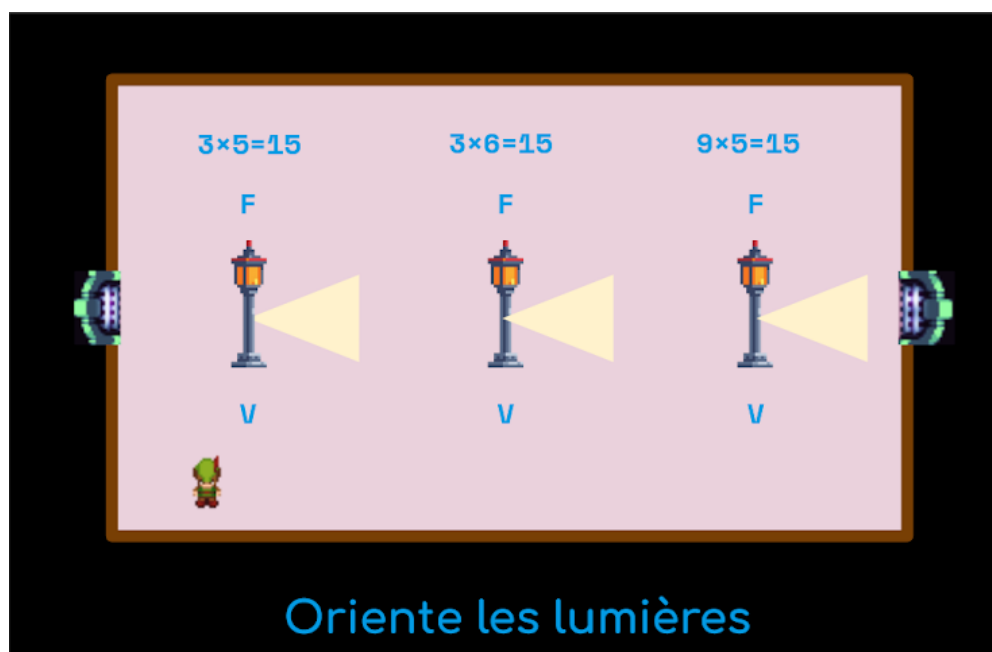
61. L'image présente un gameplay de multiple orientations. *

Le joueur doit orienter les lumières sur la bonne réponse puis s'éloigner.

À chaque erreur (après chaque orientation), si la réponse est fausse, le joueur perd une vie.

Lorsque toutes les lumières sont correctement orientées, la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

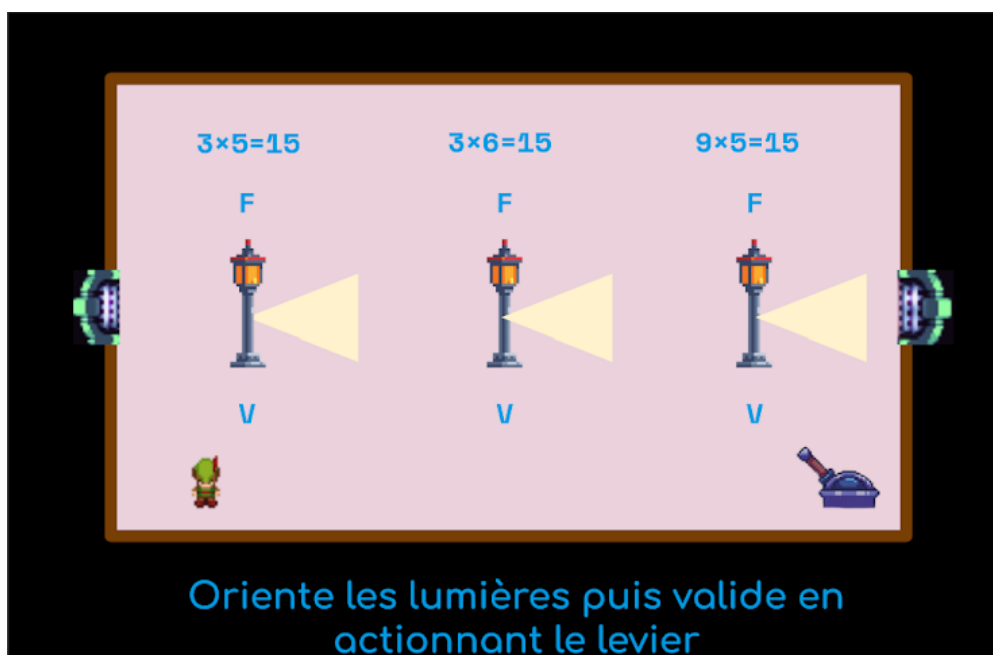
NON

62. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

63. L'image présente le même gameplay qu'à la question précédente. *

Ici la validation des résultats se fait en même temps après que le joueur ait actionné le levier.
En cas d'erreur, le joueur perd une vie. La porte s'ouvre lorsque toutes les lumières sont orientées correctement.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
- NON

64. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

Validation d'appartenance

Dans cette dernière section nous vous demandons
d'évaluer des propositions de gameplays pour de la validation de faits, exemple de question : parmi [3, 6, 8, 12,
13, 17, 21] lesquels sont des résultats de la table de 3 ?

65. L'image présente trois gameplays de sélection multiples. *

Le joueur doit sélectionner (en tuant, en cassant, en ouvrant) les bonnes ou mauvaises réponses selon l'énoncé.

Une fois les réponses attendues sélectionnées, la porte s'ouvre.

Lorsque le joueur commet une erreur, il perd une vie.

Trouvez-vous ces gameplays pertinents ?



Une seule réponse possible.

OUI

NON

66. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

67. L'image présente un gameplay de multiple orientations. *

Le joueur doit orienter le bouton vers une réponse correcte puis valider à l'aide du levier, puis répéter pour une autre réponse correcte.

Tous les résultats doivent être trouvés (multiples validations).

En cas d'erreur, le joueur perd une vie. Quand tous les résultats ont été trouvés, la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

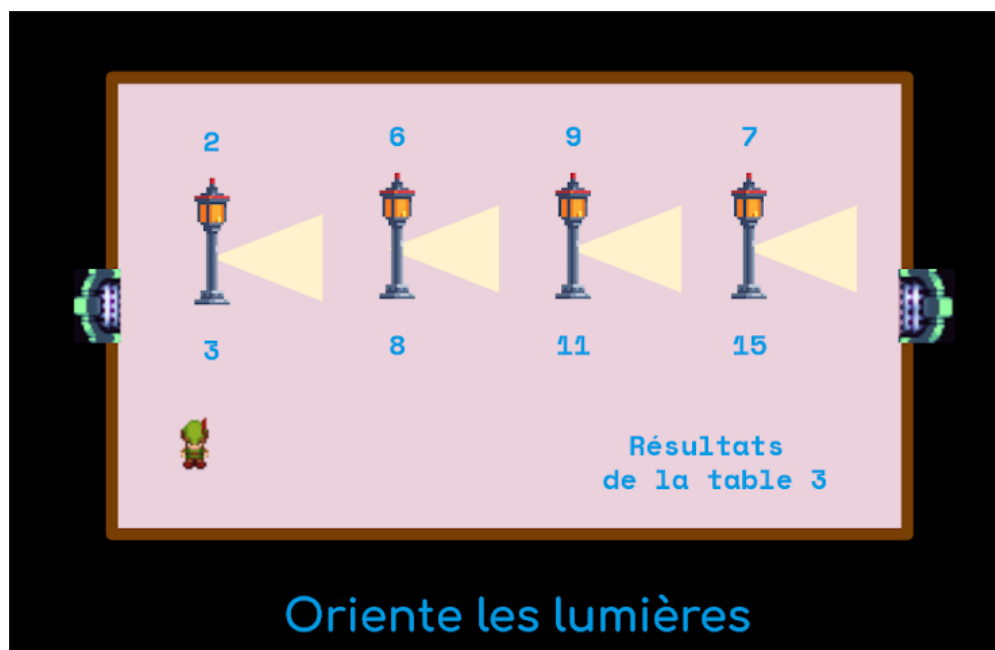
NON

68. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

69. L'image présente un gameplay de multiple orientation. *

Le joueur doit orienter les lumières vers les bonnes réponses puis s'éloigner des lumières.
Après chaque orientation, si la réponse donnée est fausse, il perd une vie.
Lorsque toutes les lumières sont correctement orientées, la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

OUI

NON

70. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

71. L'image présente le même gameplay qu'à la question précédente. *

Cependant, ici la vérification a lieu par action du joueur (levier) et la vérification se fait après avoir orienté toutes les lumières.

Si une erreur est présente lors de la vérification, le joueur perd une vie, sinon la porte s'ouvre.

Trouvez-vous ce gameplay pertinent ?



Une seule réponse possible.

- OUI
 NON

72. (Suite question précédente) Précisez votre réponse (surtout en cas de réponse négative).

Des idées ?

Il est possible qu'à la lecture des ces propositions vous ayez eu des idées de variations non présentes ou de nouveaux gameplays. Nous vous proposons de les préciser ci-après et/ou de dessiner vos idées et de nous les partager.

73. Auriez-vous des idées de gameplays possibles ? Si oui, décrivez-les ici.

Conclusion

Au cas où nous ayons besoin d'approfondir vos réponses, nous vous demandons de saisir votre nom/prénom et votre adresse mail.

Ces informations seront supprimées à la fin de l'analyse des données ou après vous avoir recontactés.

74. Votre nom et prénom

75. Votre adresse mail *

Ce contenu n'est ni rédigé, ni cautionné par Google.

Google Forms

ALGORITHM FOR GENERIC GENERATION OF TASK-ORIENTED GAMEPLAYS

Algorithm 1: GeneratePositionedElementsOfARoom (simplified)

```
1 Function createRoomElements(gameplay, facts, room, roomType):
2   for AComponent component: gameplay.getComponents() do
3     room.element.addAll(buildComponentElements(component, facts, [], roomType));
4 Function buildComponentElements(component, facts, listElements, roomType):
5   gameE ← findElementType(component.getAbility(), component.getSize());
6   position ← // get possible position from roomtype;
7   if component is Structure then
8     structure ← buildElement(component, facts, gameE, position);
9     listElements.add(structure);
10    for AComponent aComponent: component.getComponents() do
11      return buildComponentElements(aComponent, facts, listElements,
12        structure.newPosition());
12  else
13    return listElements.add(buildElement(component, facts, gameE, position));
14 Function buildElement(component, facts, gameElement, position):
15  listPositionedElement ← [];
16  if component is Structure then
17    if component.isPerFact then
18      for QuestionedFact qf: facts do
19        positionedElem1 ← // new positioned element displaying statement;
20        for ProposalParam proposal: qf.proposals do
21          positionedElem2 ← // new positioned element displaying proposal;
22          listPositionedElement.add(positionedElem2);
23        listPositionedElement.add(positionedElem1);
24    /* Only works for one fact questioned at a time*/
25    if component.isPerProposal then
26      for ProposalParam proposal: facts.get(0) do
27        positionedElem1 ← // new positioned element displaying proposal;
28        ositionedElem2 ← // new positioned element displaying element (e.g., tiles);
29        listpositionedElem.add(positionedElem1, positionedElem2);
30  else
31    if component.isPerProposal then
32      for QuestionedFact qf: facts do
33        for ProposalParam proposal: qf.proposals do
34          positionedElem ← // new positioned element displaying proposal;
35          listPositionedElement.add(positionedElem);
36    ...
37  else
38    positionedElem ← // new positioned element (e.g., enemies, traps);
39    listPositionedElement.add(positionedElem);
40  return listPositionedElement;
```

XMI TO XML CODE TRANSFORMATION IN ETL

```

1 pre {
2   "Running ETL".println();
3   var cpt = 1;
4 }
5
6 rule Dungeon2Dungeon
7   transform dIn: IN!Dungeon
8   to dOut: OUT!Dungeon {
9     "Dungeon created".println();
10    dOut.entryRoom = "Room[" + dIn.entry.x + "/" + dIn.entry.
11      y + "]";
12    // dOut.exitRoom = "Room["+ dIn.exit.x + "/" + dIn.exit.y
13      + "]";
14    dOut.objectiveID = dIn.learningobjective.id;
15    dOut.levelID = dIn.level.id;
16    dOut.gameLevelID = dIn.learnerPlayer.progression.
17      playerProgress.currentLevel;
18
19    if(dIn.curses.isDefined()) {
20      dIn.curses.println();
21      dOut.curses = "[";
22      for (curse in dIn.curses) {
23        curse.name.println();
24        dOut.curses = dOut.curses + "\"" + curse.name + "
25          \"\" + ",";
26        dOut.curses.println();
27      }
28      dOut.curses = dOut.curses + "]";
29    }
30
31    // ROOMS
32    var rooms: new OUT!Rooms;
33    dOut.rooms = rooms;
34    dOut.rooms.room ::= dIn.rooms;
35 }
36
37 rule Room2Room

```



```

34 transform roomIn: IN!Room
35 to roomOut: OUT!Room {
36     "\tRoom created".println();
37     roomOut.x = roomIn.x;
38     roomOut.y = roomIn.y;
39     roomOut.id = "Room[" + roomIn.x + "/" + roomIn.y + "];
40     roomOut.roomTypeName = roomIn.roomtype.name;
41
42     if(roomIn.task.isDefined()){
43         roomOut.idTask = roomIn.task.id;
44         roomOut.time = roomIn.task.maxTime;
45     }
46
47     roomOut.correctnessToReach = roomIn.nbExpectedAnswers+"";
48
49     if (roomIn.roomtype.isTypeOf(SmallRoomType)) {
50         //"\tsmall".println();
51         roomOut.size = OUT!RoomType#SMALL;
52     } else {
53         //"\tlarge".println();
54         roomOut.size = OUT!RoomType#LARGE;
55     }
56
57     if(not roomIn.questionedFacts.isEmpty()){
58         var facts: new OUT!Facts;
59         for (fact in roomIn.questionedFacts) {
60             var factOUT: new OUT!Fact;
61             factOUT.id = fact.questionablefact.id;
62             if(fact.question instanceof(QuestionParam)){
63                 factOUT.complete = fact.question.completeFact
64                 ;
65             }
66             if(fact.correctnessToReach.isDefined()){
67                 factOUT.correctnessToReach = fact.
68                 correctnessToReach.value.value;
69             }
70             if(fact.factCorrectness.isDefined()){
71                 factOUT.correctness = fact.factCorrectness.
72                 value.value+"";
73             }
74             facts.fact.add(factOUT);
75         }
76     }
77     roomOut.facts = facts;

```

```

76         "\t\t\tFacts created".println();
77         //roomOut.facts.facts := roomIn.questionedFacts;
78     }
79
80     if(roomIn.gameplay.isDefined()){
81         if(roomIn.gameplay.prompt.isDefined()){
82             roomOut.prompt = roomIn.gameplay.prompt.prompt;
83         }
84         roomOut.gameplayWithUndo = roomIn.gameplay.undoable;
85     }
86
87     if(not roomIn.positionedElement.isEmpty()){
88         var allelements: new OUT!Elements;
89         roomOut.elements = allelements;
90         roomOut.elements.element := roomIn.positionedElement
91         ;
92     }
93
94     // ROOM ACCESSES
95     var accesses: new OUT!RoomAccesses;
96     roomOut.roomaccesses = accesses;
97     roomOut.roomaccesses.roomaccess := roomIn.roomaccess;
98 }
99
100 rule Structures2Elements
101     transform element : IN!PositionedStructureElement
102     to elementOut : OUT!Element {
103         //"\t\t\tStrucutres creation".println();
104
105         elementOut.id = element.id;
106         elementOut.pos = element.position.id;
107         elementOut.type = element.elementType.type;
108
109         if(element.acceptedFacts.isDefined()){
110             for (soluce in element.acceptedFacts) {
111                 elementOut.acceptedFacts += soluce.value.value + ";";
112             }
113         }
114     }
115
116 rule MapStructures2Elements
117     transform element : IN!PositionedVisualizationElement
118     to elementOut : OUT!Element {
119         //"\t\t\tStrucutres creation".println();

```

```

120     elementOut.id = element.id;
121     elementOut.pos = element.position.id;
122     elementOut.type = element.elementType.type;
123
124     if(element.visualization.isDefined()){
125         elementOut.visualisation = element.visualization.id;
126     }
127 }
128
129 rule Elements2Elements
130     transform element : IN!PositionedElement
131     to elementOut : OUT!Element {
132         //"\t\t\tElements creation".println();
133
134         elementOut.id = element.id;
135
136         elementOut.type = element.elementType.type;
137         if(element.displays.isDefined()){
138             if(element.displays.size() == 1){
139                 elementOut.displayValue = element.displays.get(0).
140                     value.value + "";
141                 elementOut.interactive = element.displays.get(0).
142                     interactive;
143                 elementOut.isImageDisplay = element.displays.get(0).
144                     imageDisplay;
145             }
146             if(element.displays.size() > 1){
147                 for (display in element.displays) {
148                     elementOut.interactive = display.interactive;
149                     if(display.correctness.value.value ==
150                         ECorrectness#CORRECT){
151                         elementOut.displayValue = elementOut.
152                             displayValue + "[" + display.value.value +
153                             "];";
154                     } else {
155                         elementOut.displayValue = elementOut.
156                             displayValue + display.value.value + ";";
157                     }
158                 }
159                 elementOut.isImageDisplay = element.displays.get(0).
160                     imageDisplay;
161             }
162         }
163     }
164
165     if(element.correctness.isDefined()){

```

```

157         elementOut.correctness = element.correctness.value.value.
158             getName();
159     }
160     if(element.expectedAnswer.isDefined()){
161         for (answer in element.expectedAnswer) {
162             elementOut.correctAnswer = answer.value.value + ";";
163         }
164     }
165 }
166
167 if(element.elementType.ability.isDefined() and not(element.
168     elementType.ability.name == "DISPLAY")){
169     elementOut.ability = element.elementType.ability.name+"";
170 }
171
172 elementOut.pos = element.position.id;
173
174 if(element.fact.isDefined()){
175     elementOut.idFact = element.fact.questionablefact.id;
176 }
177
178 if(element.acceptedFacts.isDefined()){
179     for (soluce in element.acceptedFacts) {
180         elementOut.acceptedFacts += soluce.value.value + ";";
181     }
182 }
183
184 rule RoomAccess2RoomAccess
185     transform access: IN!RoomAccess
186     to accessOut: OUT!RoomAccess {
187         "\t\tRoomAccess created".println();
188         accessOut.direction = "" + access.direction;
189         accessOut.id = "a" + cpt++;
190         accessOut.accessRef = access.otherroomaccess.equivalent()
191             .id;
192     }
193
194 post{
195     "done".println();
196 }

```

GUIDELINES FOR EXTENDING THE FRAMEWORK

This appendix presents a step-by-step guide for extending the framework. First, it guides the specification of domain specific training tasks. Then, it outlines how to meta-model concepts (e.g., facts, questionable facts, tasks). Finally, it guides the implementation of extensions (i.e., generators of questionable facts and methods for instantiating questioned facts).

Framework Extension Guidelines

Updated 30 April, 2024

Our framework is a software infrastructure to guide the design and implementation of activity generator of Roguelite-oriented activities for declarative knowledge training. This is a step-by-step guide to help engineers extend our framework to specific didactic domains.

Our framework is based on the use of Model-Driven Engineering (MDE). It is implemented in Java and uses the Eclipse Modeling Framework (EMF) plugin. Hence, a robust understanding of both Java and EMF is highly recommended.

PREAMBLE: ROGUELITE & DECLARATIVE KNOWLEDGE

In our context, the produced generators provide Roguelite activities aimed at training declarative knowledge. Declarative knowledge training is specified through training path (cf. Figure 1) that explain teachers' or experts' vision on the progression of training. A training path consists of a set of objectives (e.g., "Training on the multiplication table of 2") ordered by prerequisite relationships. Each objective targets a set of facts to work on (i.e., targeted knowledge) and is broken down into progressive levels, which are themselves broken down into training tasks.

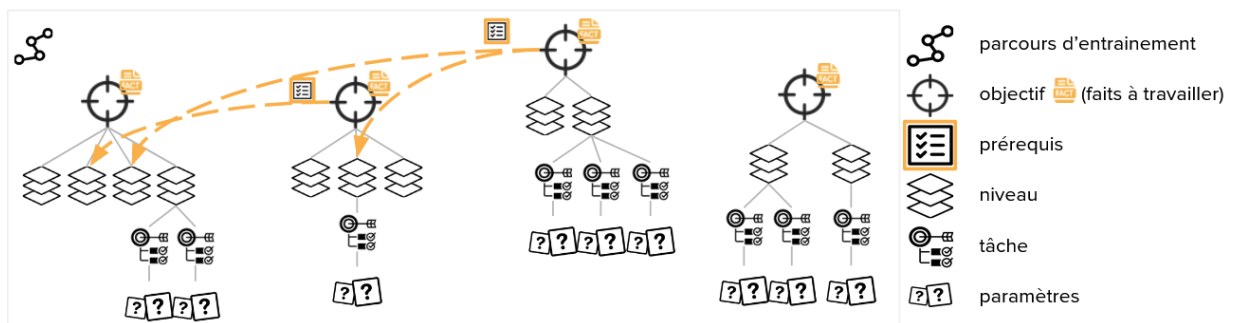


Figure 1: Training Path

Roguelite activities for declarative knowledge training are dungeons, i.e., interconnected rooms where an avatar explores and answers questions through task-oriented gameplays. For example, the avatar has to rotate a statue to answer $3 \times 5 = ?$ or push a block to say if $3 \times 5 = 12$ is true or false or break pots to select the results of the table 3 between $\{3, 5, 9, 12, 14, 17\}$.

STEP 1 DIDACTIC DOMAIN: GENERAL SPECIFICATIONS

The initial step in extending the framework involves studying the didactic domain in order to: 1) define training tasks, 2) specifying tasks' parameters/attributes, and 3) outlining the domain's facts. Indeed, we address facts last because the modelling of facts will take on a different form based on the tasks intended to be accomplished. This initial phase is the most challenging, crucial, and least guided as it heavily relies on the specific targeted domain.

SUB-STEP 1.1 TRAINING TASK DEFINITION

Our framework propose 4 training task types:

1. COMPLETION consists in completing facts having missing elements (e.g., $? \times ? = 12$)
2. IDENTIFICATION consists in identifying whether facts are true or false (e.g., $3 \times 5 = 12$?)
3. MEMBERSHIP IDENTIFICATION consists in identifying elements that share a given property (e.g., {3, 6, 7, 5, 9} select results of the multiplication table 3)
4. ORDERING consists in ordering facts using a heuristic (e.g., {Mars, Earth, Jupiter, Neptune} order starting from the closest to the sun)

In order to create your extension, you need to build subtypes of these tasks specific to your targeted didactic domain. COMPLETION and MEMBERSHIP IDENTIFICATION can be graphic tasks. Graphic tasks are tasks that require visual representation to answer (e.g., maps, pictures to legend). It is important to note that graphic tasks are quite similar independently of their type. The main difference lies in the fact that a completion question rest on one raw fact, while a membership question rest on multiple raw facts simultaneously. Consequently, in a room of a dungeon only one question of a membership task can be asked while multiple questions of a completion task can be asked (warning: number of question \neq number of expected answers).

For each task type, different questions need to be answered:

- COMPLETION
 - Is it a graphic task?
 - How many elements are missing? (e.g., one, two, three, every element of a fact)
 - Which elements are missing? (e.g., in a historical date, is it the event or the date?)
 - What is the shape of a question on a fact? The questions must replace missing elements by question marks, e.g., $3 \times 5 = ?$ or *World War II happened between ? - ?* or place the dates on the timeline (that contains the question marks or input areas).
 - How many facts are questioned at the same time?
 - Does the learner have to write down the answer? (i.e., only if the task is graphic or the number of fact questioned = 1 and number of missing elements = 1)
Else, how many bad propositions/choices are proposed to the learner?
- MEMBERSHIP IDENTIFICATION
 - Is it a graphic task?
 - What is the property concerned? (e.g., result of multiplication tables, categories of judo techniques, regions/cities)
 - Does the learner identify the elements that have the property or those that do not have it? (i.e., usually they identify the ones that have the given property, however in some cases it can be interesting to switch it)
 - What is the shape of a question on a fact? For example, results of table 3 with choices such as 3, 5, 6, 8, 9 or Figure 2

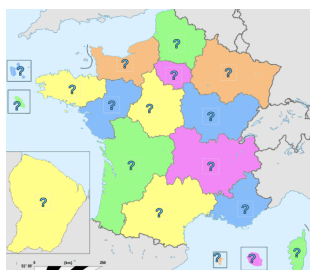


Figure 2: France Regions

- Does the learner have to write down the answer ?
Else, how many propositions/choices are provided to the learner? How many bad choices?
- How many answers are expected? (i.e., usually every correct answers, but not always)
- IDENTIFICATION
 - What is the shape of a question on a fact? Here questions are representation of facts, it is the propositions [true/false] that will explicitly express the question, e.g., $3 \times 5 = 15$.
 - How to build fake fact? Which element to replace in the fact by a fake one?
It is important to note that this task can be realised through an input response modality. However, we advise using a multiple choice modality for this particular task in our context (i.e., more gameplays \Rightarrow more variety).
 - How many facts are questioned at the same time?
- ORDERING
 - What elements must be ordered?
 - Using which order?
 - How many elements must be ordered?
 - How many facts must be ordered?

Regardless of the types, tasks share common attributes:

- The task's apparition percentage in a dungeon. Depending on the teacher's perspective, certain tasks may be favoured over others based on this percentage. However, the total percentage of all tasks at the same level (see Figure 1) must sum up to 100%.
- Is the answer's validation automatic or done by the learner? In some cases, the teacher may want the learner to validate their response rather than it being automatic, especially when multiple responses are required for a single fact (e.g., completing a fact with two missing elements).
- The maximum expected response time per question in seconds.
- The number of consecutive successes expected per fact. After how many consecutive successes is a fact considered mastered? This is required to progress in the training path.

COMPLETION	Number of facts questioned (per rooms)	—	
	Percent of apparition (per dungeon)	—	%
	Expected consecutive success (per fact questioned)	—	
	Maximum answer time	—	sec
	Answer modality	<input type="checkbox"/> CHOICE <input type="checkbox"/> INPUT	
	Number of bad choices	—	
	Learner validation	<input type="checkbox"/> TRUE <input type="checkbox"/> FALSE	
	Number of missing elements per facts	—	
	Missing elements are:	_____	
	Graphic task	<input type="checkbox"/> TRUE <input type="checkbox"/> FALSE	

(a) Completion task

MEMBERSHIP	Number of facts questioned* (per rooms)	1	
	Percent of apparition (per dungeon)	—	%
	Expected consecutive success (per fact questioned)	—	
	Maximum answer time	—	sec
		CHOICE	INPUT
	Answer modality	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	Number of bad choices	—	
		TRUE	FALSE
	Learner validation	<input type="checkbox"/>	<input type="checkbox"/>
	Number of expected answers	—	
		TRUE	FALSE
	Identification shared property	<input type="checkbox"/>	<input type="checkbox"/>
		TRUE	FALSE
Graphic task	<input type="checkbox"/>	<input type="checkbox"/>	
* one question multiple raw facts interogated			

(b) Membership Identification task

IDENTIFICATION	Number of facts questioned (per rooms)	—	
	Percent of apparition (per dungeon)	—	%
	Expected consecutive success (per fact questioned)	—	
	Maximum answer time	—	sec
		CHOICE	INPUT
	Answer modality	<input type="checkbox"/>	<input type="checkbox"/>
	Number of bad choices	1	
		TRUE	FALSE
	Learner validation	<input type="checkbox"/>	<input type="checkbox"/>

(c) Identification task

ORDERING	Number of facts questioned* (per rooms)	1	
	Percent of apparition (per dungeon)	—	%
	Expected consecutive success (per fact questioned)	—	
	Maximum answer time	—	sec
	Answer modality	CHOICE	<input checked="" type="checkbox"/>
	Number of good choices	—	
	Number of bad choices	—	
		TRUE	FALSE
	Learner validation	<input type="checkbox"/>	<input type="checkbox"/>
	Number of generated questions on facts	—	
* one question multiple raw facts interogated			

(d) Ordering task

Figure 3: Generic task minimal required parameters to specify.

SUB-STEP 1.2 DOMAIN SPECIFIC PROPERTIES

In order to build questions on facts, some domains (i.e., most domain do not) require specific parameters. For example, multiplication tables can be built in many ways such as: equal sign on the left or right, operand \times table or table \times operand. These parameters can be taken into

account at the “level” level and considered during the questioned facts’ generation.

SUB-STEP 1.3 FACTS & QUESTIONABLE FACTS DEFINITION

Now, raw facts must be designed based on the tasks defined. Think like an object-oriented developer, what composes your raw facts? For example, for multiplication tables a fact is a multiplication, i.e., three integers: the table, the operand, and the result. In another example, judo facts presenting technique are described by three strings and an image: a name, a description, and a category, and a visual representation of the technique. Whereas, judo facts presenting referee gestures are described by two strings and an image: a name, a description, and a visual representation of the gesture. Figure 4 presents these facts specifications. Note that fact such as cities on a map are at least characterized by: the type of element (e.g., city, region, county), the value (e.g., Paris, New York, Rome), the corresponding map, and their position on the map.

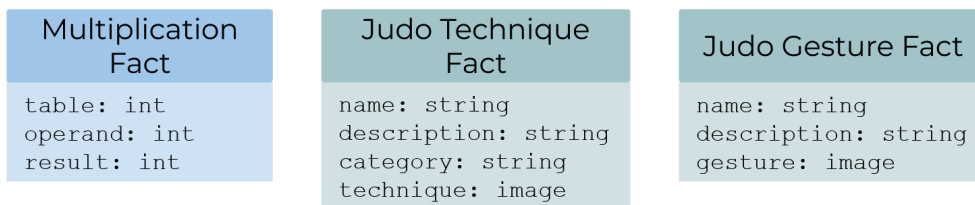


Figure 4: Raw facts’ specification examples.

In our work, questionable facts represents facts in their question form (i.e., question with their good propositions only). For each task, questionable facts have a different shape. What is the questionable fact shape for your tasks? For example, for multiplication tables, one task consists in completing a fact that has one missing element (i.e., either the table, the operand, or the result). For multiplication tables, the equal position can be on the left or the right. Therefore, questionable fact for this task have five attributes: “left operand”, “right operand”, “result”, “soluce”, and “resultOnRight”. The generator (to be developed subsequently) will instantiate these attributes based on task parameters. This process ensures that the missing element is set to -1, the other elements are assigned the correct values from the given fact, “soluce” is set to the value of the missing element, and “resultOnRight” is set to true or false based on task parameters. Another example, for multiplication tables, one task consists in identifying results of a given table (i.e., membership identification). For this task, one questionable fact is build based on multiple raw facts (i.e., it is the same for the Ordering task but different from other tasks), and two attributes are required: a set of good results, and the table (i.e., common property). In our context, bad propositions/choices of questioned facts (i.e., questions about facts present in dungeons) are computed by the algorithm. If you want to use metamodels to model bad propositions, you can. However, it is not guided, nor described in this tutorial. Figure 5 presents these two examples of questionable facts’ specification.

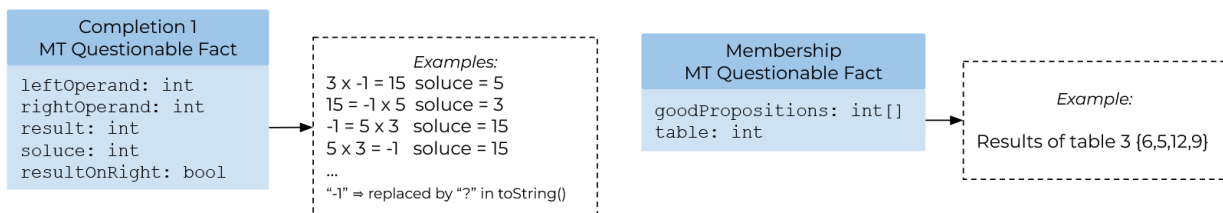


Figure 5: Questionable facts’ specification examples.

Questionable facts for graphic tasks (e.g., map to legend, map to complete) require a set of good results (i.e., position on the map/visualization and associated value which can be images or texts)

and the type of elements targeted. Positions and map or visualizations are elements dealt with that do not need to be extended.

STEP 2 METAMODELS EXTENSION

Now these specifications must be computerized by extending the generic metamodels. The file to modify is: *Generator > model > generator.ecore*. Our best advice to extend the framework is to create a new representation (see Figure 6): *mathExtension*, *judoExtension*, *historyGeoExtension* are examples you can refer to.

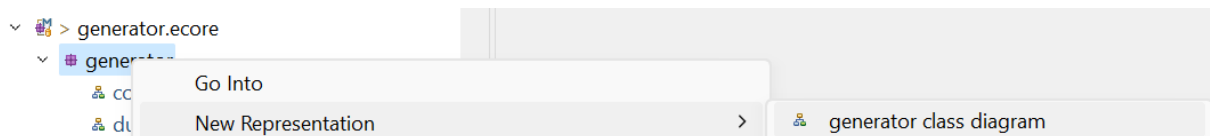


Figure 6: Create new representation.

SUB-STEP 2.1 TASKS CREATION

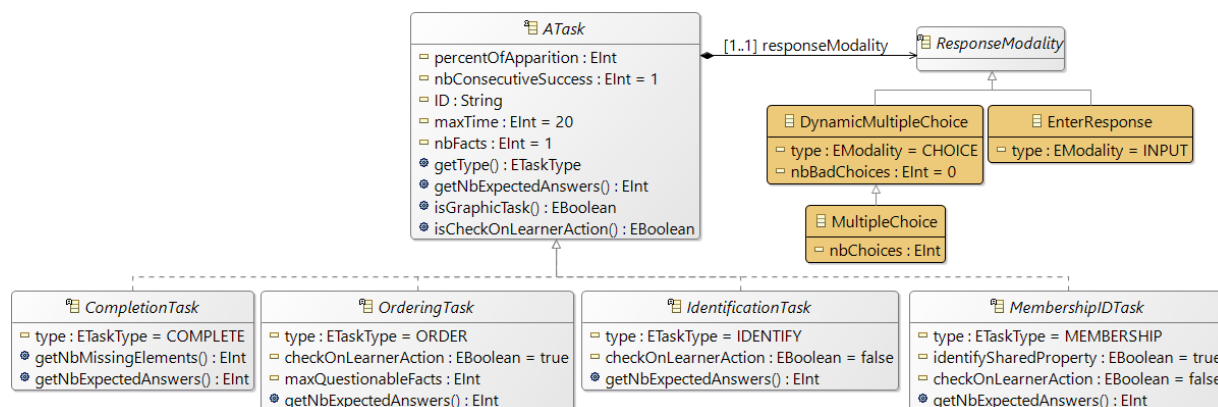


Figure 7: Tasks' metamodel to extend.

Each task previously specified must be translated into a metamodel concept, i.e., a subclass of each existing task. As it can be observed, most attribute previously mentioned are already modelled (e.g., apparition percentage, number of facts questioned). However, what is missing are elements depending on the specific didactic domains which must be added.

A Completion sub-task must declare attributes (warning: with exactly this typography):

- `nbMissingElements` of type `EInt` with the number of missing element for this type of sub-task as default value ;
- `checkOnLearnerAction` of type `EBoolean` with default value `true` or `false` (i.e., `true` is validation is done by learner action).

By default, `getNbExpectedAnswers` of `MembershipIDTask` return the number of good choices (from multiple choice answer modality). However, if the number of expected answers differs from the number of possible choices, the task must declare the attribute `nbExpectedAnswers` of type `EInt`.

When Completion and Membership sub-tasks are graphic, they must declare an attribute `graphicTask` of type `EBoolean` at `true`. Figures 8, 9 present examples of extension.

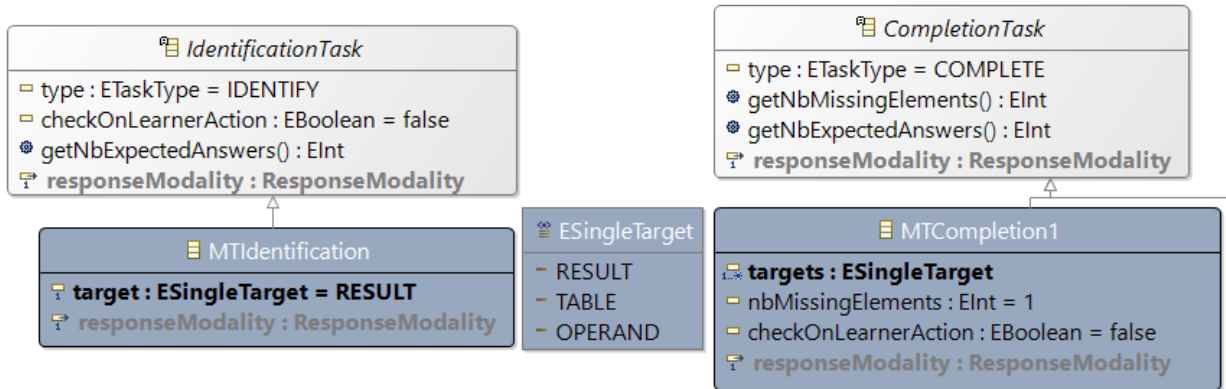


Figure 8: Extension Example.

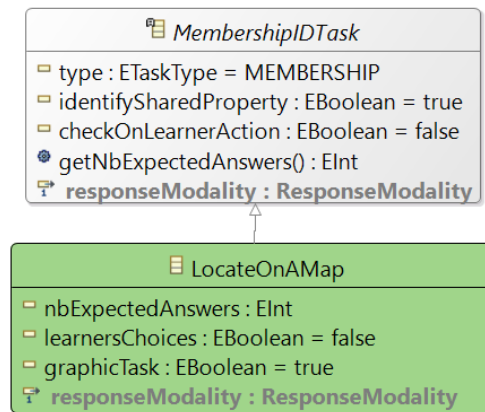


Figure 9: Extension Example 2.

SUB-STEP 2.2 LEVEL CREATION

Create a subclass of the class *Level*, add your domain specific parameters (if you have some). Figure 10 shows an example of extension of level with parameters and one without.

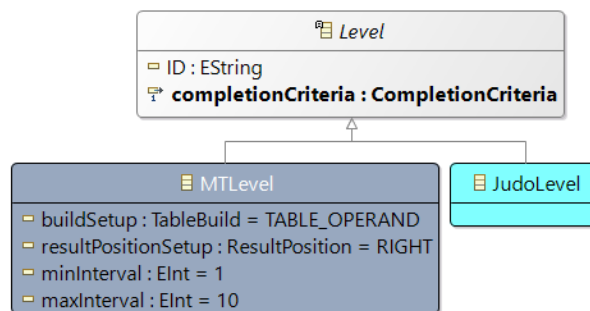


Figure 10: Level extension examples.

SUB-STEP 2.3 RAW FACTS CREATION

As you defined them in section SUB-STEP 1.3, create subclasses of *AbstractFact* (see Figure 11) to model your raw facts. Figure 12 presents examples of raw facts' extension. The first one describes multiplication facts, the second describes cities with a position on a map, the last describes judo techniques and arbitration gestures as previously presented. Note that references from

AbstractFact to VisualizationPosition are not part of the generic metamodel, as multiple positions on visualizations may be required by a fact for specific elements. For example, a fact for the “Legend a map” task requires a position for the symbol and one for the text, which you need to be able to identify independently (e.g., dedicated getter with dedicated name). In a simple list, this would be more difficult. Therefore, references to positions on visualizations (i.e., VisualizationPosition) needs to be added (c.f., GeographyFact attribute in Figure 12).

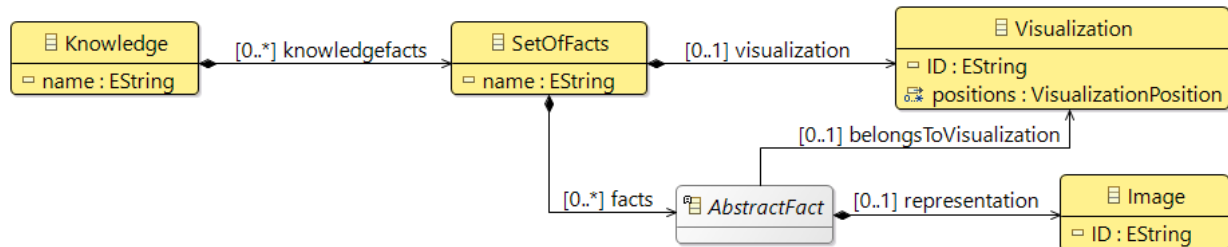


Figure 11: Raw facts’ metamodel to extend.

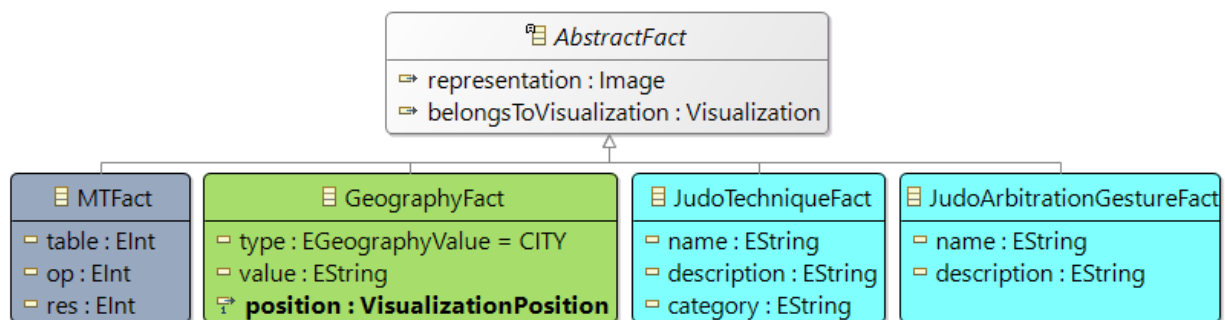


Figure 12: Raw facts’ extension examples.

SUB-STEP 2.4 QUESTIONABLE FACTS CREATION

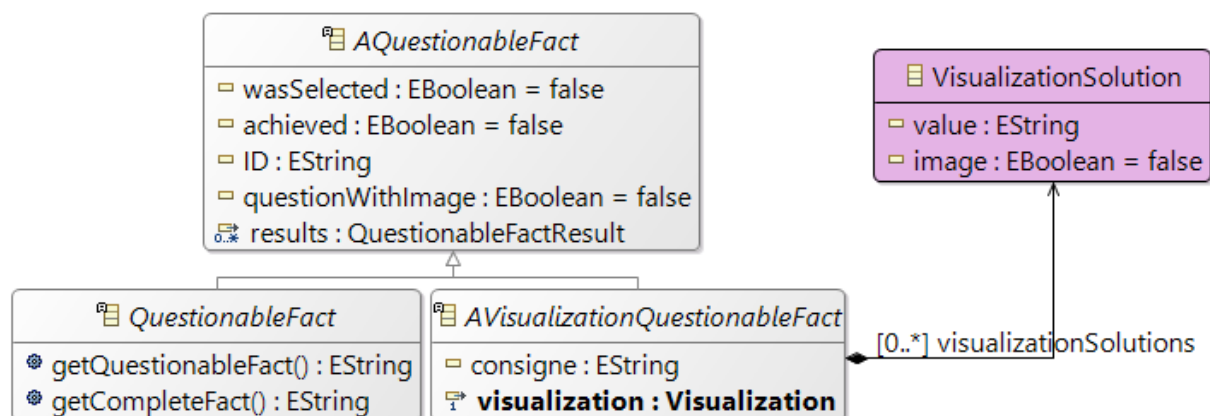


Figure 13: Questionable facts’ metamodel to extend.

As you defined them in section SUB-STEP 1.3, you need to create subclasses of *QuestionableFact* to model your textual questionable facts and *AVisualizationQuestionableFact* to model your graphical questionable facts (e.g., map facts), see Figure 13.

AVisualizationQuestionableFact contains a reference to the visualization targeted (e.g., map) and a set of *VisualizationPosition*. *VisualizationPosition* are composed of an

attribute value (i.e., string), an image (i.e., boolean) describing whether the value is an image, and a reference position to the position of the value on the map. Therefore, `AVisualizationQuestionableFact` **usually only requires an extra attribute for Membership task classification** (see Figure 14). Note that for `AVisualizationQuestionableFact`, **nothing must be added to deal with visualisation** (e.g., map) **and their position. The existing generic concepts** (i.e., `Visualization` and `VisualizationPosition`) **must be used!**

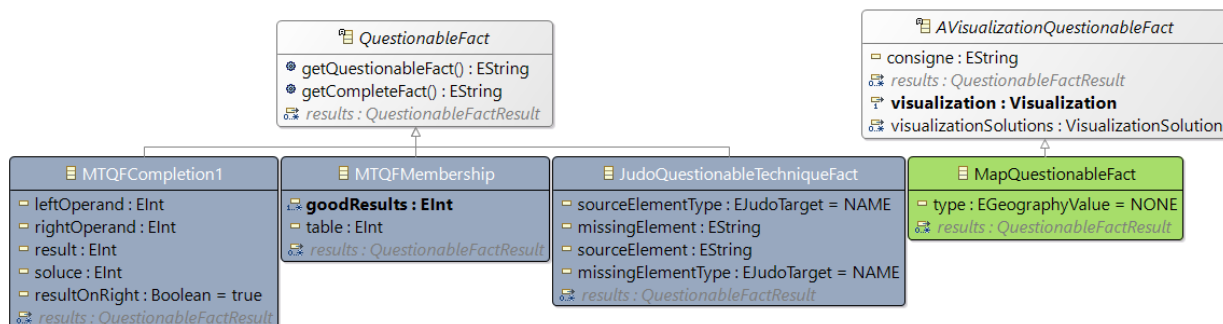


Figure 14: Questionable facts' extension examples.

Figure 14 shows examples of extensions of questionable facts. As a reminder, these facts are a form of domain-dependant “question on raw fact” with the set of good propositions (when the modality of the associated task is Multiple Choice).

SUB-STEP 2.5 MODEL CODE GENERATION & METHODS IMPLEMENTATION

Now you need to generate the code associated to the metamodels you created. To do that, *right-click (anywhere **on your** class diagram) > generate > model code.*

After generating model code, eclipse losses references to other projects (it's magical! 🪄). Therefore, you need to update the build path of the project. To that extent, *right-click on the root of the project > Build Path > Configure Build Path* (see Figure 15). Then, in the *Project* tab, click on *Classpath > choose the TransformationFlattener project > Ok > Apply and Close* (see Figure 16).

This procedure needs to be done each time you regenerate your model code.

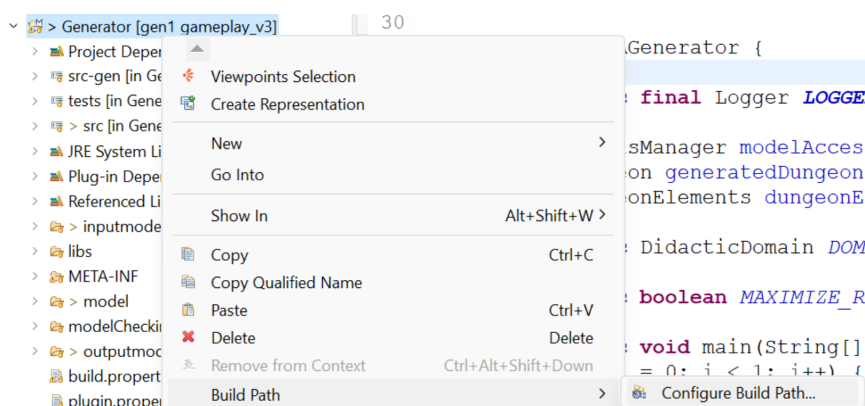


Figure 15: Configure build path step one.

Now, in the generated code, two methods of the questionable facts that inherit from `QuestionableFact` must be implemented:

- `getCompleteFact` that give a textual representation of the complete questioned fact with solutions ; **For order task, solutions must be ordered from left to right!** To that extent a

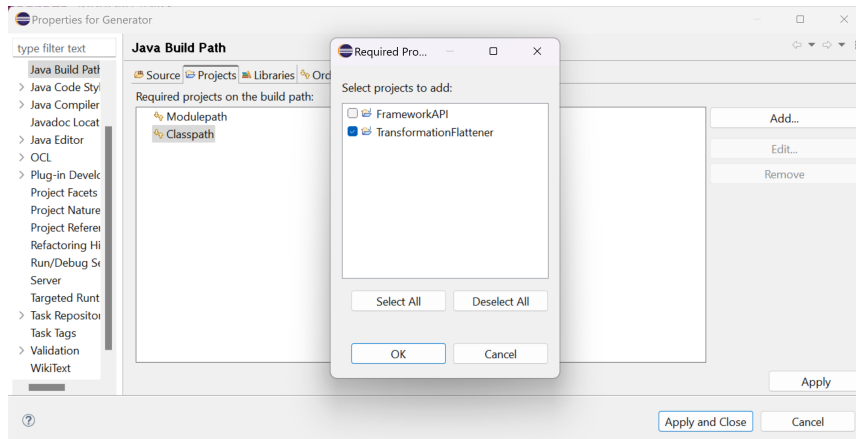


Figure 16: Configure build path step two.

template of sorting method is proposed (note that your questionable fact should have a set of solutions containing an order=integer).

- `getQuestionableFact` that give a textual representation of the questioned fact with "?" if there are missing elements.

For example, Ordering task for historical dates gives questions such as "Chronologically order", i.e., `getQuestionableFact()`. It also gives complete questions (i.e., question + answers) such as "Chronologically order: World War I, Treaty of Rome, Fall of Berlin Wall", i.e., `getCompleteFact()`, see Listing 1.

Listing 1: QuestionableFact Methods Implementation Example 1

```

1 public class OrderQuestionableFactImpl extends QuestionableFactImpl
2     implements OrderQuestionableFact
3 {
4     @Override
5     public String getQuestionableFact() {
6         return "Ordonner chronologiquement";
7     }
8
9     @Override
10    public String getCompleteFact() {
11        String fact = getQuestionableFact() + " : ";
12        int i = 0;
13        orderEvents();
14        for (HGOrderSolution sol : solutions) {
15            fact += sol.getEvent();
16            if (i < getSolutions().size() - 1) {
17                fact += " - ";
18            }
19            i++;
20        }
21        return fact;
22    }
23
24    "Warning! Simpler solutions exists but they create errors with EMF
25    pluggin.
26    In comments is put the elements that must be modify to corresond to your
27    own code in the text below.
28    * HGOrderSolution = type of Questionable fact "
```

```

26     private void orderEvents() {
27         Collections.sort(solutions, new Comparator</*HGOrderSolution*/>() {
28             public int compare(/*HGOrderSolution*/ o1, /*HGOrderSolution*/ o2
29                 ) {
30                 return o1.getOrder() - o2.getOrder();
31             }
32         });
33     }

```

As another example, Listing 2 present an example for a completion task where the complete fact can have a shape such as $2 \times 3 = 6$, $6 = 2 \times 3$, etc. and the questionable fact can have for shape: $2 \times ? = 6$, $6 = 2 \times ?$, etc.

Listing 2: QuestionableFact Methods Implementation Example 2

```

1 public class MTQFCompletion1Impl extends QuestionableFactImpl implements
2     MTQFCompletion1
3 {
4     @Override
5     public String getQuestionableFact() {
6         String left = getLeftOperand() == -1 ? "?" : getLeftOperand() + "";
7         String right = getRightOperand() == -1 ? "?" : getRightOperand() + "";
8         String res = getResult() == -1 ? "?" : getResult() + "";
9
10        if (resultOnRight) {
11            return left + " x " + right + " = " + res;
12        } else {
13            return res + " = " + left + " x " + right;
14        }
15    }
16
17    @Override
18    public String getCompleteFact() {
19        String left = getLeftOperand() == -1 ? getSoluce() + "" :
20            getLeftOperand() + "";
21        String right = getRightOperand() == -1 ? getSoluce() + "" :
22            getRightOperand() + "";
23        String res = getResult() == -1 ? getSoluce() + "" : getResult() + "";
24
25        if (resultOnRight) {
26            return left + " x " + right + " = " + res;
27        } else {
28            return res + " = " + left + " x " + right;
29        }
30    }
31 }

```

SUB-STEP 2.6 CREATE MODELS

Now you need to create several models, at least: the knowledge model and a learner-player model. To that extent, you need to create a **dynamic instance from the root object concerned by the model**. First, for the knowledge model, use the root concept Knowledge (see Figure 17) and for the learner-player model use `LearnerPlayer`.

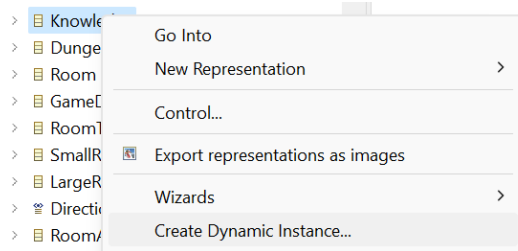


Figure 17: Create a model.

Give a name to the XMI file (e.g., “MultiplicationFacts”, “JudoFacts”) and place it in the repertory *inputmodels*. Now, you can instantiate elements in the “tree-view” provided. Figure 18 gives an example of knowledge model.

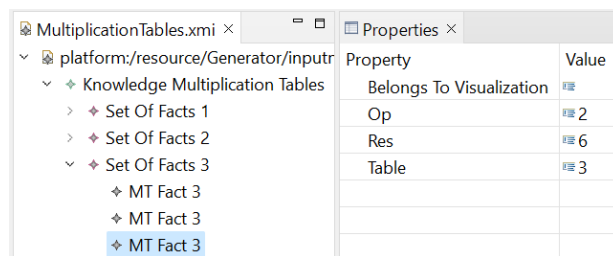


Figure 18: Knowledge Model Example.

Since **models can be linked** (i.e., some models use concepts of other models), **you can load a model inside another** one by doing: *right-click (anywhere in the tree-view) > Load Resource... > (browse the concerned model)*.

After creating a knowledge model, you need to create a learner player (i.e., same mechanism). However, **learner-player models must be placed in the sub-repertory** *inputmodels > learner-Players*.

Your final generator will need: a knowledge model, a learner-player model, a “learning domain” model (i.e., a model that contains training paths with the path of the learner) a game-model and a relation model. The learning domain already exists, its name is *LearningDomain.xmi*, and can simply be modified by adding your own path. You can also create a new one, however, it needs to be named *LearningDomain.xmi*.

In order to modify *LearningDomain.xmi*, you need to open the file, that looks as presented in Figure 19. First, you need to load your knowledge model previously created (using *Load Resource...*). Then, *right-click on the root of the tree-view Learning Domain > New Child > Learning Path*. Then, on the created child, add your objectives, levels, and tasks and specify their properties. **Once created, please remember to add the created learning path to the model of your previously created learners** (*Load Resource* ⇒ *LearningDomain.xmi* and specify learner-player parameter named *LeaningPath*)

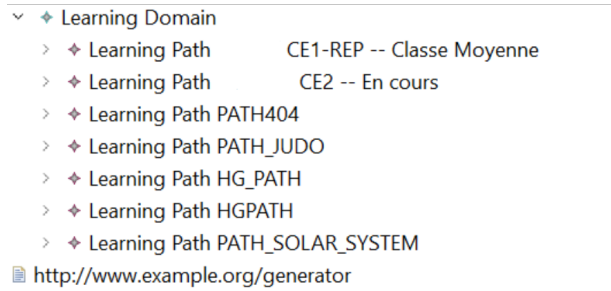


Figure 19: LearningDomain.xmi file.

Finally, the two last models (i.e., game and relations) are optional (default ones are proposed), however the first three are required.

STEP 3 QUESTIONABLE FACT GENERATORS

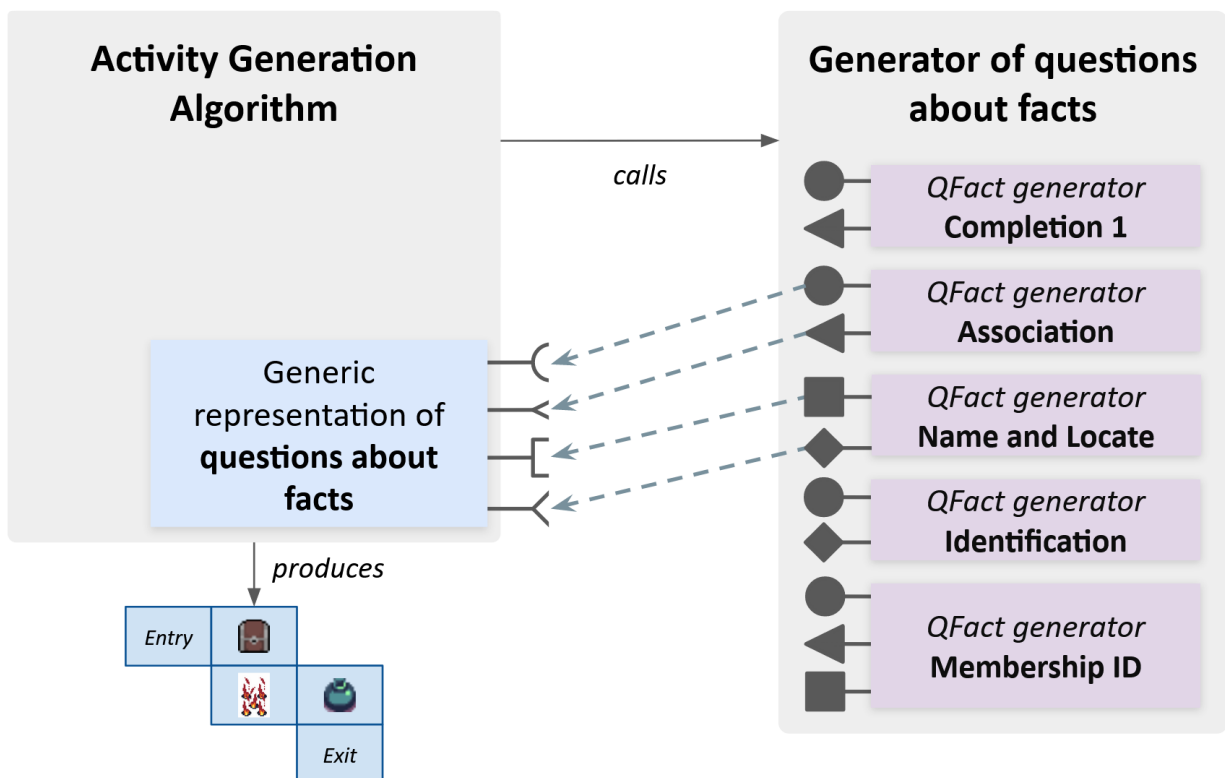


Figure 20: Principle of extension for questions on facts generations (template method pattern)

Questions on facts are built based on level and tasks parameters. Therefore, for each task created, an extension of the existing code must be made to deal with the corresponding questions on facts. The generation algorithm of questions about facts is based on a “template-method” design pattern, meaning that depending on the type of the task (i.e., Completion, Ordering, Identification, Membership Identification) a set of specific methods will have to be correctly implemented. The main idea of using such an approach is to provide a generic algorithm for gameplay generation,

as questions on facts have a generic representation (i.e., black box), and extension guides the instantiation of these generic form of questions by specifying your previously specified questionable facts and predefined methods. Figure 20 illustrate the principle.

First step, you need to create a class that extends `FactGeneratorTemplate` for each one of your tasks.

It is important to note that visual representations (maps, images) are dealt through their ID (i.e., string representations) throughout the entire process.

Some methods must be implemented independently of the task type, such as:

- `protected int correctnessToReach(AQuestionableFact fact)` which returns the number of expected answers per fact ;

```

1 @Override // LocateOnAMap
2 protected int correctnessToReach(AQuestionableFact fact) {
3     return ((AVisualizationQuestionableFact) fact).
4         getVisualizationSolutions().size();
5 }
6 @Override // MTCompletion1
7 protected int correctnessToReach(AQuestionableFact fact) {
8     return 1;
9 }

```

- `protected boolean isQuestionInteractive()` which return true if the question has interactive elements such as question marks that are replaced by their corresponding elements when a choice is made, else false ;

```

1 @Override // LocateOnAMap
2 protected boolean isQuestionInteractive() {
3     return false;
4 }
5 @Override // MTCompletion1
6 protected boolean isQuestionInteractive() {
7     return true;
8 }

```

- `List<Soluce> getListOfGoodSolutions(AQuestionableFact qFact)` which return the list of good solutions of a fact. `Soluce` is an object that can describe a proposition (value, position if it is on a map, if it is an image) ;

```

1 @Override // LocateOnAMap
2 protected List<Soluce> getListOfGoodSolutions(AQuestionableFact qFact) {
3     List<Soluce> solutions = new ArrayList<>();
4     for (VisualizationSolution prop : ((AVisualizationQuestionableFact)
5         qFact).getVisualizationSolutions()) {
6         solutions.add(new Soluce(prop.getValue(), prop.
7             getVisualizationPosition()));
8     }
9     return solutions;
10 }
11 @Override // MTCompletion1
12 protected List<Soluce> getListOfGoodSolutions(AQuestionableFact qFact) {
13     List<Soluce> solutions = new ArrayList<>();
14     solutions.add(new Soluce(((MTQFCompletion1) qFact).getSoluce()+""));
15     return solutions;

```

```
15 }
```

- `protected Map<ECorrectness, List<Soluce>> getListOfPropositions(ATask task, AQuestionableFact qFact)` which return a map with good and bad propositions of a fact. The following listing presents a template structure for this method.

```
1 @Override
2     protected Map<ECorrectness, List<Soluce>> getListOfPropositions(ATask
3         task, AQuestionableFact qFact) throws
4         BadSolutionGenerationException {
5         Map<ECorrectness, List<Soluce>> propositions = new HashMap<>();
6         List<Soluce> goodSoluces = getListOfGoodSolutions(qFact);
7         List<Soluce> badSoluces = new ArrayList<>();
8
9         // compute bad soluce as you wish
10
11         propositions.put(ECorrectness.CORRECT, goodSoluces);
12         propositions.put(ECorrectness.INCORRECT, badSoluces);
13
14         return propositions;
15     }
```

The constructor of the classes require at least one parameter of type `DungeonElements` and must call the parent constructor:

```
1 public HGFactGeneratorLocate(DungeonElements dungeonElements) {
2     super(dungeonElements);
3 }
```

In you want to use the raw facts to build bad choices, you can add another attribute in the constructor of type `ModelsManager`:

```
1 public HGFactGeneratorAssociation(ModelsManager modelsManager,
2     DungeonElements dungeonElements) {
3     super(modelsManager, dungeonElements);
4 }
```

You can access the `SetOfFacts` (i.e., group of raw facts) through:

```
modelsManager.getKnowledgeModel().getKnowledgefacts().
```

Now, depending based on the type of task, different methods must be implemented (for the others you can keep the *Auto-generated* versions).

SUB-STEP 3.1 QFACT GENERATOR FOR COMPLETION / IDENTIFICATION

One additional method must be implemented for these task to generate fact:

```
protected Set<AQuestionableFact> generateQuestionableFactsOf(ATask task,
AbstractFact fact). This method build from one raw fact all questionable facts associated to the raw fact based on the task parameters. As an example, for a fact  $3 \times 5 = 15$  for an identification task with multiple choice, this method will build two questionable facts:  $3 \times 5 = 15$  (true) and  $3 \times 5 = 12$  (false). Another example, for a completion task (i.e., result is missing, with multiple choice, equal on the right and left, construction is  $\text{table} \times \text{operand}$ ) and a fact  $3 \times 5 = 15$ , this method will return:  $3 \times 5 = ?$ ,  $? = 3 \times 5$ . Listing 3 presents an example of implementation of this method.
```

Listing 3: Example of generateQuestionableFactsOf implementation

```

1 @Override
2 protected Set<AQuestionableFact> generateQuestionableFactsOf(ATask task,
   AbstractFact fact) {
3     if(fact instanceof MTFact) {
4         MTCompletion1 taskC = (MTCompletion1) task; // Task parameters
5         MTFact factC = (MTFact) fact;
6         int min=((MTLevel)dungeonElements.getChosenLevel()).getMinInterval();
7         int max=((MTLevel)dungeonElements.getChosenLevel()).getMaxInterval();
8         if(min <= factC.getOp() && factC.getOp() <= max){
9             Set<AQuestionableFact> qfs = new HashSet<>();
10            TableBuild build = ((MTLevel) dungeonElements.getChosenLevel()).
11                getBuildSetup();
12            ResultPosition equalPos = ((MTLevel) dungeonElements.
13                getChosenLevel()).getResultPositionSetup();
14            for (ESingleTarget target : taskC.getTargets()) { // Questionable
15                facts creation
16                if(build.equals(TableBuild.MIX)) {
17                    if(equalPos.equals(ResultPosition.MIX)) {
18                        qfs.add(buildQF(factC, ResultPosition.LEFT,
19                            TableBuild.OPERAND_TABLE, target));
20                        qfs.add(buildQF(factC, ResultPosition.RIGHT,
21                            TableBuild.OPERAND_TABLE, target));
22                        ...
23                    }
24                } else {
25                    ...
26                }
27            }
28            return qfs;
29        }
30        return new HashSet<>();
31    }
32
33 private MTQFCompletion1 buildQF(MTFact fact, ResultPosition resPos,
   TableBuild build, ESingleTarget target) {
34     MTQFCompletion1 qf = new MTQFCompletion1Impl();
35     qf.setID(taskID+"-QAFAC"+factsCounter); factsCounter++; // MANDATORY
36     if(build.equals(TableBuild.OPERAND_TABLE)) {
37         qf.setLeftOperand(fact.getOp());
38         qf.setRightOperand(fact.getTable());
39     }else {
40         qf.setRightOperand(fact.getOp());
41         qf.setLeftOperand(fact.getTable());
42     }
43     qf.setResult(fact.getRes());
44     qf.setResultOnRight(resPos.equals(ResultPosition.RIGHT));
45     ....
46     return qf;
47 }

```

For every type of task, when you instantiate your questionable facts, this line:
qf.setID(taskID+"-QAFAC"+factsCounter); factsCounter++ is mandatory.

SUB-STEP 3.2 QFACT GENERATOR FOR MEMBERSHIP / ORDER

As previously mentioned, compared to completion and identification, questionable facts for membership and order task are built from multiple raw facts. Therefore, the method to override in order to create questionable facts is: `protected AQuestionableFact generateQuestionableFactOf(ATask task, List<AbstractFact> facts)`. Same as before, this method instantiates questionable facts based on task parameters (most of the time only one questionable fact is created) from a set of raw facts. Listing 4 presents an example of implementation of this method.

Listing 4: Example of `generateQuestionableFactOf` implementation

```
1 @Override
2 protected AQuestionableFact generateQuestionableFactOf(ATask task, List<
   AbstractFact> facts) {
3     MapQuestionableFact qf = new MapQuestionableFactImpl();
4     qf.setID(taskID+"-QAFACT"+factsCounter); factsCounter++;
5     qf.setVisualization(facts.get(0).getBelongsToVisualization());
6     qf.setType(((GeographyFact) facts.get(0)).getType());
7     for (AbstractFact fact : facts) {
8         VisualizationSolution soluce = new VisualizationSolutionImpl();
9         soluce.setValue(((GeographyFact) fact).getValue());
10        soluce.setVisualizationPosition(((GeographyFact) fact).getPosition())
11        ;
12        qf.getVisualizationSolutions().add(soluce);
13    }
14    if(task.getNbExpectedAnswers() == facts.size()) {
15        qf.setConsigne("Donner l'ensemble des reponses");
16    } else {
17        qf.setConsigne("Donner "+task.getNbExpectedAnswers()+" reponses");
18    }
19    return (AQuestionableFact) qf;
20 }
```

In addition, there can be conditions on the facts (e.g., type of facts targeted, specific facts attributes). In order to specify these conditions, you must implement the method: `protected boolean conditionForMembershipOrOrderTaskOnFacts(AbstractFact fact)`. This method can also be used for Order task. The following listing presents examples of this method for order and membership facts generator:

```
1 @Override // Membership
2 protected boolean conditionForMembershipOrOrderTaskOnFacts(AbstractFact fact)
   {
3     int min = ((MTLevel) dungeonElements.getChosenLevel()).getMinInterval();
4     int max = ((MTLevel) dungeonElements.getChosenLevel()).getMaxInterval();
5     return fact instanceof MTFact && min <= (((MTFact) fact).getRes()/((
   MTFact) fact).getTable()) && (((MTFact) fact).getRes()/((MTFact) fact)
   .getTable()) <= max;
6 }
7 @Override // Order
8 protected boolean conditionForMembershipOrOrderTaskOnFacts(AbstractFact fact)
   {
9     return fact instanceof HistoryFact;
10 }
```

For membership tasks, it is possible to sort the group of facts selected for the generation of questions about facts (e.g., multiplication table 1 is not interesting for membership makes no sense because every integer is a result of the table 1). To define the condition to sort the *SetOfFacts*, you must implement the method: `protected boolean conditionForMembershipTaskOnSetOfFacts (SetOfFacts setoffact)`. An example of implementation:

```

1 @Override
2 protected boolean conditionForMembershipTaskOnSetOfFacts (SetOfFacts setoffact
3     ) {
4     return !setoffact.getName().equals("1");
5 }

```

Finally, one last method for membership task is: `protected String getMembershipPropertyOfAFact (AbstractFact fact)`. This method returns the value of the “property” targeted by the task (e.g., the table for multiplication tables, the category for judo techniques). The following listing presents examples of implementations:

```

1 @Override // Property = Table
2 protected String getMembershipPropertyOfAFact (AbstractFact fact) {
3     return ((MTFact) fact).getTable()+" ";
4 }
5 @Override // Property = ID of the map they belong to
6 protected String getMembershipPropertyOfAFact (AbstractFact fact) {
7     return fact.getBelongsToVisualization().getID();
8 }

```

SUB-STEP 3.3 CONSIDER MULTIPLE SOLUTIONS (OPTIONAL)

In case you must deal (i.e., accept multiple solutions for one fact), you need to define through strings these solutions by overriding this method:

```
protected List<String> factSolutionsToString (AQuestionableFact qFact).
```

When solutions contains images, the images are replaced by their IDs in the strings representing the solutions. For example, multiplication are commutative and have two solutions. Therefore, this method was overridden to consider both solutions every time:

Listing 5: Example for a Completion 2 task for multiplication table training.

```

1 @Override
2 protected List<String> factSolutionsToString (AQuestionableFact qFact) {
3     List<String> solutions = new ArrayList<>();
4     MTQFCompletion2 qfact = (MTQFCompletion2) qFact;
5     if (qfact.isResultOnRight()) {
6         solutions.add(qfact.getSoluceLeft() + " x " + qfact.getSoluceRight()
7             + " = " + qfact.getSoluceRes());
8         solutions.add(qfact.getSoluceRight() + " x " + qfact.getSoluceLeft()
9             + " = " + qfact.getSoluceRes());
10    } else {
11        solutions.add(qfact.getSoluceRes() + " = " + qfact.getSoluceLeft() +
12            " x " + qfact.getSoluceRight());
13        solutions.add(qfact.getSoluceRes() + " = " + qfact.getSoluceRight() +
14            " x " + qfact.getSoluceLeft());
15    }
16    return solutions;
17 }

```

SUB-STEP 3.4 SORT FACTS CHOSEN (OPTIONAL)

Sometimes you might want to sort facts in order for them to never appear together in the same room (e.g., historical date that shares the same date). In order to define your own heuristic, you need to override this method: `protected List<AQuestionableFact> removeUnEligibleFactsBasedOnPreviouslySelectedFact (List<QuestionedFact> previousFacts, List<AQuestionableFact> facts)`.

This method gives you the previously selected facts and let you choose the fact that are still available in any way you wish. However, be careful to not be too restrictive, or the algorithm might not find any facts to generate any activity. Here is an example, that removes facts for which the solution (i.e., position on the chronological map) is equal:

Listing 6: Example for a Completion 2 task for multiplication table training.

```
1 @Override
2 protected List<AQuestionableFact>
   removeUnEligibleFactsBasedOnPreviouslySelectedFact (List<QuestionedFact>
   previousFacts, List<AQuestionableFact> facts) {
3     List<AQuestionableFact> eligible = new ArrayList<>();
4     for (AQuestionableFact qfact: facts) {
5         boolean conditionValide = true;
6         for (QuestionedFact fact: previousFacts) {
7             AVisualizationQuestionableFact prevF = (
8                 AVisualizationQuestionableFact) fact.getQuestionablefact ();
9             AVisualizationQuestionableFact newF = (
10                AVisualizationQuestionableFact) qfact;
11             for (VisualizationSolution sol1 : prevF.getVisualizationSolutions
12                ()) {
13                 for (VisualizationSolution sol2 : newF.
14                    getVisualizationSolutions()) {
15                     if (sol1.getValue().equals(sol2.getValue())) {
16                         conditionValide = false;
17                     }
18                 }
19             }
20         }
21         if (conditionValide) {
22             eligible.add(qfact);
23         }
24     }
25     return eligible;
26 }
```

STEP 4 FACTGENERATOR.JAVA EXTENSION

Next step consists in creating the link between our generator and your created question about facts generators. To that extent, you need to complete the class `FactGenerator.java` present in the package `factgenerator_template`.

First step, you need to create a method using this signature: `private static FactGeneratorTemplate getCorrect<domain>FactsGenerators (ModelsManager modelsManager, DungeonElements dungeonElements, ATask task)`. This method returns the correct instance of question about facts generator for a given task of your didactic domain. The following listing presents an example:

Listing 7: Example for a Completion 2 task for multiplication table training.

```
1 private static FactGeneratorTemplate getCorrectJudoFactsGenerators (
2     ModelsManager modelsManager, DungeonElements dungeonElements, ATask task)
3     {
4     FactGeneratorTemplate factGenerator;
5     switch(task.getType()) {
6     case COMPLETE:
7         if(task instanceof IdentifyTechnique) {
8             factGenerator = new JudoFactGeneratorIdentifyTechnique(
9                 modelsManager, dungeonElements);
10        } else {
11            factGenerator = new JudoFactGeneratorIdentifyArbitration(
12                dungeonElements);
13        }
14        break;
15    default:
16        factGenerator = new JudoFactGeneratorClassifyTechnique(modelsManager,
17            dungeonElements);
18        break;
19    }
20    return factGenerator;
21 }
```

Second, you need to call this method in `generateQuestionableFactsByTask` and `generateQuestionedFact` **by following the already existing convention!**

STEP 5 CALL YOUR KNOWLEDGE MODEL

Next step consists in giving the instruction to the generator to call the correct knowledge model when running. Therefore, in the class `ModelsManager.java` present in the package `managers` add a line such as: `didacticDomainFileNames.put(DidacticDomain.< YOUR DOMAIN >, < YOUR KNOWLEDGE FILE NAME >);`. This line must be added in between the `static` tag, such as:

Listing 8: Example for a Completion 2 task for multiplication table training.

```
1 static {
2     didacticDomainFileNames.put(DidacticDomain.MATHEMATICS, "
3     MultiplicationTables.xml");
4     didacticDomainFileNames.put(DidacticDomain.HISTORY_GEOGRAPHY, "
5     HistoryGeographyFacts.xml");
6     didacticDomainFileNames.put(DidacticDomain.JUDO, "JudoFacts.xml");
7 }
```

STEP 6 NAME YOUR DOMAIN & UPDATE TRANSFORMATION

Now, you need to name your domain. First, in the enumeration `DidacticDomain.java` in the package `structures` add a name for your domain. Then, in the class `ALGAGenerator.java` in the package `generators` change the variable `DOMAIN` to the value of your didactic domain name. Additionally, change the value of `FICTIF01` by the ID of the learner you want to generate a dungeon for in the line `generator = new ALGAGenerator("FICTIF01");` of the same class.

Listing 9: Extract of the main method of the generator.

```
1
2     public static DidacticDomain DOMAIN = DidacticDomain.MATHEMATICS;
3
4     public static void main(String[] args) {
5         for(int i = 0; i < 1; i++) {
6             ALGAGenerator generator;
7             try {
8                 generator = new ALGAGenerator("FICTIF01");
9                 generator.generate();
10                generator.printDungeon();
11                generator.saveDungeon("DungeonGen.xmi");
12                ....
13            }
14        }
```

Last step, you need to update the model `generator.ecore` used by the transformation project. Copy-paste the `generator.ecore` of the Generator project, in the model repertory of the Transformation project.

STEP 7 DEBUGGING & VERIFICATIONS

In case it does not work correctly, the first step is to verify that the questionable fast are correctly generated. To that extent, check in your learner-player models in the repertory *inputmodels > learnerplayers* in *Progression > LearnerProgress > CurrentObjectiveLevel > Results > ResultsByTasks > {the tasks in question}* that the questionable facts are presents and that their parameters are correctly instantiated. In the console, go to the beginning and check for any mistake indication that may come from the added code you implemented.

Debugging leads, if your questionable facts are not correctly generated (i.e., not present or incomplete in your learner-player model):

- check your XMI models for any mistake: Did you correctly associate a learning path to your learner-player? Does all of your task have a response modality? Are your identifier of path/objective/levels/tasks unique? (i.e., do not simply use *O1* for objective 1, our recommendation is to use a convention such as *PATHID+Oi* for objectives, *PATHID+OBJECTIVEID+Li* for levels and so on)
- verify that you implement every method required for your task type: one missing methods breaks the operation of the entire code.
- check your metamodels: Did you use a reference instead of a composition? Reference point to object created in other models, while composition implies that the elements are part of your current model. Therefore, using a reference instead of a composition can create errors.
- if you cannot locate the problem, our proposal is to create another path and adding each task by each task to locate which code fails.

Please remember that each time you generate questionable facts, if they are partially created (i.e., they have errors you want to correct), you must manually delete them from the model, otherwise they will not be regenerated!

Else, please contact us!

And if everything works, then it is done! Congratulations!

JUNIT TEST METHOD EXAMPLE

```

1 class GameElementsVarietyTest {
2
3     private ALGAGenerator generator;
4
5     @BeforeEach
6     void initDataSet() throws NonExistantLearnerPlayerException,
7         ContextNotFoundException {
8         generator = new ALGAGenerator(DidacticDomain.MATHEMATICS,
9             true, "LPGPELEM", "Contexts.xml", "GAMEPLAY_TEST");
10    }
11
12    @Test
13    void verifyGameElementByAbilitiesAreAllSelectedOnce() {
14        Map<String, List<ElementType>> map =
15            abilities2ElementTypes(abilities());
16        for(String s: map.keySet()) {
17            System.out.println(s+" : "+map.get(s));
18        }
19        while(!map.isEmpty()) {
20            Dungeon dungeon = generator.generate();
21            for(Room room: dungeon.getRooms()) {
22                map = removeElements(map, room);
23            }
24        }
25
26        private Map<String, List<ElementType>> removeElements(Map<
27            String, List<ElementType>> map, Room room) {
28            for(PositionedElement elem: room.getPositionedElement())
29            {
30                if(!elem.getDisplays().isEmpty() && !(elem.
31                    getElementType() instanceof Structure)) {
32                    if(map.containsKey(elem.getElementType().
33                        getAbility().getName())) {
34                        map.get(elem.getElementType().getAbility().
35                            getName()).remove(elem.getElementType());
36                    }
37                    if(map.get(elem.getElementType().getAbility().
38                        getName()).isEmpty()) {

```

```

30         map.remove(elem.getElementType().
31             getAbility().getName());
32     }
33 }
34 }
35 for (String key: map.keySet()) {
36     if(map.get(key).isEmpty()) {
37         map.remove(key);
38     }
39 }
40 return map;
41 }
42
43 private List<String> abilities() {
44     List<String> abilities = new ArrayList<>();
45     for(Ability ability: generator.getModelsManager().
46         getGameDescriptionModel().getAbilities().getAbilities()
47         ) {
48         abilities.add(ability.getName());
49     }
50     return abilities;
51 }
52
53 private Map<String, List<ElementType>> abilities2ElementTypes
54 (List<String> abilities) {
55     Map<String, List<ElementType>> map = new HashMap<>();
56     for(ElementType elem: generator.getModelsManager().
57         getGameDescriptionModel().getElements().getElementTypes
58         ().getElements()) {
59         if(elem.getNbDisplays() > 0 && !(elem instanceof
60             StatementElementType)) {
61             List<ElementType> types = new ArrayList<>();
62             types.add(elem);
63             if(map.containsKey(elem.getAbility().getName()))
64                 {
65                 types.addAll(map.get(elem.getAbility().
66                     getName()));
67             }
68             map.put(elem.getAbility().getName(), types);
69         }
70     }
71     return map;
72 }
73 }

```

MODEL VALIDATION SOURCE CODE (EVL)

```

1 context LearningDomain!LearningPath {
2     /*
3     Checks if every SetOfFacts of an Objective, belongs to the
4     corresponding Knowledge of the objective's path.
5     */
6     constraint objectiveFactsBelongsToPathKnowledge {
7         check {
8             return self.objectives.forAll(obj |
9             obj.setofacts.forAll(facts |
10            self.knowledge.knowledgefacts.contains(facts)));
11        }
12        message {
13            return "SetOfFacts does not belong to the correct
14            knowledge";
15        }
16    }
17 }
18
19 context LearningDomain!Objective {
20     /*
21     Checks if every required level of an Objective's Prerequisite
22     , is not a level of the actual objective.
23     */
24     constraint objectivePrerequisiteLevelsAreNotObjectiveLevels {
25         check {
26             return self.prerequisites.forAll(prereq |
27             not self.levels.contains(prereq.requiredLevel));
28        }
29        message {
30            return "Prerequisite of an objective references a
31            level the objective";
32        }
33    }
34 }
35
36 context LearningDomain!CompletionCriteria {
37     /*

```

```

34     Checks if every CompletionCriteria percentages are valid (
35         over 0 and under 100).
36     */
37     constraint completionCriteriaAreOver0AndUnder100 {
38         check {
39             return (self.successPercent >= 0 and self.
40                 successPercent <= 100) and
41                 (self.encountersPercent >= 0 and self.
42                     encountersPercent <= 100);
43         }
44     }
45 }
46
47 context LearningDomain!Prerequisite {
48     /*
49     Checks if every Prerequisite percentages are valid (over 0
50     and under 100).
51     */
52     constraint prerequisitesPercentageAreOver0AndUnder100 {
53         check {
54             return (self.successPercent >= 0 and self.
55                 successPercent <= 100) and
56                 (self.encountersPercent >= 0 and self.
57                     encountersPercent <= 100);
58         }
59     }
60     /*
61     Checks if every prerequisites are unlockable : its percentages
62     are inferior or equals to its requiredLevel percentages
63     */
64     constraint prerequisiteAreUnlockable {
65         check {
66             return (self.successPercent <= self.requiredLevel.
67                 completionCriteria.successPercent) and
68                 (self.encountersPercent <= self.requiredLevel.
69                     completionCriteria.encountersPercent);

```

```

67     }
68     message {
69         return "Prerequisite is not achievable (percentages
70             are superior to those of the level completion
71             criteria)";
72     }
73 }
74 context LearningDomain!MultipleChoice {
75     /*
76     Checks that multiple choice have a coherent number of bad
77     choices based on the total number of choices
78     */
79     constraint nbOfBadChoicesAreInferiorToNumberOfChoices {
80         check {
81             return self.nbChoices > self.nbBadChoices;
82         }
83         message {
84             return "MultipleChoice should have a number of
85                 choices superior (>) to the number of bad choices."
86             ;
87         }
88     }
89 }
90 context DungeonGen!Dungeon {
91     /*
92     Checks that the objective belongs to the learner-player path
93     */
94     constraint objectiveBelongsToLearnerPath {
95         check {
96             return self.learnerPlayer.learningpath.objectives.
97                 contains(self.learningobjective);
98         }
99         message {
100             return "Dungeon objective does not belong to learner
101                 training path.";
102         }
103     }
104     /*
105     Checks that the level belongs to the chosen objective
106     */
107     constraint levelBelongsToObjective {
108         check {

```

```






105         return self.learningobjective.levels.contains(self.
106             level);
107     }
108     message {
109         return "Dungeon level does not belong to the chosen
110             objective.";
111     }
112 }
113 /*
114 Checks that the tasks belongs to the chosen level
115 */
116 constraint tasksBelongsToLevel {
117     check {
118         return self.rooms.forAll(room | room.task == null or
119             self.level.tasks.contains(room.task));
120     }
121     message {
122         return "Dungeon tasks does not belong to the chosen
123             level.";
124     }
125 }
126 }
127
128 context DungeonGen!Room {
129     /*
130     Checks that the position of elements belongs to the room
131     types of to structures
132     */
133     constraint elementsPositionBelongsToRoomTypeOrStructure {
134         check {
135             return self.positionedElement.forAll(pe |
136                 self.roomtype.elementPositions.contains(pe.position)
137                 or pe.position.ID.contains("id/"));
138         }
139         message {
140             return "Elements positions does not belong to the
141                 chosen roomtype or structures presents in the room.
142                 ";
143         }
144     }
145 }
146 }
147 }

```


FRAMEWORK USABILITY EVALUATION QUESTIONNAIRE

Empty Questionnaire

XP Ingénieur Questionnaire

1. Avez-vous trouvé le guide suffisamment complet pour permettre d'étendre le *framework* ?
 - oui
 - non
2. Pensez-vous qu'une forte expérience en Ingénierie Dirigée par les Modèles pour réussir à étendre le *framework* ?
 - oui
 - non
3. Estimez la difficulté à réaliser l'extension :
 -  Très facile
 -  Assez facile
 -  Ni trop facile, ni trop dur
 -  Un peu trop dur
 -  Vraiment trop dur

4. Estimer la difficulté de manière détaillée :

Étapes	Très facile	Assez Facile	Ni trop facile, ni trop dur	Un peu trop dur	Vraiment trop dur
Compréhension des consignes	😊	😊	😐	😞	😞
Compréhension du domaine didactique	😊	😊	😐	😞	😞
Compréhension du fonctionnement du framework (parcours d'entraînement...)	😊	😊	😐	😞	😞
Extension des méta-modèles	😊	😊	😐	😞	😞
Extension des modèles	😊	😊	😐	😞	😞
Conception du code des générateurs de faits par tâche	😊	😊	😐	😞	😞
Intégration du code des générateurs de faits dans le code du générateur	😊	😊	😐	😞	😞
Débogage	😊	😊	😐	😞	😞



5. Commentaires, précisions, autres :











































Engineer Answers to the Questionnaire

XP Ingénieur Questionnaire

1. Avez-vous trouvé le guide suffisamment complet pour permettre d'étendre le *framework* ?
 - oui
 - non
2. Pensez-vous qu'une forte expérience en Ingénierie Dirigée par les Modèles pour réussir à étendre le *framework* ?
 - oui
 - non
3. Estimez la difficulté à réaliser l'extension :
 - 😊 Très facile
 - 😊 Assez facile
 - 😊 Ni trop facile, ni trop dur
 - 😞 Un peu trop dur
 - 😞 Vraiment trop dur

4. Estimer la difficulté de manière détaillée :

Étapes	Très facile	Assez Facile	Ni trop facile, ni trop dur	Un peu trop dur	Vraiment trop dur
Compréhension des consignes					
Compréhension du domaine didactique					
Compréhension du fonctionnement du framework (parcours d'entraînement...)					
Extension des méta-modèles					
Extension des modèles					
Conception du code des générateurs de faits par tâche					
Intégration du code des générateurs de faits dans le code du générateur					
Débogage					

5. Commentaires, précisions, autres :

Le guide est très complet et les consignes sont très bien expliquées. Il manquait juste à éclaircir et réorganiser quelques parties (dont on a déjà pris note). Le guide ne demande pas forcément à avoir une grande expérience en IDM tellement les étapes sont claires. Au final, j'ai très apprécié participer à cette expérimentation et j'ai trouvé ça très intéressant. Merci !

Title: Generation of Adapted Training Game Activities: a Model-Driven Engineering Design and Implementation Framework

Keywords: Procedural Generation – Adaptation – Modelling – Training – Declarative Knowledge – Model-Driven Engineering

Abstract: Procedural generation is a method widely used in video games to deliver varied content tailored to players. However, this method is rarely used in the field of Technology Enhanced Learning (TEL). In this PhD thesis, our focus is on the generation of game activities for declarative knowledge training (i.e., factual information such as laws and multiplication tables). In this context, it is necessary to provide learners with varied and adapted activities to avoid task drop-out caused by boredom. The scope of this thesis covers three angles of adaptation: the teacher's perspective on training, learner-players progression and players preferences.

This PhD work falls within the field of engineering research of TEL systems. The aim is to characterise the generation of activities and to propose a “generic” approach, i.e., independent of any specific didactic domain. Thus, the aim is to be able to reuse generation elements for different domains. This thesis is based on the AdapTABLES research project, which provides an initial ground of study and experimentation. The aim of this project is to design and develop a multiplication table training game. The research contribution (i.e., study and design methods) was developed in this context, but was also generalised and evaluated in other contexts.

First, we identified a game genre compatible with declarative knowledge training: the Roguelite. In this game genre, the activities

or game levels are procedurally generated and incorporate a high degree of variability. Repetition is encouraged by a “permanent death” mechanism. Then, we: 1) characterised and specified the generation by analysing the different adaptation needs (i.e., teacher, learner-player) and 2) proposed a framework (i.e., conceptual framework and software infrastructure) based on the principles of Model-Driven Engineering to design and implement generators for declarative knowledge training in the context of Roguelite oriented games. The generators designed are independent software components producing levels (i.e., dungeons) in XML format that can be interpreted by an educational game.

Three generators have been designed using the framework: one for multiplication training, a second for history and geography facts (i.e., required for the *Diplôme National du Brevet des Collèges*, a French exam taken in 9th grade) training, and a third for judo facts training. The multiplication tables generator is currently being used in an educational game designed as part of the AdapTABLES project. The framework and its components have been validated using system tests and semantic constraints validation of models, as well as experimentation with an engineer to assess the usability of the framework. Moreover, the game developed for the AdapTABLES project and the associated generator were used several times in ecological conditions.

Titre : Génération d'activités de jeux d'entraînement adaptées : un framework de conception et d'implémentation fondé sur l'ingénierie dirigée par les modèles

Mot clés : Génération Procédurale – Adaptation – Modélisation – Entraînement – Connaissances Déclaratives – Ingénierie Dirigée par les Modèles

Résumé : Ce travail de thèse s'inscrit dans une perspective informatique du domaine de recherche de l'ingénierie des EIAH (Environnements Informatiques pour l'Apprentissage Humain) dont les contributions visent à soutenir et à guider la conception pluridisciplinaire d'EIAH et des environnements supports associés. La recherche a montré que la mémorisation à court et long termes de connaissances déclaratives (e.g., lois, faits, règles) nécessite de la répétition. Cependant, celle-ci devient rapidement ennuyeuse pour les apprenants. Il a également été montré que proposer des activités de jeux répétitives et non adaptées aux apprenant-joueurs provoque un sentiment d'ennui. En conséquence, le contexte de la rétention à long termes de connaissances déclaratives nécessite de proposer aux apprenants des activités d'entraînement variées et adaptées pour éviter l'abandon des tâches causé par l'ennui.

La génération procédurale est une méthode très utilisée dans les jeux vidéo pour proposer du contenu varié et adapté aux joueurs. Elle s'appuie généralement sur un ensemble de données structurées et d'un ensemble de règles définies au travers d'algorithmes. Cependant, cette méthode est peu utilisée dans le domaine des EIAH. Dans cette thèse, nous nous intéressons à la génération d'activités de jeu pour l'entraînement aux connaissances déclaratives (i.e., informations factuelles telles que les lois, les tables de multiplication). L'adaptation est une thématique très large qui peut viser différents aspects, dimensions et objectifs. Vis-à-vis du contexte de cette thèse, trois angles d'adaptations, paraissant les plus pertinents, ont été

abordés : le point de vue de l'enseignant sur l'entraînement de chaque apprenant, la progression de l'apprenant-joueur vis-à-vis des connaissances à travailler et les préférences de joueurs en termes de *gameplays* (i.e., éléments « amusants » qui peuvent être contrôlés, décidés et réalisés par le joueur).

La thèse s'appuie sur le projet de recherche AdapTABLES qui lui fournit un premier terrain d'étude et d'expérimentation. Ce projet vise la conception et le développement d'un jeu d'entraînement aux tables de multiplication. La contribution de recherche (i.e., étude et moyens de conception) a été élaborée dans ce contexte mais a été généralisée et évaluée également dans d'autres contextes (i.e., repères d'histoire-géographie du brevet des collèges, connaissances théoriques de judo). La problématique identifiée concerne la facilitation de la conception de générateurs d'activités de jeu adaptées et variées, destinées à l'entraînement aux connaissances déclaratives. Plusieurs questions de recherche en découlent : *comment proposer une approche suffisamment générique pour considérer les connaissances déclaratives indépendamment d'un domaine didactique spécifique ? Qu'est-ce qu'une activité d'entraînement de jeu adaptée et variée ? De quels éléments éducatifs et de quels éléments de jeu sont composées ces activités ? Comment associer les éléments de jeu et les éléments éducatifs de manière cohérente ? Comment structurer ces éléments et leurs relations pour guider la génération d'activités cohérentes ? Comment spécifier ces informations informatiquement pour développer des générateurs d'activités ?*

L'objectif est de caractériser la génération

d'activités et de proposer une approche « générique », c'est-à-dire indépendante d'un domaine didactique spécifique, pour guider la conception de générateurs d'activités d'entraînement aux connaissances déclaratives. Ainsi, l'idée consiste à permettre la réutilisation des éléments de génération pour différents domaines didactiques. Sachant que la structure d'une activité change en fonction du genre de jeu visé, nous avons tout d'abord identifié un genre de jeu théoriquement compatible avec l'entraînement aux connaissances déclaratives : le *Roguelite*. Dans ce genre de jeu, les activités ou niveaux de jeu sont générés procéduralement et intègrent une grande variabilité. La répétition est favorisée par une mécanique de mort permanente nécessitant de recommencer le jeu du début mais certains éléments de jeu peuvent être conservés pour faciliter les parties suivantes.

Dans notre contexte nous définissons donc un générateur d'activité de jeu de type *Roguelite* pour l'entraînement aux connaissances déclaratives comme un composant logiciel (i.e., élément constitutif du jeu d'entraînement) dont l'algorithme permet de construire des activités variées à partir de trois types d'informations fournies en entrée : des informations sur l'entraînement et les connaissances visées, des informations sur le jeu et des informations sur l'apprenant-joueur concerné. Ces composants logiciels produisent en sortie des descriptions détaillées d'activités (i.e., niveaux de donjon) adaptées aux apprenants-joueurs.

Nous proposons un *framework* de conception et d'implémentation de générateurs d'activités adaptées et variées pour l'entraînement aux connaissances déclaratives à travers des jeux de type *Roguelite*. L'originalité de cette proposition et de ce positionnement est d'aborder l'adaptation en prenant en compte simultanément les dimensions de jeu et d'entraînement (i.e., apprentissage). De plus, le *framework* est un outil (i.e., infrastructure logicielle) disposant d'un mécanisme d'extension permettant de prendre en compte de nombreux domaines didactiques.

Notre proposition s'inscrit dans le contexte

d'une approche d'Ingénierie Dirigée par les Modèles (IDM) qui est fondée sur la notion de modèle (i.e., abstraction d'un système selon un point de vue) et repose sur quatre principes : la capitalisation (i.e., les modèles doivent être réutilisables), l'abstraction (i.e., les modèles doivent être indépendants des technologies), la modélisation (i.e., les modèles doivent adopter une vision productive, c'est-à-dire permettre la génération de code final du logiciel), et la séparation des préoccupations. La transformation de modèles est une opération centrale en IDM qui permet la génération automatique de modèles cibles à partir de modèles sources. Pour permettre ces transformations, les modèles doivent être conformes à des méta-modèles qui définissent la structure et les règles que doivent respecter les modèles. Les principes de l'IDM, ainsi que les différents outils permettant de soutenir son utilisation, font de l'IDM une approche très intéressante dans notre contexte.

Afin de construire le *framework*, nous avons dans un premier temps caractérisé et spécifié la génération en analysant les différents besoins d'adaptation (i.e., enseignant, apprenant-joueur). Pour cela, un ensemble de modèles conceptuels interreliés représentant les différents concepts nécessaires à la génération a été proposé (i.e., cadre conceptuel). Ces différents modèles conceptuels sont fondés sur une abstraction des concepts permettant la mise en place d'un mécanisme d'extension aux différents domaines didactiques (i.e., généralité) mais également l'explicitation des relations entre les éléments ludiques et éducatifs. Pour cette modélisation, les éléments d'entraînements et de jeu ont été abstraits de manière non exhaustive en types de tâche d'entraînement et en catégories de *gameplays* pour les *Roguelites*. À partir de ces concepts, la création d'une activité repose sur la bonne association des tâches et des *gameplays*. En conséquence, nous avons proposé une méthode systématique permettant de définir des relations entre type de tâche d'entraînement et catégorie de *gameplays* fondée sur l'utilisation d'un pivot défini à partir des exercices de questionnaires numériques.

Ce *framework* est composé d'un cadre conceptuel et d'une infrastructure logicielle extensible à des domaines didactiques spécifiques. Le cadre conceptuel s'appuie sur différents modèles. Tout d'abord le modèle du domaine capturant le parcours d'entraînement et les connaissances déclaratives, puis le modèle du jeu décrivant l'ensemble des éléments, *gameplays* et objets de jeu nécessaires pour l'élaboration d'une grande variété de donjons. Ensuite, le modèle de l'apprenant-joueur permet de conserver toutes les progressions et résultats de chaque apprenant et joueur. Le modèle de relations entre les éléments de jeu et d'entraînement capture les informations clés qui permettront d'assurer la correspondance des faits à questionner en *gameplays* et éléments concrets de jeu permettant de questionner dans le jeu les connaissances. Enfin, le modèle de l'activité décrit l'ensemble des données structurées composant le donjon généré. Ce modèle est le seul à être généré. L'infrastructure logicielle capture l'ensemble des éléments communs pour tout domaine didactique tels que décrits dans le cadre conceptuel. Elle propose également le mécanisme d'extension qui guide l'ajout des éléments spécifiques à un domaine didactique visé. L'avantage d'une approche extensible est de faciliter l'implémentation de générateurs d'activités, par des ingénieurs ou développeurs, en limitant le développement nécessaire aux informations reliés aux connaissances déclaratives du domaine didactique. Ce *framework* propose un algorithme, des modèles, et méta-modèles déjà existants à étendre et qui seront donc réutilisés pour spécifier un générateur d'activités. Il se base sur un ensemble de métamodèles et modèles informatisés à l'aide du *framework* de modélisation d'Eclipse (EMF) et propose un mécanisme d'extension fondé sur le patron de conception *template method*. Les générateurs

conçus à l'aide du *framework* sont des composants logiciels indépendants produisant des niveaux (i.e., donjons) au format XML, pouvant être interprétés par un jeu éducatif. Le *framework* et les générateurs produits par ce dernier doivent respecter certaines propriétés. Ainsi le *framework* doit permettre d'exprimer différents domaines didactiques et d'exprimer la vision des enseignants sur l'entraînement des apprenants individuellement. En ce qui concerne les générateurs, les activités générées doivent être 1) adaptées au niveau et résultats de l'apprenant dans son parcours d'entraînement, 2) adaptées aux préférences de jeu du joueur, 3) variées sur le plan éducatif et de jeu.

Afin d'évaluer la généricité et l'extensibilité du *framework*, trois générateurs ont été conçus à partir du *framework* : l'un pour l'entraînement aux multiplications, un second pour l'entraînement aux repères d'histoire-géographie au programme du brevet des collèges et un troisième pour l'entraînement aux faits de judo (i.e., connaissances théorique nécessaire au passage de grade). Le générateur des tables de multiplication est actuellement utilisé dans un jeu d'apprentissage conçu dans le cadre du projet AdapTABLES. Plus précisément, à chaque demande d'un nouveau niveau, le jeu envoie une requête au générateur, qui lui renvoie un niveau généré adapté à l'apprenant-joueur concerné. Le *framework* et ses composants ont également été validés à partir de tests systèmes à l'aide du *framework* JUnit 4, de validation de propriétés statiques sur les modèles, mais aussi au travers d'une expérimentation avec un ingénieur afin d'évaluer l'utilisabilité du *framework* et de son guide d'extension que nous avons également proposé. De plus, le jeu développé pour le projet AdapTABLES et le générateur associé ont été utilisés plusieurs fois en conditions écologiques.