Mapping Facts to Concrete Game Elements for Generation Purposes: A Conceptual Approach

Bérénice Lemoine, Pierre Laforcade

Mapping Facts to Concrete Game Elements for Generation Purposes: A Conceptual Approach

First Author[0000−1111−2222−3333], Second Author[1111−2222−3333−4444], and Third Author[2222−−3333−4444−5555]

NOWHERE
{no-one}@research.com

Abstract. Designing serious games or serious game activities requires mapping the educational elements and the game elements. This mapping is mainly addressed from a high-level game design perspective. Moreover, low-level mapping methods are generally domain-specific. Our aim is to address this problem, at an algorithmic level, in the context of activity generation (i.e., automatic creation of activities) for declarative knowledge training. This paper presents a generic modelling approach of the facts to be questioned and the gameplays, as well as an algorithm for the automatic and domain-independent generation of various gameplays for training purposes.

Keywords: Modelling · Generation · Gameplay · Serious Games.

1 Introduction

Recently, the design and use of learning games has become a recurrent theme [4]. Designing learning game activities requires to map the game elements with the educational elements [12]. This mapping requirement is more important in the context of game activity generation, since the generation algorithm needs to know the relations between elements to automatically build a coherent activity. The relations between elements might vary in regard to the game genre, the targeted knowledge, and the didactic domain. Some works propose methods for mapping game elements and educational elements during game general design [9, 11]. However, to the best of our knowledge, no work guides the implementation of these relations at an algorithmic level.

Declarative knowledge (DK), i.e., factual information, is part of the necessary knowledge to perform a task. Their memorisation, generalisation, and retention requires repetition [10]. Retrieval practice is a form of test-based learning, which has been shown to improve long-term retention of facts [14]. In our context, *training* on DK is considered a form of retrieval practice that involves repeatedly providing learner-players with various forms of questions about facts. Such training can generally be performed through formative quizzes or dedicated serious games. In order to reduce the feeling of boredom caused by repetition, serious games designed for declarative knowledge training must offer a wide variety of activities. However, existing training games are often lacking variety (e.g.,

no variation of gameplays, no aesthetic variation of the activities). In addition, these games are always specific to a single didactic domain. Activity generation is a solution for designing varied training activities that few works addresses in Technology-Enhanced Learning [3].

This article tackles the domain-independent mapping of questioned facts (i.e., questions on facts) to gameplays (i.e., fun things that can be controlled, decided and achieved by players [12], which are described by game elements) at the implementation level. Our proposal is a generic modelling of questioned facts and gameplays, as well as a domain-independent algorithm for gameplay (i.e., structured game elements) generation.

2 Research Context

2.1 Related Works

Several works have approached the mapping problem between game and educational elements by proposing relations between high-level concepts. Rapeepisarn et al. [13] proposed an extension of Prensky's work [12] by adding the relationships between learning styles to the existing relationships between game genres, knowledge to be learned and learning activities. Gosper and McNeill [8] defined relationships between learning outcomes, learning processes, assessment and game genres. Moreover, Dondi and Moretti [7] attempted to link knowledge types and learning objectives to high-level game features (e.g., content engine, evaluation engine) that the game should possess. Although very interesting for the general design of learning games, these relationships cannot be used at the implementation level to match learning content to concrete game elements.

Some works offer methods to guide the specification of relations (i.e., either to analyse games or to conceive them). Lim et al. [11] proposed the LM-GM framework that supports the transition between learning objectives and game mechanics (e.g., collaboration, orientation, exploration) through concepts called Serious Game Mechanics. The game mechanics considered are high-level concepts that can have many concrete implementations in a game. Hall et al. [9] proposed a method that guides designers in defining the transition between learning content to core-gameplays, by having them answer questions from both real-world and game-world perspectives. Although interesting, this work is also more aimed at general game design.

In conclusion, mappings between learning elements and game elements are not addressed from a low-level design standpoint, but only at high conceptual levels or at low-level in studies for very specific contexts [5]. Indeed, it is not easy to propose reusable techniques at a sufficiently generic level of abstraction.

2.2 {Anonymised} project

{Anonymised} is a research project that aims to build a Roguelite-oriented game for multiplication tables training. This project includes a user group composed

of mathematics experts who have also been involved in the game design (e.g., training specification, gameplays evaluation).

Roguelite has been shown to be an adequate genre for DK training [1]. A Roguelite training activity is a dungeon (cf. Figure 1), i.e., a set of interconnected rooms traversed by an avatar in which the training takes place. A dungeon has an entry room and an exit room. A room has accesses to others rooms (i.e., its neighbours), is associated to a training task (e.g., complete the fact, identify the correctness of a fact) and a gameplay, i.e., a set of positioned game elements that can be interactive (e.g., blocks to be pushed by the avatar, pots to be moved) or not (e.g., decoration blocks, texts). Based on the gameplay, and facts questioned, a room has a set of positioned elements. A positioned element has a type (i.e., game element that it implements, e.g., a chest, a pot, an enemy) and a position in the room. These elements can also have zero to multiple display values that displays facts propositions or statements of a specific questioned fact, or simple textual information. To enable the game to evaluate learners' actions, when a positioned element proposes choices or expects responses, it must declare whether it represents a correct response (i.e., *isAnswerElement*) or what values it expects (i.e., *expectedAnswers*).

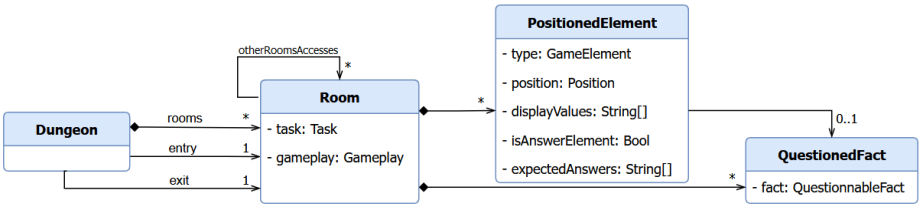

Fig. 1: Conceptual Modelling of Roguelite Activities for DK Training.

2.3 Research Questions

Our overall objective concerns the generation of Roguelite-oriented activities for training DK. In previous work, we identified training tasks with teachers and education specialists in mathematics and history-geography, such as *complete a fact where the multiplicand is missing*, *place historical dates in chronological order*, *identify the results of a table*, *name and locate countries of the European Union*, and so on. For a genericity purpose, an abstraction of these tasks led us to define four task types: Completion (i.e., completing a fact with missing elements), Order (i.e., ordering facts using a heuristic), Identification (i.e., attesting the validity or invalidity of facts) and Membership Identification (i.e., identifying elements with a common property). In addition, the definition of gameplays for Roguelite activities led us to define 5 gameplay categories inspired by the game classification of Djaouti et al. [6]: SELECT (i.e., selecting objects with the right

answer), MOVE (i.e., moving the correct objects to the expected areas), ORIENT (i.e., orienting the object to the right answers), POSITION (i.e., placing the avatar in the right positions) and DIRECT RESPONSE (i.e., typing in the right answers). In order to design such activities independently of any specific didactic domain (i.e., consider declarative knowledge in general), we identified 4 design questions:

1. How can training tasks be mapped onto gameplays? *(mapping)*
2. How can the facts questioned (i.e., build from tasks) be defined domain-independently in order to be used generically? *(modelling)*
3. How can gameplays be defined in terms of variable game elements? *(modelling)*
4. Once a task is paired with a compatible gameplay. How can these questioned facts be transformed into specific game elements? *(instantiation)*

To answer #1, we proposed a systematic approach for mapping task types (e.g., Completion, Order) to gameplay categories (e.g., SELECT, MOVE) [2]. This approach guides the specification of machine-readable relations that describe the conditions under which a gameplay category is compatible with a task type. Thus, to select a gameplay for a given task, the algorithm must sort the gameplays according to their categories (i.e., the categories that are compatible on the basis of the defined relations). It is important to note that some gameplays may be specially designed for a specific task type and thus have a parameter restricting their availability to that type only.

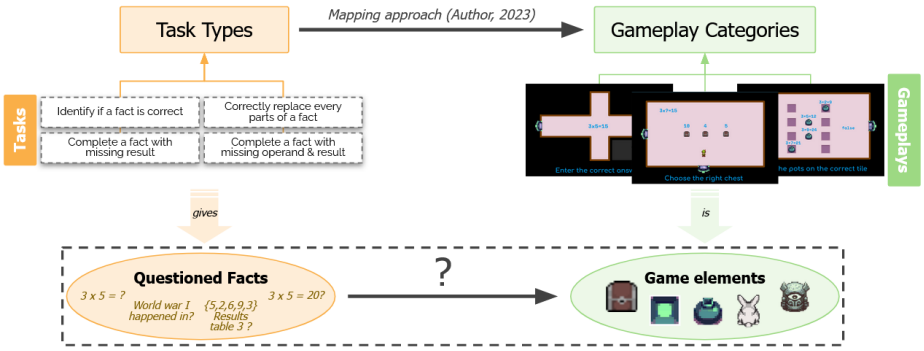

Fig. 2: Research Problem Illustration.

Facts are raw information such as $3 \times 5 = 15$. Depending on the task and its parameters, the questions about the facts to practice changes. For example, completing a fact with the missing operand will yield questions such as $3 \times ? = 15$, while identifying if elements share a given property will yield questions such as "Which ones are results of tables three? {3, 5, 9, 13}". These *questioned facts* have different structures. In addition, gameplay also have different structures and elements (e.g., some elements can wear one proposition, others can

wear multiple propositions, some game elements are simple elements, others are composites). Preliminarily, it would seem that the association of questioned facts with gameplays must be performed specifically for each task/gameplay pair. To that extent, our question (cf. Figure 2) concerns #2, #3, and #4: *How to model facts being questioned and gameplays to drive the generation of corresponding game elements?* Our proposal consists in modelling both concepts (i.e., the facts being questioned and the gameplays) at a sufficient level of abstraction. Such modelling would allow coverage of different forms of questioned facts, as well as variety in terms of game elements available for a given gameplay.

3 A Conceptual Design Approach

This section presents our proposal for modelling the questioned facts, the gameplays as well as a possible generation algorithm. In this article, the illustrative examples focus on multiplication tables.

3.1 Questioned Facts Model

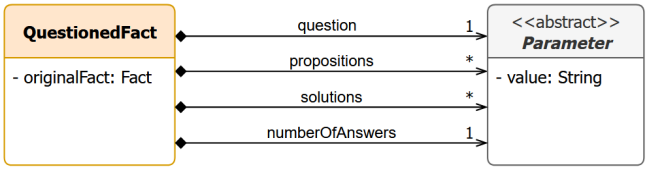

Fig. 3: Proposed modelling of questioned facts.

Questioned facts are questions about facts. To represent questioned facts generically, our main idea is to consider them as elements with possible parameters. Accordingly, each parameter is instantiated if necessary. Although the form of the questioned facts varies according to the training task concerned, the concepts that compose questioned facts are generic. Let's take two tasks T1 and T2 as an example. T1 consists in choosing from a set of propositions the answer corresponding to the multiplication result for each fact. From the parameters of T1, the facts questioned would be constructed to give a question such as $2 \times 6 = ?$ and a set of propositions such as {8, 12, 14} for example. T2 consists in choosing from a set of propositions those that are the possible results of a given table. From the parameters of T2, the constructed questioned facts would give questions such as "Which elements are results of table 3?" as well as a set of propositions such as {3, 5, 7, 9, 12}. It can be seen that, although the questions associated with the facts do not have the same form, the facts from both tasks are composed of a question (i.e., a text) and a set of propositions. Consequently, a questioned fact has a parameter *question*, describing the question to be asked,

and a parameter *propositions*, representing the list of possible choices for answering. Since questioned facts represent questions about facts, they need to know their solutions. Knowing the question and the propositions alone is not enough to assess a learner's answer. To this extent, the facts questioned have another parameter which indicates the correct solutions to the corresponding question. Furthermore, from an automation standpoint, it is necessary to know the number of expected answers (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. It is important to note that this parameter could be deduced from the number of solutions, but for the sake of clarity we decided to make it explicit.

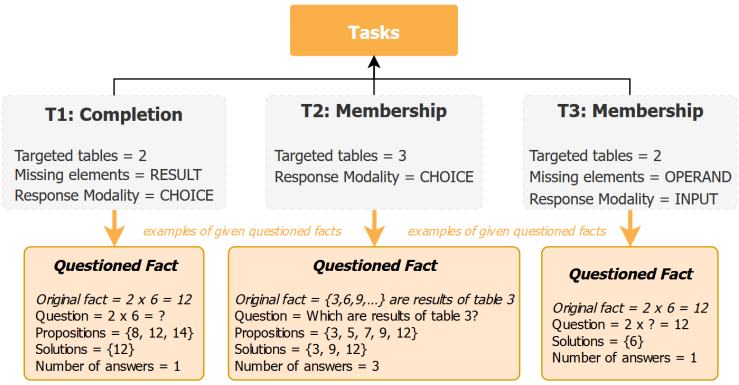

Fig. 4: Examples of questioned facts in generic form.

Furthermore, our previous example of tasks only considered tasks where the response modality was *choice* (i.e., selecting from a list of propositions). Let's consider a task T3, which consists, for each fact, of typing the answer corresponding to the operand of the multiplication. A questioned fact built from T3 would have a *question* such that $2 \times ? = 12$, its *propositions* parameter would not be instantiated, its *solutions* parameter would contain 6, and the *expected answers* parameter would be equal to 1. Figure 4 presents the questioned facts examples built in a generic form from the different tasks. It is obviously important to note that facts can be constructed in a generic form, but that their construction necessarily depends on the domain.

3.2 Gameplays Model

In a game, gameplay is represented by the set of elements with which the player interacts or which provide him/her with information. Statically defining gameplays in terms of specific game elements allows a certain level of diversity. However, it creates two principal constraints: 1) it is time-consuming, i.e., gameplays have to be described one by one according to the game elements available; and 2)

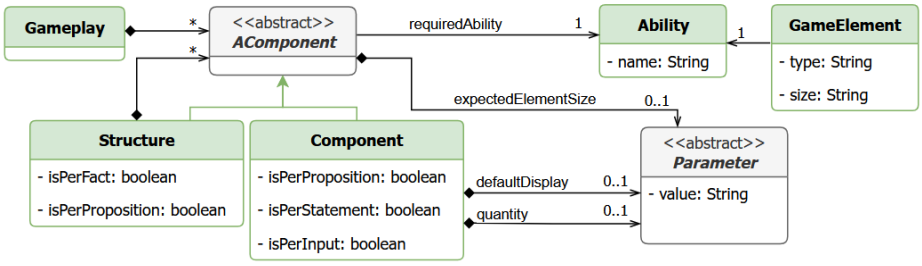

Fig. 5: Proposed modelling of gameplays & game elements.

it is static, i.e., adding a game element means having to specify new gameplays for that element. Our proposal is to use *abilities* to describe gameplays through variable game elements. This involves describing game elements in terms of abilities, such as: a block can be pushed (i.e., *pushable*), a pot can be moved (i.e., *movable*), a bridge can be crossed (i.e., *crossable*), etc. Abilities capture the elements' behaviour (i.e., how avatars can interact with them). This enables the definition of different elements with the same ability, for example: a cube and a pot can be moved (i.e., *movable*). Such modelling of game elements allows gameplays to be described in terms of components[1] that rely on a specific ability rather than a specific game element. Thus, it creates gameplay variability in the sense that the ability is known, but the actual game element will be chosen by the algorithm. There are two types of gameplays components: simple component, i.e., elements that are not composed of other components (e.g., chest, pots, enemies), and 2) structure components, i.e., elements that are composed of other components (e.g., components that describe blocks to be pushed on specific tiles: $structure = [block, tile]$)). As the context is the training of DKs, gameplay components have an intention. They can represent a fact, a statement, a proposition, and so on. Structures (i.e., composite components) can be instantiated for each fact questioned in a room or for each proposition of a fact (cf. Figure 6).

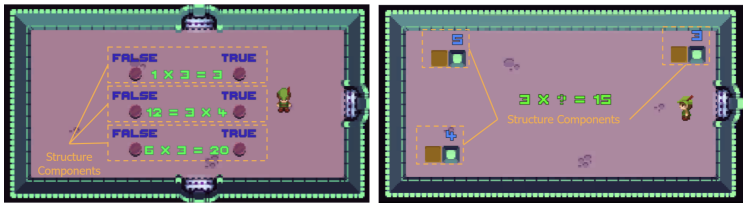

Fig. 6: Gameplays with structures per fact (left) / per propositions (right).

[1] 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 Fig. 1.

Simple components can be instantiated to represent statements, propositions, or input area. These component parameters are necessary for the algorithm to correctly instantiate the game elements corresponding to the gameplay, as they allow the algorithm to know which parts of questioned facts must be associated to which components. Some simple components can describe decoration elements or answer areas (e.g., tiles where the player must place elements). These elements can therefore have a default display or specify a default necessary quantity that can depend on the expected number of answers to the question. In addition, game elements are described according to their size. Therefore, a specific ability can be represented by different sized elements. As a result, a game component can specify an expected size so that the algorithm maintains consistency when instantiating a gameplay.

3.3 Generation Algorithm

As mentioned in the previous sections, our overall objective is to generate training activities (i.e., dungeons) for DK training. A dungeon is a set of interconnected rooms that are associated to a training task and a corresponding gameplay. The game elements composing the gameplay (i.e., called positioned elements) must be built based on the questioned facts to work on. Algorithm 1 describes the main structure of the generation algorithm. The general idea is to go through the gameplay components and, depending on the type (i.e., Structure or Simple Component), to call the corresponding method, to build positioned elements, with the required parameters.

Algorithm 1: Generate Room Positioned Elements (simplified)

```
1  Function createRoomElements(gameplay, facts, room):
2      for AComponent comp: gameplay.getComponents() do
3          room.element.addAll(buildComponentElements(comp, facts, []));
4  Function buildComponentElements(comp, facts, listElems):
5      gameE ← findGameElement(comp.getAbility(), comp.getSize());
6      if comp is Structure then
7          listElems.add(buildElement(comp, facts, gameE);
8          for AComponent sComp: comp.getComponents() do
9              return buildComponentElements(sComp, facts, listElems);
10     else
11         return listElems.add(buildElement(comp, facts, gameE);
```

For example, let's take the following questioned fact $QE1$ = {*question*="2 × ? = 12", *propositions*=[8, 4, 6], *solutions*=[6], *numberOfExpectedAnswers*=1} and a gameplay $G1$ = {*Structure1*, *Component1*}. *Structure1* = {*isPerFact*=false, *isPerProp*=true, *ability*=HORIZONTAL, *components*=[*StrComp1*, *StrComp2*]}

describes a horizontal structure that must be instantiated for each proposition of a questioned fact (i.e., the number of structure in the room equals the number of propositions of the questioned fact). *StrComp1* = {*isPerProp*=true, *isPerStatement*=false, *isPerInput*=false, *ability*=PUSHABLE} is a simple component that describes that each horizontal structure must comprise a pushable element that bears one of the facts propositions. *StrComp2* = {*isPerProp* =false, *isPerStatement*=false, *isPerInput*=false, *ability*=DETECTOR, *expectedSize*=small} is a simple component that describes that each horizontal structure must comprise a small element that can detect another element (i.e., the pushable element). *Component1* = {*isPerProp*=false, *isPerStatement*=true, *isPerInput*=false, *ability*=DISPLAY} is a simple component that describes that the room must contain an element to display the statement (i.e., question). This description can instantiate gameplays such as the one shown on the right in Figure 6.

From there, building the positioned elements of a room consists in: 1) finding a game element with the correct size and ability, 2) selecting a position in the room, 3) linking questioned facts parameters values to positioned elements parameters values based on component parameters, 4) adding the built positioned element to the room. Figure 7 present an example of positioned elements that can be generated from a set of game elements, a gameplay description, and a given questioned fact.

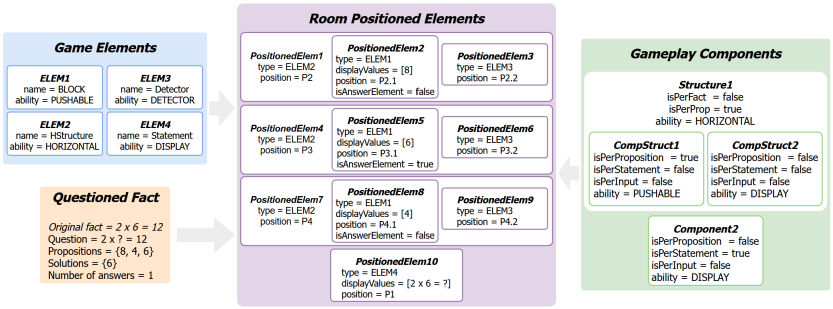

Fig. 7: Example of generated positioned elements.

3.4 Real-Case Study Application

The proposed approach has been applied for the implementation of an activity generator dedicated to multiplication table training. Our activity generator is implemented in Java and uses the Eclipse Modelling Framework (EMF) to model every piece of data required for generation. This generator produces detailed XML descriptions of dungeons that are currently being used in a game prototype developed in the context of {anonymised} project. The prototype acts as a *game player* which interprets XML files and translates them into playable dungeons. Figure 6 presents screenshots of the prototype.

4 Conclusion & Perspectives

To conclude, this article proposes an approach to model questioned facts and gameplays generically. This approach enables abstraction from any didactic domain and facilitates the variety of gameplays proposed for the purpose of game activity generation for DK training. The approach has been implemented in an activity generator for multiplication table training. This generator is currently being used in a prototype designed as part of the {anonymised} project. The main limitation of our work is that the approach has only been applied to one didactic domain. As a result, we intend to implement the approach in a second didactic domain about history-geography facts.

References

1. Author, N.: Title1. In: Anonyme (2023)
2. Author, N.: Title2. In: Anonyme (2023)
3. Bezza, A., Balla, A., Marir, F.: An approach for personalizing learning content in e-learning systems: A review. In: Second International Conference on E-Learning and E-Technologies in Education. pp. 218–223. IEEE, Lodz, Poland (2013)
4. Codish, D., Ravid, G.: Detecting playfulness in educational gamification through behavior patterns. IBM Journal of Research and Development **59**(6), 1–14 (2015)
5. Debabi, W., Champagnat, R.: Towards Architecture for Pedagogical and Game Scenarios Adaptation in Serious Games (2017)
6. Djaouti, D., Alvarez, J., Jessel, J.P., Methel, G., Molinier, P.: A Gameplay Definition through Videogame Classification. IJCGT, pp. 1–7 (2008)
7. Dondi, C., Moretti, M.: A methodological proposal for learning games selection and quality assessment. BJET **38**(3), 502–512 (2007)
8. Gosper, M., McNeill, M.: Implementing game-based learning: The MAPLET framework as a guide to learner-centred design and assessment. In: Assessment in Game-Based Learning, pp. 217–233. Springer Nature, United States (2012)
9. Hall, J.V., Wyeth, P.A., Johnson, D.: Instructional objectives to core-gameplay: A serious game design technique. In: Proceedings of the First ACM SIGCHI Annual Symposium on Computer-human Interaction in Play. pp. 121–130. ACM, Toronto Ontario Canada (2014)
10. Kim, J.W., Ritter, F.E., Koubek, R.J.: An integrated theory for improved skill acquisition and retention in the three stages of learning. Theoretical Issues in Ergonomics Science **14**(1), 22–37 (2013)
11. Lim, T., Carvalho, M.B., Bellotti, F., Arnab, S., de Freitas, S., Louchart, S., Suttie, N., Berta, R., Gloria, A.D.: The LM-GM framework for Serious Games Analysis. Pittsburgh: University of Pittsburgh (2013)
12. Prensky, M.: Computer Games and Learning: Digital Game-Based Learning. Handbook of Computer Game Studies (2005)
13. Rapeepisarn, K., Wong, K.W., Fung, C.C., Khine, M.S.: The Relationship between Game Genres, Learning Techniques and Learning Styles in Educational Computer Games. In: Technologies for E-Learning and Digital Entertainment, vol. 5093, pp. 497–508. Springer Berlin Heidelberg (2008)
14. Roediger, H.L., Pyc, M.A.: Inexpensive techniques to improve education: Applying cognitive psychology to enhance educational practice. Journal of Applied Research in Memory and Cognition **1**(4), 242–248 (2012)