How to assist designers model learning games with Petri nets?

Mathieu Muratet, Amel Yessad, Thibault Carron

How to assist designers to model learning games with Petri nets?

Mathieu Muratet
mathieu.muratet@lip6.fr
Sorbonne Université, CNRS, LIP6, INSHEA
Paris / Suresnes, France

Amel Yessad
amel.yessad@lip6.fr
Sorbonne Université, CNRS, LIP6
Paris, France

Thibault Carron
thibault.carron@lip6.fr
Sorbonne Université, CNRS, LIP6, Université Savoie Mont Blanc
Paris / Chambéry, France

ABSTRACT

In previous research, we presented a methodological framework that provides players with adaptive feedback. The core of this framework relies on modeling the learning game with a Petri net. However, this modeling is a challenging task. Indeed, Petri nets are well adapted to model dynamic and complex systems but require a mastery of the underlying mathematical formalism to build them manually. In particular, when the learning game is characterized by a large freedom of action. In this paper, we present an authoring tool and its domain-specific language to assist designers to model learning games with Petri nets. We carried out a case study where our contribution was implemented. Results show that our contribution helps designers to build the Petri net in combination with classical Petri net editors which are still useful to visualize, to check and to validate the Petri nets built.

KEYWORDS

Learning Game, Behavioral Model, Petri Net, Design Pattern, DSL

ACM Reference Format:
Mathieu Muratet, Amel Yessad, and Thibault Carron. 2022. How to assist designers to model learning games with Petri nets?. In *FDG '22: Proceedings of the 17th International Conference on the Foundations of Digital Games (FDG '22), September 5–8, 2022, Athens, Greece.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3555858.3555937

1 INTRODUCTION

Adaptive feedback in Technology-Enhanced Learning (TEL) is important to help learners acquiring competencies, progressing in problem solving and improving the learning process [4, 13, 18]. In learning games, the adaptive feedback remains a challenging issue because the analysis of the learner traces is complex. Indeed, in learning games with a large amount of available actions, the learners are free to interact with the game objects and to explore the game environment. This makes it hard to understand and analyze the learner's traces and to answer questions like: Does the learner progress to the solution? What is the player's next action to perform in order to progress towards the solution? Which feedback is relevant to help the learner solve the problem, acquire a competency or break through erroneous cognitive reasoning?

To provide automatically hints adapted to the learner in learning games and help the player to progress in problem solving, we need to model the resolution process of the learning game. In previous research [9, 16, 19], we proposed a methodological framework to tackle this challenge by means of Petri nets (see Fig. 1). The framework is divided in two workflows: the first deals with the use of Petri nets to generate feedback adapted to the player difficulties (the feedback loop), and the second is the workflow to model the game with Petri nets.

We proposed a solution in [10] for the first workflow (the feedback loop) and detailed the algorithm that exploits Petri nets properties to analyze the learners' actions, to characterize them with pedagogical labels, and to calculate a label-based score. These pedagogical labels provide users, and particularly teachers, with information about the learner's behavior such as, correct actions, erroneous actions, actions taking place too late or too early in the learner's solving process.

In this paper we focus on the second workflow that aims to assist designers to build the Petri net used by the feedback loop (see FilteredPn in Fig. 1). When designers develop the learning game and describe the different monitored interactions, as detailed in this paper, they build the learning game and export automatically a first Petri net called a full Petri net (see FullPn in Fig. 1). Afterward, an expert plays the game (several times if there is more than one solution available) and the traces are used to filter the full Petri net in order to build the filtered Petri net automatically. The full Petri net includes all monitored actions the learner can perform and the filtered Petri net is a sub-part of the full Petri net including only actions used by an expert to solve the game. Thus, it becomes possible to compare between the experts' solution and the player solution in order to provide relevant feedback to the player.

A critical part of this second workflow is the building of the FullPn. Indeed, modeling a complex system, like a game, with a Petri net is not a trivial task [11].

Thus our main issue is: How to assist designers to model complex environments like learning games with Petri nets? We make the hypothesis that providing small patterns of Petri nets and hiding mathematical formalism of Petri nets with domain-specific language will help to model links between game objects. These links are processed to automatically build the full Petri net that models the entire learning game and thus avoids designers having to build it manually.

In the two next sections, we introduce the Petri net formalism and we give an overview of the existing research and related work. Section 4 deals with the contribution of this paper: the domain-specific language that helps designers to model links between game objects. Section 5 presents a case study in which our contribution was implemented and discusses the results. Finally, we conclude the paper and present some future works identified as useful to pursue this research.

FDG '22, September 5–8, 2022, Athens, Greece

ACM ISBN 978-1-4503-9795-7/22/09…$15.00
https://doi.org/10.1145/3555858.3555937

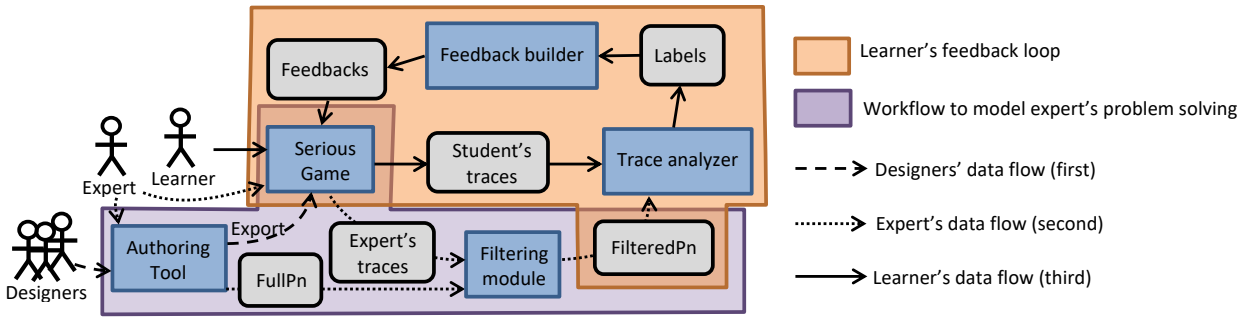

Figure 1: Global architecture of the framework

2 PETRI NET FORMALISM

Petri net is a graphical and mathematical modeling tool useful to simulate the dynamics of complex systems. As a graphical model, it is readable by human modelers and can be used as a visual communication aid similar to flow charts. A Petri net (PN) is a bipartite graph with two disjoint sets of vertices P and T. It is a 5-tuple [8] $PN = (P, T, F, W, M_0)$ where:

- $P = p_1, p_2, \ldots, p_m$ is a finite set of places,
- $T = t_1, t_2, \ldots, t_n$ is a finite set of transitions,
- $F \subset (P \times T)U(T \times P)$ is a set of arcs (flow relation),
- $W : F \mapsto \mathbb{N}^*$ is a weight function,
- $M_0 : P \mapsto \mathbb{N}$ is the initial marking or state,

$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

The behavior of many systems can be described in terms of system states and their changes. To simulate the dynamic behavior of a system, a state (or a marking) is changed according to the following transition (firing) rule:

(1) A transition t is said to be enabled if each input place p of t is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of the arc from p to t.
(2) An enabled transition may fire (depending on whether or not the corresponding event takes place in the simulated system).
(3) The firing of an enabled transition t removes $w(p, t)$ tokens from each input place p of t, and adds $w(t, p)$ tokens to each output place p of t, where $w(t, p)$ is the weight of the arc from t to p.

The set of all markings (states) that are reachable from the initial marking of a Petri net by firing the transitions represents the range of states of the system and is called the reachability graph. This graph is the core model we use to analyze the players' actions. In case the number of game states is infinite, particularly for unbounded Petri nets (some places become unbounded because of the increasing number of their tokens), the reachability graph cannot be calculated and is replaced by another graph called the coverability graph. More details about the Petri net formalism can be found in [8, 12].

In the context of this work, the places represent the states of the game objects and the transitions the actions that the player can perform in the game objects. Then, performing game actions can be simulated with a Petri net by triggering (firing) the transition associated with the game action according to the previous rule. This trigger will fire transition in the Petri net and change the current Petri net marking to keep it synchronized with the game state.

In the Petri nets we use to model games, we include "inhibitor" arcs that have a different pattern of operation compared to the standard arcs. In the case of an "inhibitor" arc, the transition t is enabled if each input place p of t is marked with less than $w(p, t)$ tokens. This kind of arc is interesting to model games, especially to model actions that are not possible if the game state exceeds a threshold. For instance, considering a simple game where players can be attacked (they lose 1 health point for any damage) and have the power to heal themselves (they get 10 health points back). Players can heal themselves only if their health points are less than or equal to 90 (to avoid exceeding the maximum of 100 health points). In this example, an inhibitor arc avoids creating an artificial place to store the number of health points lost (see Fig. 2). Saving this artificial place is crucial in our case because it avoids mixing the healing action and another game action, such as the attack in this example. In this context, using inhibitor arcs enables designers to simplify their work. Indeed, in this example, the designers will be able to express a constraint on the healing action without considering side effects to other game actions (see section 4 for details on how to express these constraints).

In sum, we use the Petri net formalism to model learning games and we use the coverability graph to check properties about the player's game solving. Thus, for example, we can answer questions like: is the game end reachable from the current state of the game (there is a sequence of enabled transitions from the current state to the final state)? Is the player stuck and can no longer progress in the game (no transition is reachable from the current game state)? See [10] for more details.

3 POSITIONING

As we introduced in section 2, Petri net is a powerful tool to simulate the dynamics of complex systems. In his thesis, Dormans [6] reviews several approaches to assist game designers and shows that Petri net is one of the most promising simulation tool but is less accessible to designers. This has been a known difficulty for

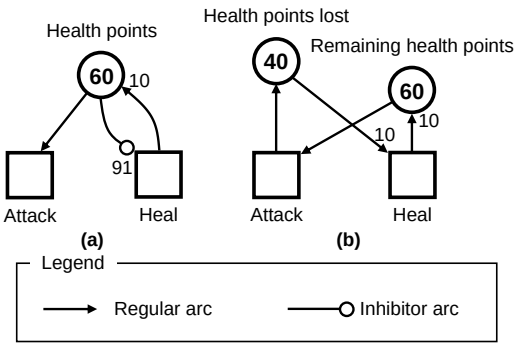

Figure 2: An attack/heal system modeled with inhibitor arc (a) and without inhibitor arc (b)

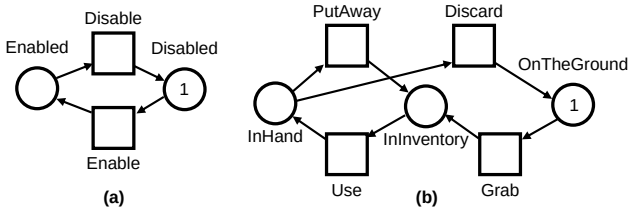

Figure 3: Activable behavior pattern (a) and pickable behavior pattern (b) modeled with Petri nets formalism

quite some time, Naedele and Janneck [11] identify in 1998 that "*it is easy to grasp the basic concepts of places, transitions, and concurrent sequences, and that one is very quickly able to apply this to the modeling of systems that have an internal structure that only consists of applications of those basic concepts. The situation changes when one tries to model complex technical systems*". Their contribution to help in this field was to identify and describe several design patterns such as removing all tokens stored in a particular place or synchronizing two concurrent sub-nets. These design patterns are very interesting but no hints are provided to help designers to connect patterns together.

Gomes and Barros [7] have the same comment "*Petri nets are often difficult to use in practice due to the problem of rapid model growth*". Authors present a condensed view of the main structuring mechanisms for general system modeling with Petri nets. They introduce composition and refinement/abstraction strategies. Composition strategy includes fusion of Petri nets (called horizontal composition, as the nets are glued together at specific points, places or transitions) and folding that consists in identifying structural symmetries inside Petri nets. Refinement/abstraction strategy includes static macros (a place or a transition model a sub-net) and dynamic invocations (when a transition fires, a new net is dynamically created). One of their conclusions is that work on simple and intuitive notations for set modification would be interesting for further research.

Abstraction strategy was also studied with hierarchical Petri nets. Balas *et al.* [2] explain how they use hierarchical Petri nets to model plots and control their evolution in games emphasizing a story and featuring large worlds inhabited by virtual characters. Araújo and Roque [1] discuss the applicability of Petri nets to model game systems and game flows compared with other languages such as UML and show that it is possible to model game systems with hierarchical Petri nets. A lot of tools enable designers to edit and analyze Petri nets [17], but all of them require mastering Petri net mathematical formalism.

Dormans [5], inspired by Petri nets, proposes the Machination framework that allows designers to model and simulate games in an early stage of development. The author indicates that this framework seems to be more relevant for strategy games, simulation games, and board games, mainly because the internal economy of these types of games plays a more important role than in certain other types of games. Machinations framework uses foreground feedback structures that play an important role in the design process. Like Petri net editors, modeling with Machinations requires mastery of formalism and graphical operators. Moreover, Machinations was designed for an early stage of development while we need a model that can process in live game sessions.

All these research inspired us. We used the design pattern approach of Naedele and Janneck [11] to provide small and reusable Petri nets to designers rather than asking them to model the game from scratch. We also use hierarchical Petri nets to model the game at various levels, especially the macro-transition approach [7] that abstracts sub-nets by transition inside the super-net. Finally, we propose a domain-specific language to help designers to update Petri nets. This domain-specific language implements the AND-split, AND-join, and OR-split introduced by Sun *et al.* [14].

4 CONTRIBUTION

In section 1 we introduced the global architecture of the framework and we noticed that a key part of this framework was the building of a Petri net that models the learning game. In this section, we detail the method and tools we designed to tackle this challenge.

Our approach consists of asking designers to build manually small Petri nets. Each one of these small Petri nets models a behavior pattern that can be linked to a game object. We assume that this effort is much less than if we ask them to build a full Petri nets manually (which can be composed of several hundred places and transitions). We designed a set of generic Petri nets: activable, pickable, activationCount, exclusiveChoice, unique action, etc. The objective is to capitalize on all these small Petri nets to avoid to create them from scratch for each new game level. Fig. 3 depicts the two first examples previously cited. A game object can be linked with several behavior patterns and a behavior pattern can be linked with several game objects.

4.1 A domain-specific language to describe links between game objects

Each game object the designer decides to include in the Petri net is called a "monitored object". If a game object is monitored, all learner interactions with this object are considered in the feedback loop (see Fig. 1). We propose connecting the small Petri nets by

formally explaining the links between the monitored entities with a domain-specific language.

To illustrate the process of using the full Petri net building we focus on an example (see Fig. 5), which presents a game level where the player has to open a locked frozen door. A key and a boiler are available in the scene to use in opening the door. The player has to pick up the key and turn on the boiler to melt the ice on the lock then use the key to unlock the door. The player can grab the key and turn on the boiler in any order, but opening the door has to be done last. In this example, we have three monitored game objects: the key, the boiler and the door.

Fig. 4 models relations to express links between **MonitoredObjects**. An object, like the door, is considered "monitored" if it contains at least one **MonitoredComponent**. Each **MonitoredComponent** contains a small Petri net (the "Activable behavior pattern" to the door, see Fig. 3) and a list of **MonitoredActions**. Each **MonitoredAction** has a reference to the monitored **Transition** (the **gameAction**) in the small Petri net (here the action "open" is associated to the transition "Enable" in the Activable behavior pattern) and a list of links. Links define constraints with other monitored objects (the key and the boiler). A **Link** is composed of four main attributes: a **Verb** to express the type of the constraint (REQUIRE AT LEAST, REQUIRE LESS THAN, GET or PRODUCE), a **weight** to value the constraint, a **gameState** that references a place of the Petri net in a monitored component and a **label** to rename the link. A monitored action can include several links with other monitored actions. The **logicRel** inside **MonitoredAction** enables to combine links with an AND/OR expressions. A **logicRel** has to follow a simple formal grammar where L is the set of available labels:

```
LogicRel ::= Label | ( LogicRel Op LogicRel )
Op           ::= + | *
Label        ::= L
```

The **logicRel** is distributed of $*$ (AND) over + (OR) to process expressions like $(l_x * l_y) + (l_x * L_z)$ rather than $l_x * (l_y + L_z)$. Then logic relation enables to link automatically small Petri nets together with basic Petri nets patterns (AND-split, AND-join, and OR-split [14]). An operator $*$ connects all links to the **gameAction** of the **MonitoredAction**, whereas an operator + duplicates the **gameAction** of the **MonitoredAction** and applies links to each copy.

Fig. 6 depicts several scenarios depending on AND/OR occurrences inside **logicRel**. The first scenario (S1) is the basic Petri net that models a simple counter that adds one token inside Counter place each time the GameAction is fired. The second (S2) is the adapted Petri net automatically built with a AND constraint: "require at least 1 token in State1 AND 15 tokens in State3" ($link1 * link2$), in the resulting Petri net two arcs was added to link the GameAction with its input states. The third (S3) is the same as the second, but with an OR constraint: "require at least 1 token in State1 OR 15 tokens in State3" ($link1 + link2$), in this case the GameAction was copied and each copy was linked with its appropriate input state. The fourth scenario (S4) illustrates a more complex constraint that combines AND and OR operators: "Require at least 1 token in State1 OR (15 tokens in State3 AND 2 tokens in State4)" ($link1 + (link2 * link3)$), in this case the output Petri net contains a copied GameAction (+ operator), on the first copy one arc was linked with State1 ($link1$) and on the second copy two arcs was linked with State3 and State4 ($link2 * link3$). The last scenario shows simply all kind of verbs to see their instance in Petri net formalism.

In our example of the locked frozen door (see Fig. 5), first, we link the key to the pickable behavior pattern and the door and the boiler with the actionable behavior pattern. Then, we define the opening of the door with the following constraint: "the boiler must be enabled AND the key must be in the avatar's hand". The output Petri net of this game level automatically built based on the behavior patterns selected and the links defined is shown in Fig. 7 (a).

Expressing a link obeys the following outlines:

(1) Selecting the action to apply a constraint on. This action refers to a transition included in one of the behavior patterns attached to a first game object.
(2) Selecting the state linked to this constraint. This state refers to a place included in one of the behavior patterns attached to a second game object.
(3) Selecting the type of link and its weight. We defined three kinds of links:
    - The "Get" link refers, in the Petri net formalism, to an arc that from the input place consumes tokens equal to the weight of the arc.
    - The "Require" link offers two possibilities:
        - The "at least" constraint refers to a "read" arc (in Petri net formalism) that does not consume tokens, but enables the output transition if the number of tokens inside the input place is greater than or equal to the weight of the arc.
        - The "less than" constraint refers to an "inhibitor" arc (in Petri net formalism) that does not consume tokens but disables the output transition if the number of tokens inside the input place is greater than or equal to the weight of the arc.
    - The "Produce" link refers, in the Petri net formalism, to an arc that produces tokens equal to its weight inside its output place.

To summarize, a link enables to create arcs between a **gameAction** and a **gameState**, the **Verb** defines the type of arc, and the **weight** specify the amount of tokens in play. Several links define several constraints on a same **gameAction**. The **logicRel** structures the links and how to connect the **gameAction** with all linked **gameStates**.

4.2 A plugin for Unity to monitor game entities and build the Petri net model

Fig 8 presents a screenshot of the monitoring module[1] that we integrate into the Unity environment. In this screenshot we implemented previous example and we changed the constraint of the previous example (the frozen door): "opening the door requires the boiler enabled OR the key in the avatar's hand". Formally, first we select the game object to apply the constraint on, the *Door* (1). Then we select one of its action, the *turnOn* action, renamed "Open" (2) and we add two constraints, the first is linked with the *Key* and

[1]The monitoring module used in this research is part of the FYFY project: https://github.com/Mocahteam/FYFY, accessed March 11, 2022

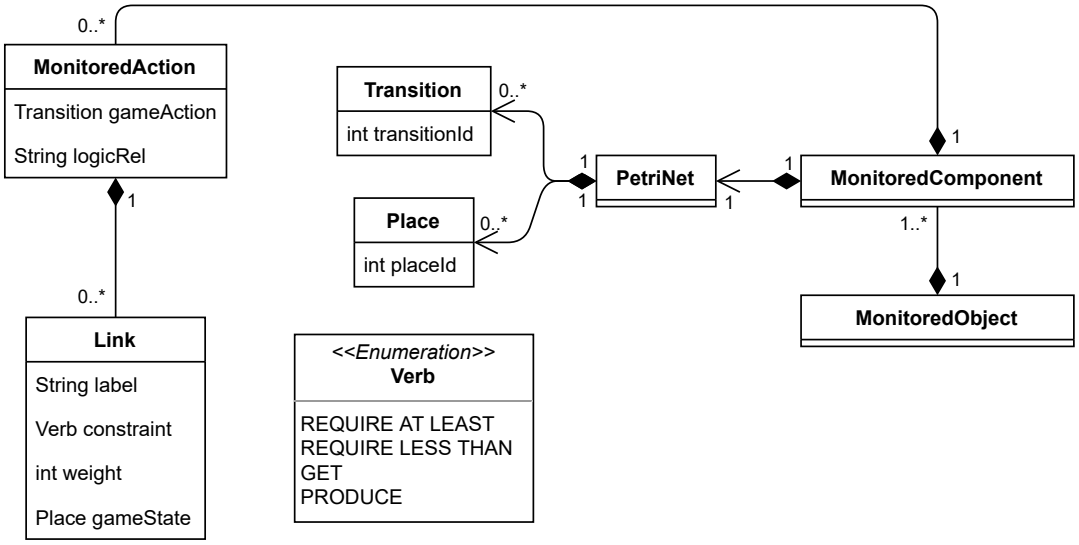

Figure 4: Monitoring model

Figure 5: Screenshot of "The frozen door" game level

"requires at least 1 (Key) inHand" (3) (this link is noted "$l0$"), the second is linked with the *Boiler* and "requires at least 1 (Boiler) on" (4) (this link is noted "$l1$"). At least one of these two links is required to open the door, so the logic expression is: "$l0 + l1$" (5). We can translate this expression "$l0 + l1$" like: "Open the frozen door Requires at least 1 Key inHand OR Requires at least 1 Boiler on". Then the resulting full Petri net with an OR operator is quite different in comparison with the AND version (see Fig. 7).

Thus, the monitoring module integrated into Unity provides the following functionalities:

- Selecting game objects to bind with the monitoring system.
- Associating small Petri nets to each monitored game object.
- Defining links between monitored elements.
- Building the Petri net with all links previously defined.

5 CASE STUDY

We have experimented with our proposition during the development of an open source virtual escape game called E-LearningScape[2] (see fig. 9). E-LearningScape is a numerical adaptation of a physical escape game[3]. In this numerical adaptation, the participants (in teams of 2 to 5 players) play the role of a sandman immersed in the dream of Camille. Their challenges will be to help Camille structure her thought in her dream by solving enigmas. The team members gather around a computer. One player controls the game and moves inside the virtual universe in a first-person navigation to discover interactive game objects and fragments of dreams giving access to clues in the real world. All members of the team solve enigmas inside and outside the game, these two facets feeding each other. E-LearningScape has two objectives, the main one is to introduce learners to concepts in one of the three knowledge domains considered in the game (pedagogy, accessibility or computer science), and the second objective is to promote team work and cohesion.

E-LearningScape is made up of 17 enigmas that require the players to combine more than 40 game objects found into the game. Without being exhaustive, here are some examples of enigmas: finding a wire to connect words on a pin panel; deciphering a message with a cylinder mirror; revealing hidden clues with a black light; etc.

As in classic escape games, several enigmas can be performed in parallel and the players have to associate found clues to resolve them. The monitoring module we work on aims to generate in-game hints to help players to solve enigmas. As we explain in previous sections, the monitoring module require Petri nets and we focus this case study on the enigmas modeling.

[2]E-LearningScape: https://github.com/Mocahteam/E-LearningScape, accessed February 03, 2022
[3]https://sapiens-uspc.com/projets-innovants/learningscape-2/, accessed February 03, 2022

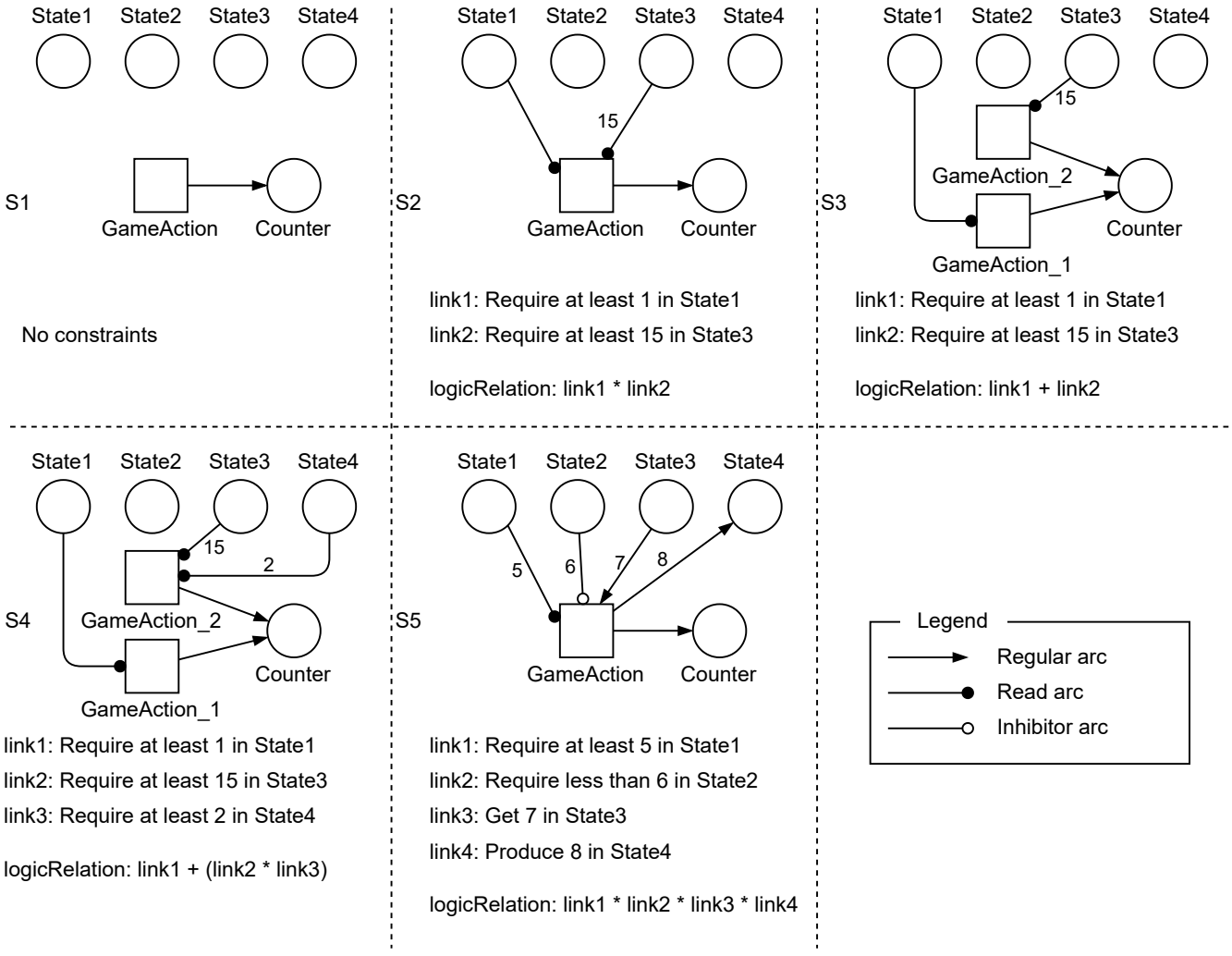

Figure 6: Examples of Petri net adaptations depending on logic relations

5.1 Enigmas modeling

The developer, who has coded the game, masters Unity and knows the Petri nets principle (he is not an expert of Petri nets). He had to model the 17 enigmas with the help of our Unity plugin and Tina editor [3].

The current model of the game is composed of 23 full Petri nets with a sum of 297 places, 231 transitions, and 558 arcs. All these full Petri nets were built from eight small Petri nets. From these eight small Petri nets, the developer reused four generic Petri nets we introduced in section 4 ("activable", "activationCount", "exclusiveChoice", and "unique action") and designed with Tina four new small Petri nets. The first, "activableAndCount" is a merge between "activable" and "activationCount" patterns, it allows to count how many time a game object has been activated. The second, "answerQuestion" models question validation options (right and wrong answer). The last two are a simple place and a simple transition used to complete specific behavior. Fig. 10 shows these eight small Petri nets used to model E-LearningScape.

We do not show full Petri nets of all enigmas in this paper due to place limitation, but the reader can download and open each enigma's PNML files available at [15] with Tina tool. We show just one example in Fig. 11 that illustrates what a full Petri net looks like, this full Petri net models the pin panel enigma that we mentioned in section 5.

Yet we present some interesting parts of Petri nets that were built. The game was not modeled with a unique Petri net for two main reasons: the first is a technical reason, computing the coverability graph of this Petri net would take too much time; the second is a usability reason, checking the validity of such model seemed very complex to the developer. Then, the developer chose to model enigmas' sequence with a super-net with only "unique action" patterns. For example, a gears enigma (the players have to select the right gear to unlock a mechanism) is available after resolving three

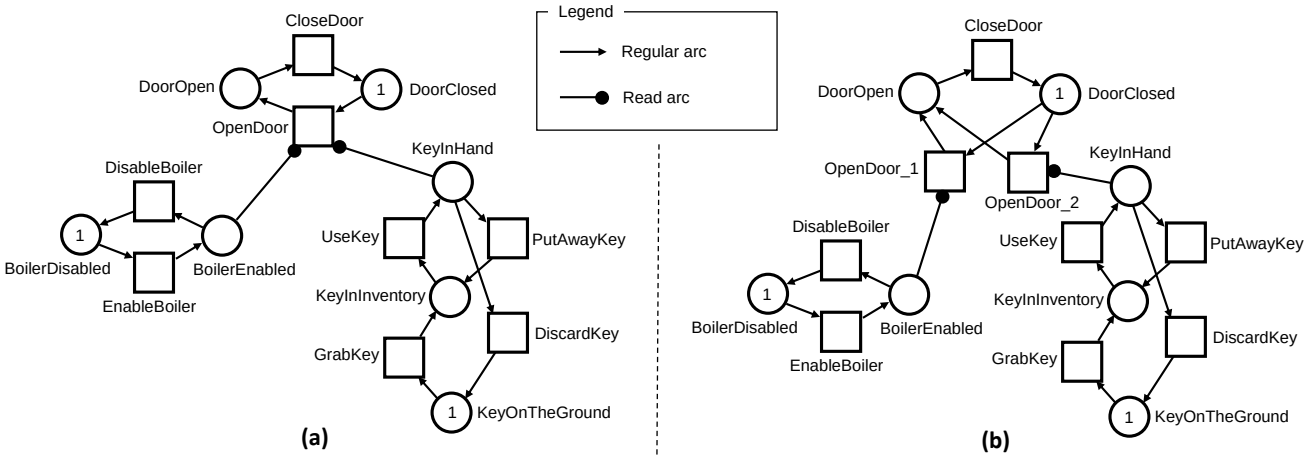

Figure 7: Petri net of "The frozen door" game level built automatically with a AND constraint "The door can be opened if the boiler is enabled AND the key is in hand" (a) and with an OR constraint "The door can be opened if the boiler is enabled OR the key is in hand" (b)

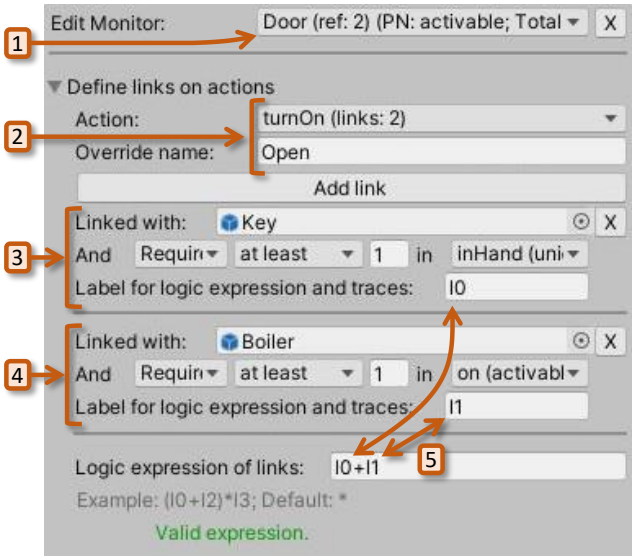

Figure 8: Screenshot of the user interface of the monitoring module integrated into Unity

previous enigmas. To model this constraint, the developer attached the "unique action" pattern to the four enigmas and add links to the "perform" action of the gears enigma (see Fig. 12).

Each transition of this super-net models an entire enigma. The details to resolve each enigma are defined in dedicated full Petri nets. The full Petri net presented in Fig. 11 is one example. We discuss the conception of these full Petri nets in the next section.

5.2 Discussion

First of all, we notice 17 enigmas and 23 Petri net (6 Petri nets in addition). Five enigmas were modeled with two Petri nets. The last

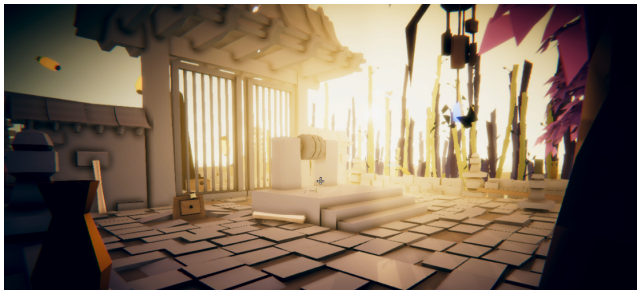

Figure 9: Screenshot of "E-Learning Scape", a virtual escape game on teaching skills

additional Petri net is the super-net that structures which enigma is available depending on the others.

Among the enigmas that require two Petri nets, we found the last four enigmas of the game. For these enigmas the answers can be entered freely in four input fields, but when an answer was excepted in one input field it will no longer be accepted in the others. The case of these last four enigmas is interesting because the monitoring module has to know which enigma is validated. Then the developer chose to model the heart of each enigma in its own Petri net and the validation mechanism in different Petri nets. We focus on this validation mechanism. The developer used the "exclusiveChoice" and "answerQuestion" patterns. We present in Fig. 13 a simplified example including only two choices instead of four. The two "exclusiveChoice" patterns model the two possible input fields and the "answerQuestion" pattern the answer of one enigma. "choice1" of each "exclusiveChoice" models that the associated input field was chosen to validate the answer of this enigma and "choice2" models that the associated input field was chosen to answer to another enigma. Selecting "choice1" in one enigma has to inhibit others "choice1". To integrate this constraint, the developer

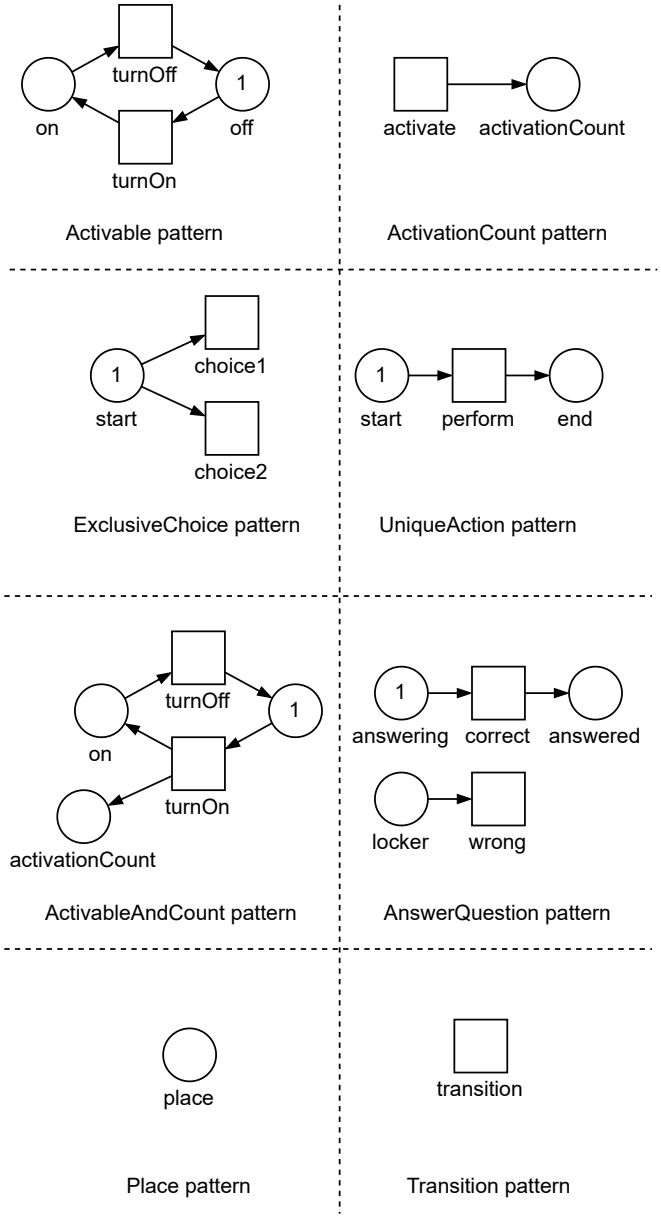

Figure 10: The eight small Petri nets used to model the 17 enigmas of E-LearningScape

added the following links on each "choice1": "Require less than 1 answer in answering AND produce 1 answer in answering". Then, "choice1" will be enabled if the "answering" place does not contain tokens. We notice that this modeling is not perfect because once the "correct" action will be fired, the "answering" place will be emptied and unlock the second "choice1". Developer would have to add another constraint on "answered" place like: "Require less than 1 answer in answered". In this case this mistake is not a problem because when the enigma is validated it is saved inside the super-net, then full Petri nets of this enigma will not be used anymore.

Another interesting modeling choice is the collecting behavior. Developer did not choose to use the "pickable" pattern we introduced in Fig. 3. He preferred using "uniqueAction" and "activable" patterns because, in E-LearningScape, a pickable game object can't be discarded and principally because it was implemented with two game objects, the "world object" and the "inventory object". Once the player clicks on the world object, it is disabled and inventory object is enabled. Then player believes he grabbed the object. Then developer attached the "unique action" pattern to the world object because it can be clicked once and the "actionable" pattern to inventory object because it can be selected and unselected. To model the constraint that the inventory game object can be used only when the word object was clicked he added a simple link to the two "actionable" actions (turnOn and turnOff). For instance, for a wire available in the game: "turnOn InventoryWire requires at least 1 WordWire picked". Fig. 14 shows how this constraint is defined in Unity and dotted area in Fig. 11 shows output result in full Petri net.

Finally, developer used the Unity plugin to associate small Petri nets with game objects and to define links between game objects to reflect game simulation. He exported regularly the Petri net he worked on with the Unity plugin and checked it with the simulator functionality of Tina. When he detected some inconsistencies with game simulation, he did not edit the Petri net with Tina and preferred to make modifications with the Unity plugin. He explained this choice because he was not sure how to update the Petri nets in particular for cases that contain complex relations. In this case, he preferred using the AND/OR expressions of the plugin.

We also identified situations not easy to model with current tool: (1) managing game objects that could appear dynamically in the game (not specified in the initial Petri net) and (2) simplifying the constraints defined on a minimum and maximum game object state, for instance to constrain a game action if the state of a game object is included between two values. These perspectives will enable us to improve current framework.

6 CONCLUSION AND FUTURE RESEARCH

In this paper, we presented a contribution to help learning game designers to model learning games. We based our work on Petri nets and we defined a domain-specific language to describe the links between small and reusable Petri net patterns that can be easily built. Based on this logical description we built automatically a full Petri net that models the player interactions in the game.

The tool we designed makes our conceptual approach operational. We experimented with our contribution by reviewing usage in a professional context. This case study shown a loop between the Unity plugin and Tina. The game developer designed 23 full Petri nets with a sum of 297 places, 231 transitions, and 558 arcs with only 8 small Petri net patterns. The plugin helps developer to build the Petri net thanks to the domain-specific language and Tina enables them to visualize, to check and to validate the resulting Petri nets.

The work done with E-LearningScape is a first step to implement a feedback loop inside the game. Currently the 17 enigmas

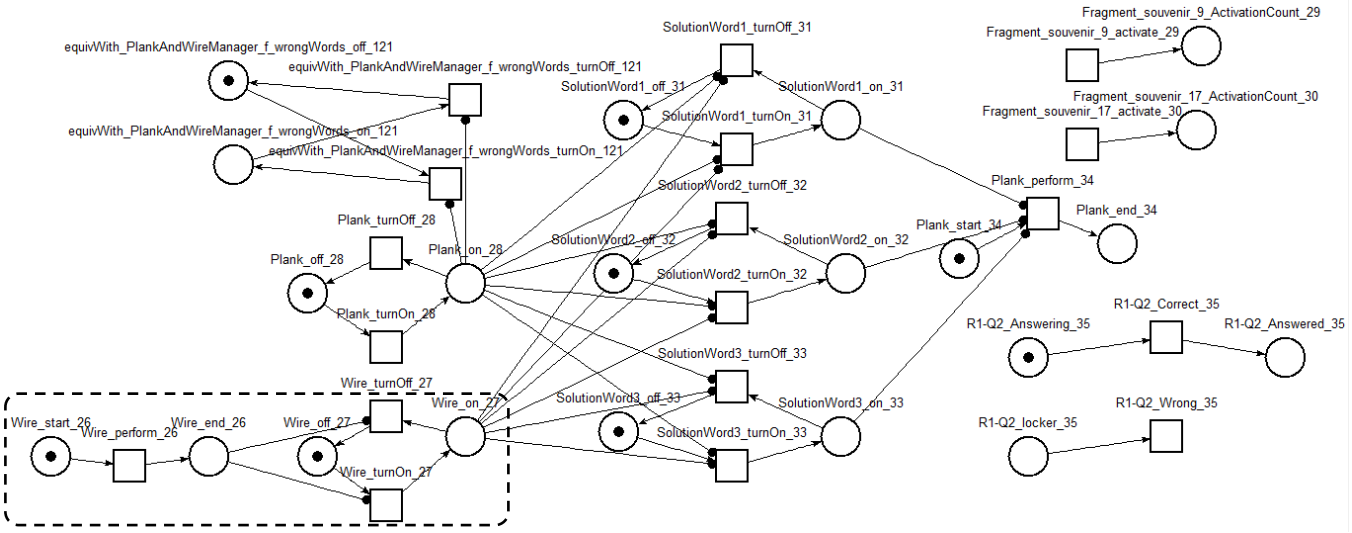

Figure 11: Full Petri net of the pin panel built from "unique action", "activable", "activationCount" and "answerQuestion" patterns (dotted area is detailed in section 5.2)

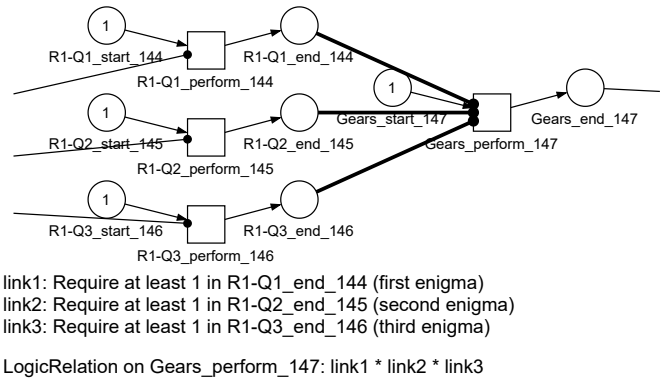

Figure 12: Part of the super-net to show constraints added (bold arcs) to Gears enigma (Gears_perform_147) that will be available when enigmas R1-Q1, R1-Q2 and R1-Q3 will be resolved (end state reached)

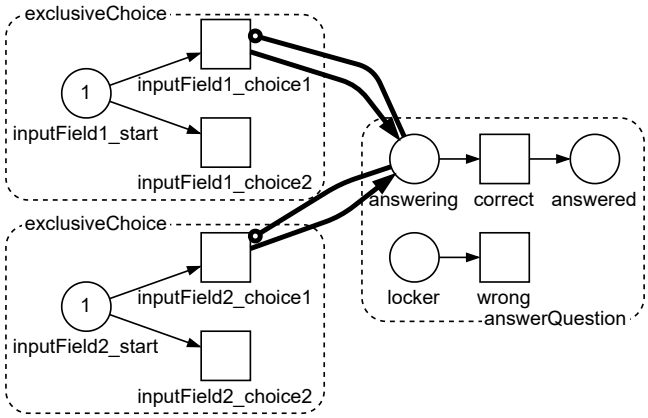

Figure 13: Inhibit choice1 if another was selected before

of the game are modeled and learners' traces are analyzed accordingly. Next steps will be to assess feedback efficiency depending on learner's needs.

Currently we designed a small set of Petri nets patterns and we believe that an interesting research direction is to identify core design patterns of Petri net for a broad games mechanics.

ACKNOWLEDGMENTS

The authors would like to thank Jessica Lament, Séverine Maillet, Sophie Delamasantière-Boutal and John James Boutal for their contributions to reviewing and improving this paper.

REFERENCES

[1] Manuel Araújo and Licínio Roque. 2009. Modeling Games with Petri Nets. In *2009 Digital Games Research Association Conference*. Brunel Univ.

[2] Daniel Balas, Cyril Brom, Adam Abonyi, and Jakub Gemrot. 2008. Hierarchical Petri Nets for Story Plots Featuring Virtual Humans. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (Stanford, California) *(AIIDE'08)*. AAAI Press, 2–9.

[3] Bernard Berthomieu, P.-O Ribet, and F Vernadat. 2004. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research - INT J PROD RES* 42 (07 2004), 2741–2756. https://doi.org/10.1080/00207540412331312688

[4] Andrew Thomas Bimba, Norisma Idris, Ahmed Al-Hunaiyyan, Rohana Binti Mahmud, and Nor Liyana Bt Mohd Shuib. 2017. Adaptive feedback in computer-based learning environments: a review. *Adaptive Behavior* 25, 5 (2017), 217–234.

[5] Joris Dormans. 2011. Simulating Mechanics to Study Emergence in Games. In *Proceedings of the 19th AIIDE Conference on Artificial Intelligence in the Game Design Process (AIIDE'11-19)*. AAAI Press, 2–7.

[6] Joris Dormans. 2012. *Engineering emergence: applied theory for game design*. Ph.D. Dissertation. Institute for Logic, Language and Computation.

[7] L. Gomes and Joao Paulo Barros. 2005. Structuring and composability issues in Petri nets modeling. *IEEE Transactions on Industrial Informatics* 1, 2 (2005), 112–123. https://doi.org/10.1109/TII.2005.844433

[8] Tadao Murata. 1989. Petri nets: Properties, analysis and applications. *Proc. IEEE* 77, 4 (1989), 541–580.

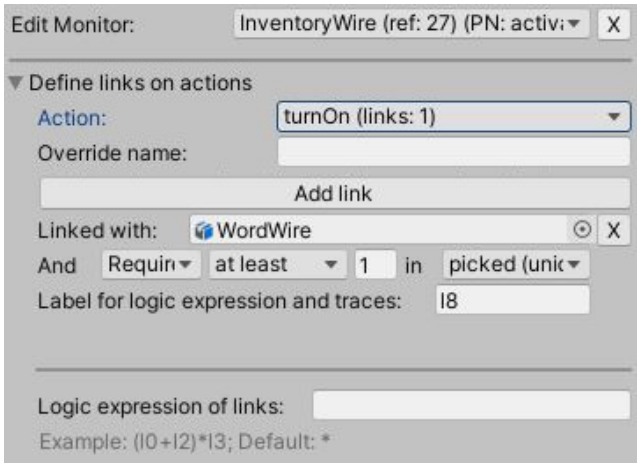

Figure 14: Wire constraint defined in Unity

[9] Mathieu Muratet, Amel Yessad, and Thibault Carron. 2016. Framework for Learner Assessment in Learning Games. In *Adaptative and Adaptable Learning: 11th European Conference on Technology Enhanced Learning, EC-TEL 2016* (Lyon, France), Mike Sharples, Katrien Verbert, and Tomaž Klobučar (Eds.). Springer.

[10] Mathieu Muratet, Amel Yessad, and Thibault Carron. 2016. Understanding Learners' Behaviors in Serious Games. In *ICWL 2016 - International Conference on Web-based Learning*. Rome, Italy. https://doi.org/10.1007/978-3-319-47440-322 Best paper award.

[11] Martin Naedele and Jorn Janneck. 1998. Design Patterns in Petri Net System Modeling. 47–54. https://doi.org/10.1109/ICECCS.1998.706655

[12] James Lyle Peterson. 1981. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[13] Valerie J Shute. 2008. Focus on formative feedback. *Review of educational research* 78, 1 (2008), 153–189.

[14] Weixiang Sun, Tao Li, Wei Peng, and Tong Sun. 2007. Incremental Workflow Mining with Optional Patterns and Its Application to Production Printing Process. *INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS VOL* 12 (01 2007), 45–55.

[15] Mocah team. 2021. Full Petri nets of E-LearningScape. https://github.com/Mocahteam/E-LearningScape/tree/master/completeNets. [Online; accessed 03-September-2021].

[16] Pradeepa Thomas, Amel Yessad, and Jean-Marc Labat. 2011. Petri Nets and Ontologies: Tools for the "Learning Player" Assessment in Serious Games. In *ICALT*. 415–419.

[17] Weng Jie Thong and Mohamed Ameedeen. 2015. A Survey of Petri Net Tools. *Lecture Notes in Electrical Engineering* 315 (01 2015), 537–551. https://doi.org/10.1007/978-3-319-07674-4_51

[18] Kurt VanLehn, Collin Lynch, Kay Schulze, Joel A Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. 2005. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education* 15, 3 (2005), 147–204.

[19] Amel Yessad, Pradeepa Thomas, Bruno Capdevila Ibáñez, and Jean-Marc Labat. 2010. Using the Petri Nets for the Learner Assessment in Serious Games. In *ICWL*. 339–348.