Computing and Informatics, Vol. 40, 2021, 216–248, doi: 10.31577/cai_2021_1_216

FORMAL APPROACH BASED ON PETRI NETS FOR MODELING AND VERIFICATION OF VIDEO GAMES

Franciny M. Barreto

Federal University of Jataí
Department of Computer Science
BR 364, km 195, 3800
Jataí – GO, Brazil
e-mail: franciny@ufg.br

Stéphane Julia

Federal University of Uberlândia
Faculty of Computation
2121 João Naves de Ávila Ave
Uberlândia – MG, Brazil
e-mail: stephane@ufu.br

Abstract. Video games are complex systems that combine technical and artistic processes. The specification of this type of system is not a trivial task, making it necessary to use diagrams and charts to visually specify sets of requirements. Therefore, the underlying proposal of this work is to present an approach based on the formalism of Petri nets for aiding in the design process of video games. The activities of the game are represented by a specific type of Petri net called Work-Flow net. The definition of a topological map can be represented by state graphs. Using Colored Petri nets, it is possible to define formal communication mechanisms between the model of activity and the model of the map. The simulation of the timed models allows then to produce an estimated time that corresponds to the effective duration a player will need to complete a level of a game. Furthermore, a kind of Soundness property related to gameplay in a game Quest can be verified through state space analysis. For a better understanding of the approach, the video game Silent Hill II is used.

Keywords: Petri nets, video games, WorkFlow net, state graph, soundness verification, simulation, CPN tools

Mathematics Subject Classification 2010: 68-M99

1 INTRODUCTION

A video game is a synthesis of code, images, music and animation that come together in the form of entertainment. The creation of video games differs from the creation of classic software because of the pre-production phase and due to the extensive use and integration of multimedia resources. In addition, the creation of video games involves some activities that do not necessarily exist when considering the process of traditional software development. Some of these activities are called game design and level design.

According to [10], game design is a difficult task that combines technical and artistic processes. The game design phase defines the main aspects related to the universe of the game, such as epoch and style, goal to be achieved, etc. In the level design phase, the main actions and objects of the game are defined [4].

A game has to be interactive, entertaining and give controlled freedom to the player. It is important to propose challenges that are neither easy nor too difficult to solve. Furthermore, it is of fundamental importance to ensure that the game experience leads to a succession of goals within a reasonable time [10]. To accomplish all these requirements is not a trivial task, even more in complex games.

Over the years, video games have evolved and become more complex. The increasing complexity of game development highlights the need for tools to improve productivity in terms of time, cost, and quality [13]. It is important to adopt Software Engineering practices to address the challenges that game developers face.

Some studies have shown the use of UML diagrams to show how different objects in a game will interact according to some actions that will be performed by the player [1, 14]. UML diagrams are interesting as they produce an execution structure of the game. However, they do not present in an explicit way the possible scenarios that exist in a mission or at a game level [3].

On the other hand, some studies show the use of formal methods in game modeling, like the ones presented in [2, 10, 3]. In [10], for example, a new type of Petri net called transactions net is presented. The transactions net allows modeling logical and temporal transactions while the topological map of the game is modeled by a type of graph called hypergraph. The authors created a specific communication mechanism between the transactions net and the hypergraph to establish the influence that a model has over the other. However, formal analysis is applied only in the transactions net (because it is represented by a Petri net).

The study in [3] presents a new approach based on WorkFlow nets to specify the scenarios existing at a quest level. In this approach, the sequent calculus of linear

logic was used to prove the correctness of the scenarios a player can execute within a quest of a game. Such an approach only considers models based on the activities of the player and ignores completely the topological map vision of the game.

The research presented in this paper shows an approach where the scenario of a video game is represented by the combination of two types of Petri nets: WorkFlow nets and State Graphs. The WorkFlow nets are used to represent the activities that exist at a game level. The topological map that represents the areas of the virtual world where the player can progress is then represented by a State Graph. To specify the communication between both models, a synchronous communication mechanism is considered. A time version of the models is also presented in order to estimate the time duration a player will need to complete a specific level of the game. The modeling, analysis and simulation of the video game Silent Hill will be implemented on CPN (Colored Petri nets) Tools.

2 THEORETICAL FOUNDATIONS

2.1 Petri Nets

A Petri net is a graphical and mathematical modeling tool that allows one to model, analyze and control discrete event systems that involve parallel activities, concurrency between processes and asynchronous communication mechanisms [9]. Petri nets are formally defined as a directed bipartite graph with two types of nodes called *places* and *transitions*. These nodes can be connected by directed arcs. An arc can only connect a place to a transition or a transition to a place [20]. In graphical notation, the places are represented by circles and transitions by rectangles. Formally, Petri nets can be defined as follows [20].

Definition 1 (Petri nets)**.** A Petri net is a triple $PN = (P, T, F)$, where:

- $\mathbf{P}$ is a finite set of places of PN.
- $\mathbf{T}$ is a finite set of transitions of PN.
- $\mathbf{F} \subset (\mathbf{P} \times \mathbf{T}) \cup (\mathbf{T} \times \mathbf{P})$ represents a set of directed arcs that connect places to transitions and transitions to places.

According to [20], the concepts of input place and output place are then defined in terms of flow relation $\mathbf{F}$ as follows:

- A place p is an input place of a transition t if $(p, t) \in F$. The pre-set' $= \{p \mid (p, t) \in F\}$ defines all input places of a transition t.
- A place p is an output place of a transition t if $(t, p) \in F$. The post-set' $= \{p \mid (t, p) \in F\}$ defines all output places of a transition t.

In a Petri net, the occurrence of an event in the system is represented by the transition to which this event is associated. A transition t can only be fired if each input place of t contains at least one token. A token indicates whether the

condition associated with a place is satisfied. At any time, a place contains zero or more tokens, drawn as black dots. In Figure 1, for example, there are two tokens in the place *wait*, which means two clients are waiting to use the x-ray machine. One token in place *free* indicates that the x-ray machine is free and can be used. In that way, the transition *enter* is enabled and can be fired because it exists at least one token in each of its input places.

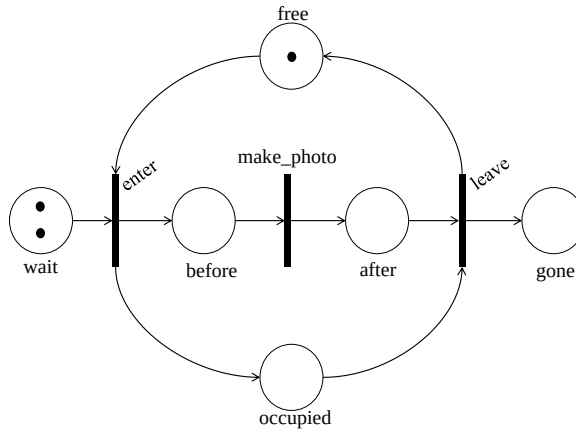


Figure 1. A Petri net for the business process of an x-ray machine

The state of a Petri net, often referred to as marking denoted by M, corresponds to a distribution of tokens over the places of the net. The notation (PN, M) is used to denote a Petri net PN with an initial marking M. Important properties of Petri nets depend directly on the initial marking of the net. Such properties are defined in [9] and are presented as follows.

- Reachability: a marking M_n is said to be reachable from a marking M_0 (initial marking) if there exists a sequence of transition firings that transforms M_0 to M_n. This property ensures if certain states will be reached or not.
- Liveness: a Petri net (PN, M) is said to be live if, and only if, for every reachable state M' and every transition t there is a state M'' reachable from M' which enables t. This property guarantees that the system is deadlock free.
- Boundedness: a Petri net (PN, M) is bounded if, and only if, for each place p there is a natural number n such that for every reachable state, the number of tokens in p is less than n. The net is called *safe* if for each place the maximum number of tokens does not exceed 1.
- Reversibility: a Petri net (PN, M_0) is said to be reversible if it is always possible to return to the initial marking through a sequence of firings, regardless of the marking considered.

2.2 WorkFlow Nets

A Petri net that models a workflow process is called a WorkFlow net (WF-net) ([19, 16]). A WF-net satisfies the following properties:

1. It has only one source place named i and only one sink place named o. These are special places such that the place i has only outgoing arcs and the place o has only incoming arcs.
2. A token in i represents a case that needs to be handled and a token in o represents a case that has been handled.
3. Every task t (transition) and condition p (place) should be in a path from place i to place o.

The formal definition of a WorkFlow net is presented below.

Definition 2 (WorkFlow net)**.** A Petri net $PN = (P, T, F)$ is a WorkFlow net if, and only if:

- There is one source place $i \in P$ such that $\bullet i = \phi$.
- There is one sink place o ∈ P such that $o\bullet = \phi$.
- Every node $x \in P \cup T$ is on a path from i to o.

A WF-net has one input place i and one output place o because any case handled by the procedure represented by the WF-net is created when it enters the workflow management systems and is deleted once it is completely handled by the workflow management systems, i.e., the WF-net specifies the life-cycle of a case. The third requirement in Definition 2 has been added to avoid "dangling tasks and/or conditions"; in other words, tasks and conditions which do not contribute to the processing of cases [19].

A business process specifies which tasks need to be performed and in which order to execute them. Modeling a business process in terms of a WF-net is quite straightforward: tasks are modeled by transitions, conditions are modeled by places, and cases are modeled by tokens [19]. An example of WorkFlow net is presented in Figure 2.

Figure 2 illustrates the workflow process which takes care of the processing of claims related to a car damage presented in [18]. The tasks required to process the claims are: *check_insurance*, *contact_garage*, *pay_damage* and *send_letter*. The tasks *check_insurance* and *contact_garage* determine whether the claim is justified. These tasks may be executed in any order. If the claim is justified, the damage is paid (task *pay_damage*). Otherwise a letter of rejection is sent to the claimant (task *send_letter*). The tasks are modeled by transitions. The two tasks *check_insurance* and *contact_garage* may be executed in parallel. Thus, there are two additional transitions: *fork* and *join*. The places *p1*, *p2*, *p3*, *p4* and *p5* are used to route a case through the procedure in a proper manner.

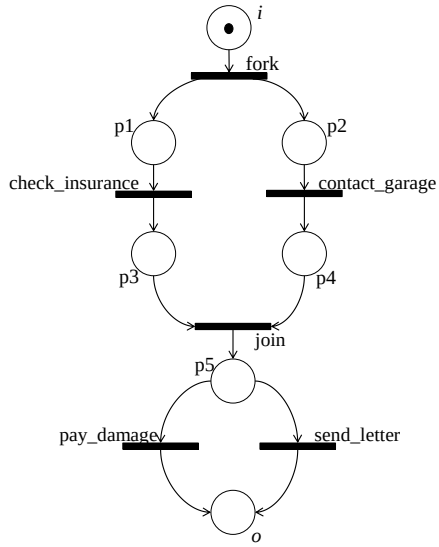


Figure 2. Example of a WorkFlow net

2.3 Soundness Property

Soundness is a correctness criterion defined for WF-nets. A WF-net is Sound if, and only if, the following three requirements are satisfied [16]:

1. For each token put in the place i, one and only one token will appear in place o.
2. When the token appears in place o, all the other places are empty for this case.
3. For each transition (task), it is possible to move from the initial marking to a marking in which that transition is enabled, i.e., there are no dead transitions.

The Soundness property is related to the dynamics of a WF-net. The first requirement states that starting from the initial marking i, it is always possible to reach the marking with one token in place o. The second requirement states that the moment a token is put in place o, all the other places should be empty. The third requirement states that for each transition t, it is possible to reach (starting from i) a marking where t is enabled [19].

Following, the formal definition of soundness property in the WF-net context, proposed in [19, 21], is presented.

Definition 3 (Soundness)**.** A process modeled by a WF-net $PN = (P, T, F)$ is Sound if, and only if:

- For every marking M reachable from marking i, there exists a firing sequence

leading from marking M to marking o. Formally:

$$\forall_M (i \xrightarrow{*} M) \rightarrow (M \xrightarrow{*} o).$$

- Marking o is the only marking reachable from marking i with at least one token in place o. Formally:

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \rightarrow (M = o).$$

- There are no dead transitions in $(PN,\, i)$. Formally:

$$\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{*} M'.$$

In [19], a method was proposed to verify the soundness property of a WF-net. Given a WF-net $PN = (P, T, F)$, one must decide whether PN is Sound. For this, an extended net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is created. $\overline{PN}$ is the Petri net obtained by adding an extra transition t^* which connects places o and i. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows ([17]):

- $\overline{P} = P$,
- $\overline{T} = T \cup t^*$,
- $\overline{F} = F \cup \{\langle p, t^* \rangle, \langle t^*, i \rangle\}$.

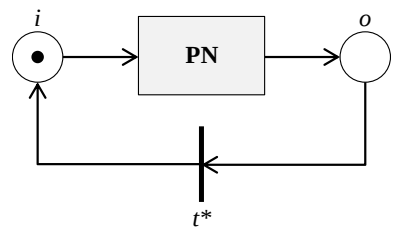


Figure 3. Example of extended Petri net

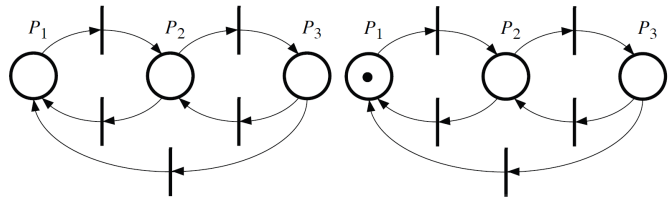
Figure 3 illustrates the relation between PN and $\overline{PN}$. The following theorem can be proven.

Theorem 1. A WorkFlow net PN is Sound if, and only if, $(\overline{PN}, i)$ is live and bounded.

The proof of this theorem can be found in [17]. Thus, the verification of the Soundness property boils down to checking whether the extended Petri net $\overline{PN}$ is live and bounded. This means that standard Petri-net-based analysis tools can be used to decide Soundness. An overview about WF-net can be found in [18, 19, 16].

2.4 State Graphs

An unmarked PN is a state graph if and only if every transition has exactly one input and one output place [11], as illustrated in Figure 4 a).



a) Example of a state graph unmarked

b) Example of a state graph marked with one token

Figure 4. Example of a state graph

A marked Petri net known as a state graph will be equivalent to the state graph in the classical sense (representing an automaton which is in only one state at a time) if, and only if, it contains exactly one token located in one of the places of the set P [11]. Figure 4 b) shows an example of a state graph with one token. It is important to note that in a state graph, the weight of all the arcs is 1.

2.5 Colored Petri Nets

Petri Nets are traditionally divided into low-level Petri nets and high-level Petri nets. Low-level Petri nets are characterized by simple tokens (natural numbers associated to places) that generally indicate the active state of a system or the availability of a resource. High-level Petri nets are aimed at practical use, in particular because they allow the construction of compact and parametrized models.

Classic Petri nets belong to the class of low-level Petri nets. They allow the representation of parallelism and synchronization, thus they are appropriate for the modeling of distributed systems. However, when classic Petri nets are used for the modeling of process, the size of very large and complex systems became an issue of major complication. In that way, one disadvantage is that classic Petri nets fall short if they are used to precisely model complex systems, making then unsuitable for the modeling of systems having large state spaces or a complex temporal behavior [15]. Then, many extensions of basic Petri net models arose from the need to represent these complex systems. One of them is the Colored Petri Net (CPN).

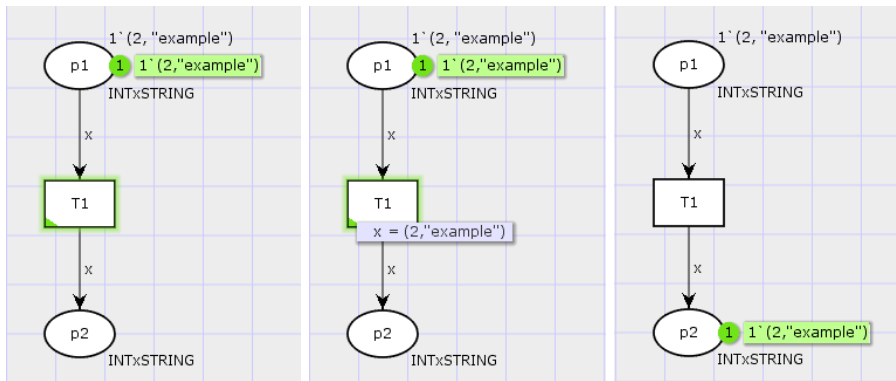
The idea of CPN is to put together the ability to represent synchronization and competition of Petri nets with the expressive power of programming languages with their data types and the concept of time. Colored Petri Nets (CPNs) belong then to the class of high-level Petri Nets, and they are characterized by the combination of Petri nets and a functional programming language [8], called CPN ML. Thus, the

formalism of Petri nets is well suited for describing concurrent and synchronizing actions in distributed systems, whereas the functional programming language can be used to define data types and manipulation of data [6].

The formal definition of CPN [6] is presented bellow.

Definition 4 (Colored Petri Net)**.** A non-hierarchical Colored Petri Net (CPN) is a nine-tuple CPN = (P, T, A, Σ, V, C, G, E, I) where:

- **P** is a finite set of places.
- **T** is a finite set of transitions T such that $P \cap T = \emptyset$.
- **A** $\subseteq p \times t \cup t \times p$ is a set of directed arcs.
- Σ is a finite set of non-empty color sets.
- **V** is a finite set of typed variables such that Type[v] $\in \Sigma$ for all variables $v \in V$.
- **C**: $p \rightarrow \Sigma$ is a color set function that assigns a color set to each place.
- **G**: $t \rightarrow \text{EXPR}_v$ is a guard function that assigns a guard to each transition t such that $Type[G(t)] = Bool$.
- **E**: $a \rightarrow \text{EXPR}_v$ is an arc expression function that assigns an arc expression to each arc a such that $Type[E(a)] = C(p)_{MS}$, where p is the place connected to the arc a.
- **I**: $p \rightarrow \text{EXPR}_\emptyset$ is an initialization function that assigns an initialization expression to each place p such that $Type[I(p)] = C(p)_{MS}$.



a) Example of a simple CPN b) During the firing of transition *T1* c) After the firing of transition *T1*

Figure 5. Elements of a Colored Petri Net

The states of a CPN are represented by means of places. Every place has a type associated which determines the kind of data that the place may contain. Each place will contain a varying number of tokens. Every token has a data value, that

is known as a color [15], and belongs to the type associated with the place. These colors do not just mean colors or patterns, they can represent complex data types [6]. Figure 5 illustrates an example of the basic elements of a CPN.

The CPN in Figure 5 a) has two places called *p1* and *p2*. The places have the type (color set) $INT \times STRING$, as well as the variable x associated to the arcs of the net. This type is formed by cartesian product of the color sets *INT* (integers) and *STRING*. The places accept only tokens of this same type. In that way, the inscription *1'(2, "example")* corresponds to one token whose attributes are given by the integer *2* and the string *"example"*.

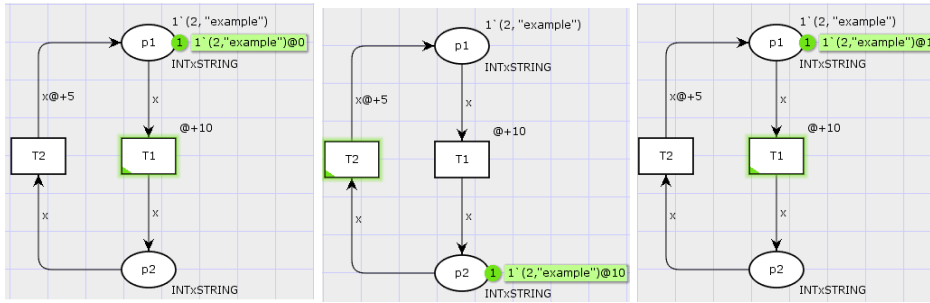
In the example of Figure 5 a) the transition *T1* will be enabled only if there is at least one token in place *p1* (input place of *T1*). During the firing of a transition in a CPN model, the variables of its input arcs will be replaced with the token value. Figure 5 b) shows that variable x was associated with the value (2, "example"). After firing *T1* the place *p2* has one token. Since there is no token in its input place, *T1* is not enabled anymore (Figure 5 c)).

CPN models allow adding time information to investigate the performance of systems. For this, a global clock used to represent model time was introduced. The clock values may either be discrete or continuous [5]. In a timed CPN model the token can carry a time value, called timestamp. To calculate the timestamps to be given to a token it is necessary to use time delay inscriptions attached to the transition or to the individual output arcs [6]. A time inscription on a transition applies a time delay to all output tokens created by that transition. On the other hand, a time inscription on an output arc applies a time delay only to tokens created at that arc [6].

To exemplify a firing transition in a timed CPN, consider the Figure 6 a). Transition *T1* represents an operation which takes 10 time units. Thus, *T1* creates timestamps for its output tokens by using time delay inscriptions attached to the transition (inscription $@ + 10$). In that way, every time *T1* is fired its output token timestamps will be increased by 10 time units. Figure 6 b) illustrates *T1* after firing. The outgoing arc to *p1* has a time delay expression $@ + 5$. Thus, the timestamp given to the tokens created on this output arc is the sum of the value of the global clock and the result of evaluating the time delay inscription of the arc, as it can be seen in Figure 6 c).

Another advantage of CPN model is that it can be structured into different related modules. The concept of module in CPN is based on a hierarchical structuring mechanism which supports bottom-up as well as top-down working style. The basic idea behind hierarchical CPN is to allow the modeler to construct a large model by combining a number of small CPN into a single model [5]. According to [15] it facilitates the modeling of large and complex systems, such as information systems and business processes.

CPN hierarchy also offers a concept known as fusion places. This concept allows the modeler to specify that a set of places are considered to be identical. Such places are called fusion places and a set of fusion places is a fusion set. Anything that happens to one place of the set also happens to the other places of the set.



a) Initial marking of a timed CPN model b) After the firing of transition *T1* c) After the firing of transition *T2*

Figure 6. An example of a timed Colored Petri Net

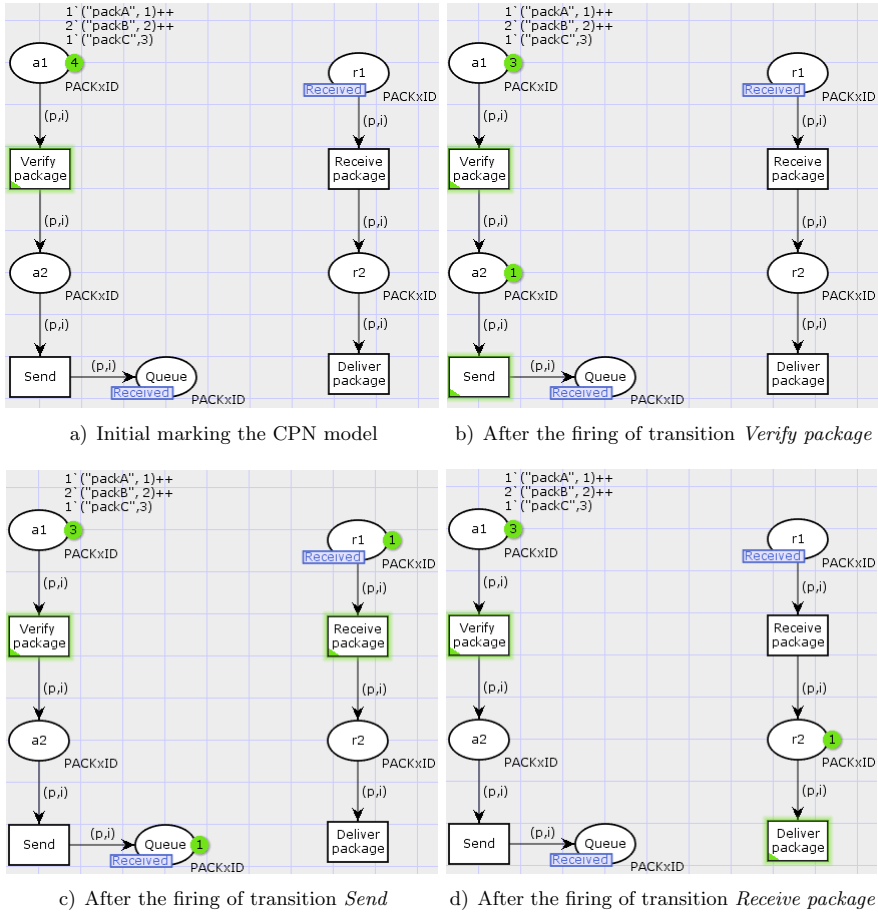
Thus, when a token is added/removed from one of the places, an identical token will be added/removed in all the other places of the fusion set.

Figure 7 illustrates an example of CPN with fusion places. This model represents a procedure where packages are verified and sent to another procedure to be delivered. Figure 7 a) shows the initial marking of the net. After firing the transition *Verify package*, the package will be sent, represented by transition *Send* (Figure 7 b)). After firing the transition *Send*, the token will be added in place *Queue* which represents a queue. The places *Queue* and *r1* are defined as fusion places and they are marked with a fusion tag named *Received*. This tag represents to which fusion set these places belong. Thus, if *Queue* has one token, the place *r1* will also have one token as illustrated in Figure 7 c). After firing transition *Receive package*, the token in *r1* will be consumed (Figure 7 d)) and the transition *Deliver package* can be fired.

When all members of a fusion set belong to a same page (the same part of a CPN model) and that this page only has one instance, a fusion place is nothing more than a drawing convenience that allows the user to avoid too many crossing arcs in his visual model [6]. Thus it is possible to simplify the net graphical structure without changing its meaning.

The practical application of CPN modeling and analysis heavily relies on the existence of computer tools supporting the creation and manipulation of CPN models. CPN Tools [5] is a tool suited for editing, simulating and providing state space analysis of CP-net models.

In this paper, the CPN Tools is used to represent graphically and analyze the proposed models, and to simulate the timed versions of the model. By performing analysis and simulation of the proposed models, it will make possible to investigate different game scenarios and to explore qualitatively and quantitatively the global behavior of the game.



a) Initial marking the CPN model

b) After the firing of transition *Verify package*

c) After the firing of transition *Send*

d) After the firing of transition *Receive package*

Figure 7. An example of a Colored Petri Net with fusion places

3 RELATED WORK

In [10], a new method to aid in the creation process of video games is presented. The authors approach models to represent the logic of the game (sequence of actions) and virtual space. To model the logic of the game, the authors used the formalism of Petri nets. In this approach, a Petri net called the Transaction Net determines the beginning and the end of each player action. Thus, each model represents a set of activities that can be performed by the player in the game. To model the virtual space, the authors used hyper-graphs. Each node of the hyper-graph represents a region of the game where the actions are performed, and the hyper-edges represent the paths between those regions. In order to unite the two structures and represent the game in a global context, the authors created a mechanism called

connections. This mechanism attempts to replace a hyper-edge of the hyper-graph by the reachability tree of a Petri net. Each time a task is performed, a place on the topological map is released and the hyper-edge replaced. The verification of the game model is treated according to each formalism.

In [3] a new approach based on a particular type of Petri net, called WorkFlow net, is presented to specify existing scenarios in a game. In this approach, the authors used a WorkFlow net to represent the flow of activities that must be performed by the player in order to achieve a specific goal in the game. This flow of activities is associated with the notion of quest, i.e., a mission that the player must perform. According to [3], in terms of the model, each quest is a subprocess of a larger WorkFlow Net. The integration of several quests form a net of quests and it is through this that the overall result of the analysis of the game model is established. As each quest is a subprocess, in case of a change in a quest already studied, a new study of good properties will be done only for the quest that has changed. In this approach, the authors performed a qualitative analysis using linear logic. According to the authors, the translation of the models into linear logic trees has the objective of proving the soundness property of the net that corresponds to the consistency of the scenario modeled from the point of view of the game.

The concept of Petri nets in game modeling is also used in [12] to present an approach that operates in the early stages of game design, detecting structural errors in singleplayer and multiplayer games. To differentiate the singleplayer games from multiplayer, [12] used Colored Petri nets that can assign types to the tokens and use guard functions in the transitions. All elements of the game are then mapped to appropriate Petri net constructors. For example, the rooms of the game (e.g., a room or a depot) are represented by places in the Petri net, as are actions (e.g., opening a door) and variables representing boolean values. Players are represented by the tokens and the events of the game by the transitions. The creation of the Petri net is added as an extension of the StoryTec tool [12]. Thus, any game that is created with this tool can be automatically transformed into a Colored Petri net. The generated net is then exported to the XML format (Extensible Markup Language) that can be read by the CPN Tools. According to [12] when model analysis is done in CPN Tools it is possible to identify structural errors like deadlocks (situations in which players cannot change the state of the game), livelocks (situations in which players can change the state of the game but cannot reach the end), unreachable scenes (situations in which players cannot reach a scene under any circumstances) and impossible actions (situations in which actions never can be achieved due to unsatisfied conditions).

Most of the work related to this research show the use of Petri nets as efficient modeling language to formally specify and analyze video games. The approach presented in [10], for example, is interesting as it defines diagrams to represent aspects of the game that are important for its creation process, thus facilitating the study of gameplay. However, verification happens in the logical model or in the topological map since they are distinct formalisms that do not allow the integration of the two models in a single view. In the work presented in [3] the authors did

not present a model to represent the virtual world map of the game where the player's actions are performed. In addition, the authors presented a qualitative analysis using a formalism different from the Petri nets. Finally, in [12], despite the automatic generation of the model, this approach presented produced a complex model. Thus, the model needs to go through optimization strategies to deal with the state explosion problem and be formally analyzed.

4 MODELING GAME LEVELS

The video game Silent Hill II [7] will be used to illustrate the approach presented in this paper. The modeling method presented in this section uses the case of a real game. However, the same modeling principle can be applied to any singleplayer game that has activities and the presence of a virtual topology.

4.1 Logical Model

For the representation of the logical model, it is necessary to consider the concept of level. The use of the term level in video games has been employed for a long time and, generally, can be approached in two ways. The first associates the term level with the difficulty of the game phase. The second associates level to a certain stage of the game. In this work, the second approach is used. Thus, a level is considered a part of the game.

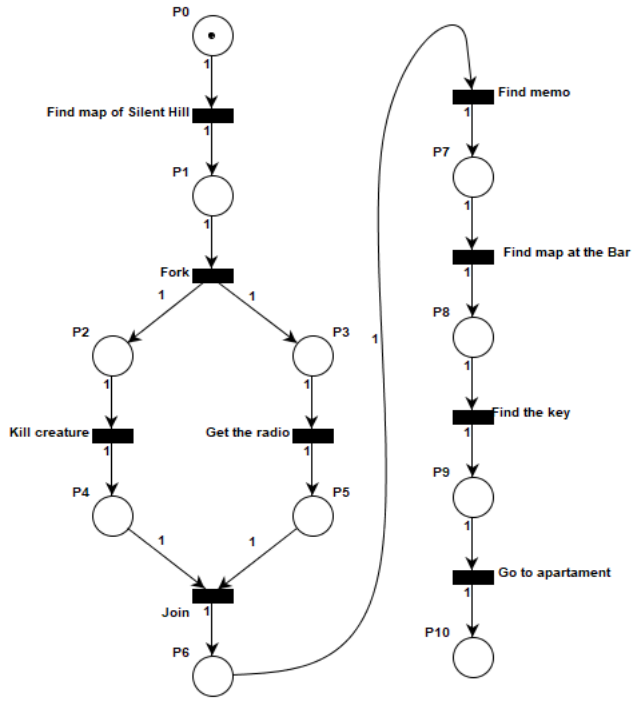
Most games can be structured into a group of levels. Every game has one main goal that the player must reach in order to win and each level has a specific goal associated to the level. To achieve this goal the player must perform a certain sequence of tasks. These tasks must be performed by the player sequentially or simultaneously to a certain extent. In addition, some tasks are mandatory and some optional. After all level tasks are performed correctly, the goal is reached and the player can go to the next level.

Since a game consists of several levels and a level consists of a set of activities, the WorkFlow nets seem suitable to produce a logical model of a game level. Indeed, a game level is similar in its structure to the classic representation of a workflow process. Both have a beginning and a final goal that will be reached after performing some activities. Thus, WorkFlow nets will be well adapted to model the routing structure of the activities a player will have to performed in a specific level of a video game.

To model a game level, it is necessary to identify first the activities of the level. To complete the first level of Silent Hill II, the player must perform the following sequence of activities:

1. Find the Silent Hill map on the parking observation deck.
2. Kill the creature and get the radio at the tunnel.
3. Find the memo at the trailer.

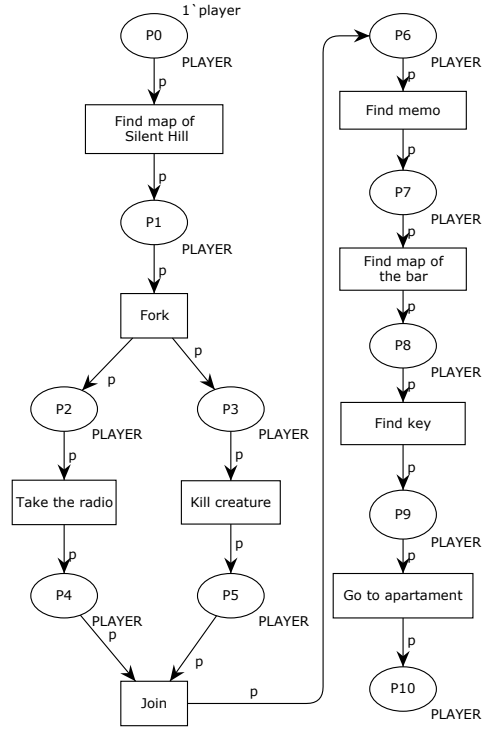
4. Find the second map at Neely's Bar.
5. Find the key on a corpse at Martin street.
6. Go to Wood Side Apartment.



a) Logical model of the first level of Silent Hill II

After performing these activities, the player will complete the first level and will be able to move to the next level. The WorkFlow net in Figure 8 a) represents the logical model of the first level of the game.

The activities are associated with the transitions and the conditions with the places. The initial place is *P0* and represents the beginning of the first level. The token in *P0* represents the player. The place *P10* represents the end of the level. The first task that the player has to perform is *Find map of Silent Hill*. The tasks *Get the radio* and *Kill creature* belong to a parallel routing. That means the player can execute them in any order. *Find memo*, *Find map of the bar*, *Find key* and *Go to Apartment*, are tasks which have to be executed in sequence and then belong to a sequential routing of the WF-net. *Go to Apartment* corresponds to the last task of the first level. After completing all the tasks, the player ends the first level, reaching the final place *P10*.



b) Logical model of the first level of Silent Hill II represented by a Colored Petri net model of the CPN Tools

Figure 8. Logical model of Silent Hill II

Figure 8 b) shows the Colored Petri net model of the Workflow net presented in Figure 8 a) and adapted to be directly implemented on the CPN Tool. In the presented approach, the same color set *PLAYER* is associated to all places of the model. The inscription *player* then represents the value attached to the token that represents a specific player in the corresponding level. The variable p associated to the arcs of the model belongs to the same type *PLAYER* and can receive tokens of the same color set. Thus, the token *1'player* can go through the entire CPN, from the beginning to the end, representing the evolution of the player in the game.

In most of games, there exist activities based on finding objects, solving puzzles and interacting with game characters named NPC (Non-Player Character). Some specific interactions propose challenges where the player must win a competition against a NPC in order to perform the next activity. In case of failure (death of the player during a fight for example) the player can have to perform the same activity more than once. In the first level of the Silent Hill II, such a situation exists. The activity *Kill creature* (Figure 8 b)) consists in fighting a monster (a NPC). If the

player does not win, he will have to perform the same activity over again, i.e., the player will have to fight repeatedly against the monster until he manages to defeat him.

The logical model of the level, the activity *Kill creature* is part of a parallel route where the other activity is *Take the radio*. Thus, these activities can be executed in any sequence. In particular, four option can be considered:

1. The player successfully performs the activity *Take the radio* first, and then the activity *Kill creature*. After that, he moves to the next activity following the game flow;
2. The player successfully performs the activity *Kill creature* first and then the activity *Take the radio*. After that, he moves to the next activity following the game flow;
3. The player performs the activity *Kill creature* and loses. As a consequence, the game is restarted to the checkpoint that corresponds to the conclusion of the previous activity (*Find map of Silent Hill*);
4. The player successfully performs the activity *Take the radio* first, and then does not manage to complete the activity *Kill creature*. As a consequence, the game is restarted to the checkpoint that corresponds to the conclusion of the previous activity (*Find map of Silent Hill*).

The Figure 9 illustrates the final logical model of the level. The activities *Take the radio* and *Kill creature* are represented by the transitions of the same name. The transition *L2* represents the option 3 and the transition *L1* the option 4. *L2* is fired when places P2 and P3 are marked (which corresponds to option 3). *L1* is fired if a token is in *P3*. After the firing of *L1*, *L3* is fired and a token is produced in place *P11* and in place *P4* (which corresponds to option 4). Such a control structure corresponds to a kind of iterative route in a Workflow net.

It is important to note that each activity of a game takes a minimum duration to be performed. The duration depends, basically, on the player experience. More experienced players tend to perform the game challenges more easily than the less experienced ones. A player experience can come from experience based on previous games or after repeating the same challenge over and over again. Thus, the more a player executes the proposed activities, the more experience he will obtain. After gaining some experience, it is more likely that a player will perform given activities again in a most efficient way, spending less time every time he runs it.

The duration of each game activities is important because from it, it will be possible to calculate the average time required for a level to be completed by a player. Therefore, it is necessary to represent the duration of the activities on the logical model in an explicit manner. For this, a random time function associated to each activity of a level will be considered. Such a function will simulate then the duration a player needs to perform a specific activity.

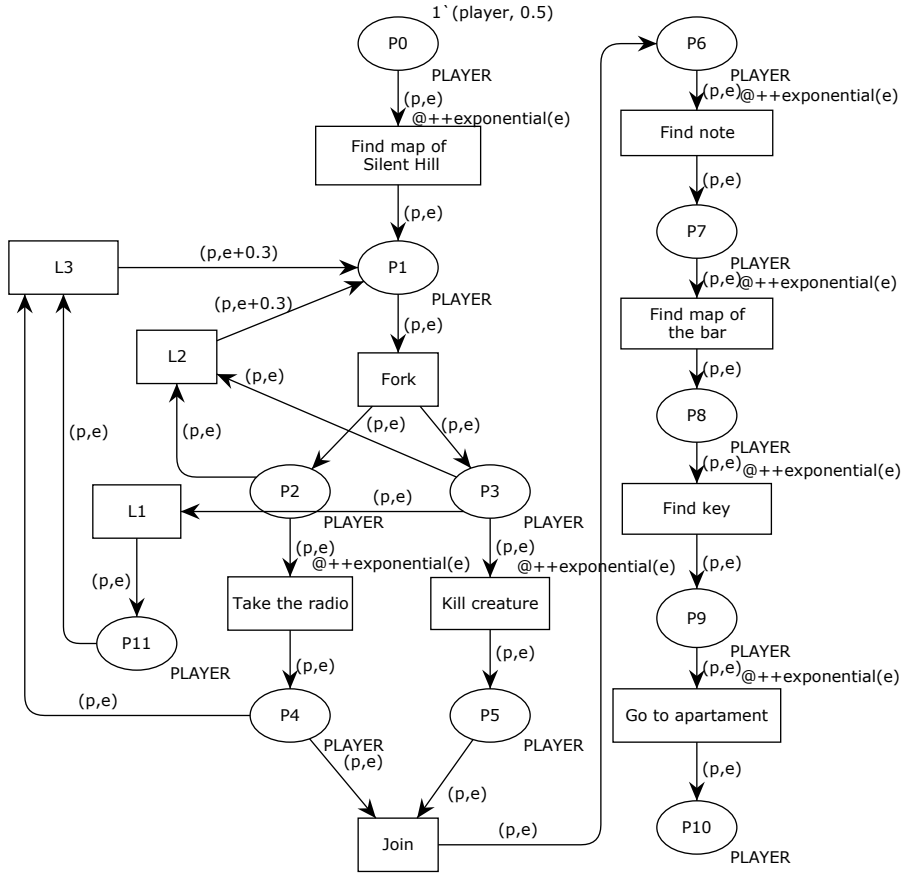


Figure 9. First level of Silent Hill II with an iterative activity

In the proposed approach, a negative-exponential probability distribution will be used. This function is one of the most widely used random distribution functions for simulating inter-arrival times in most simulation problems [6]. The function is based on an exponential parameter r, where r is a positive real number. The CPN Tool function *exponential*(r) produces a value based on the distribution exponential with mean $1/r$, for $r > 0$.

In Figure 9, the exponential function is associated with each activity. The parameter of the function is given by the real parameter e. This parameter is based on the experience of the player. Initially, e has the value 0.5[1]. This value may change

[1] At first, this value was defined only for test purposes. As future work, it will be interesting to consider a more accurate study based on statistical data to determine e with more precision.

according to the evolution of the game. The inscription $@++exponential(e)$ assigns a delay to the token used to fire the transition corresponding to the considered activity.

When a player repeats a same activity, as it is the case with the activity *Kill creature* of the logical model, his experience increases. Such a statement is expressed by the inscription $(p, e + 0.3)$ associated to the outgoing arcs of transitions *L3* and *L2*. The value of the parameter e is increased by 0.3. In that way, the duration (mean duration) to perform the activity *Kill creature* each time the player is killed will be shorter, expressing the accumulated experience.

4.2 Topological Model

It is in the virtual world of a game that activities are performed. Thus, it is also important to describe the topological properties of the virtual game world along with the evolution of the player within it [10].

A game has a set of areas where the player has to fulfill specific challenges. These areas do not change during the game.

It was in [10] that a formal definition of the topological map concept existing in video games appears for the first time. The model proposed by the authors was based on a kind of graph with the possibility of adding dynamically some arcs connecting adjacent areas after the player managed to liberate some kind of passage. The problem with this kind of approach is clearly the difficulty of formally implementing communication mechanisms between the logical model and the topological model. In particular, when the logical model is represented by a Petri net, it will be particularly difficult to implement formal communication mechanisms between the logical model and a topological model represented by a kind of graph without the notion of dynamic marking that exists in Petri net models. Therefore, in order to produce formal communication mechanisms between the logical and the topological model of video games, the approach presented in this article considers the representation of the topology of the virtual world of the game using a kind of Petri net model called state graph.

The semantic of a state graph makes the modeling of the topological map of a game easy. Each region of the virtual world is called an area. In the corresponding state graph, a specific area is modeled by a specific place. The boundaries between areas are represented by simple transitions. The location of the player is then represented by a token in a specific place and the arc orientation represents in which direction the player may go between two adjacent areas.

To illustrate this approach, the map of the first level of Silent Hill II in Figure 10 can be considered. The numbers represent special subsets of the virtual world where the player has to fulfill specific tasks. These regions do not change during the game. They are named as follows:

1. Observation deck,
2. Forest,

3. Church,
4. Backyard,
5. Tunnel,
6. Trailer,
7. Bar,
8. Martin Street,
9. Wood Side Apartment.

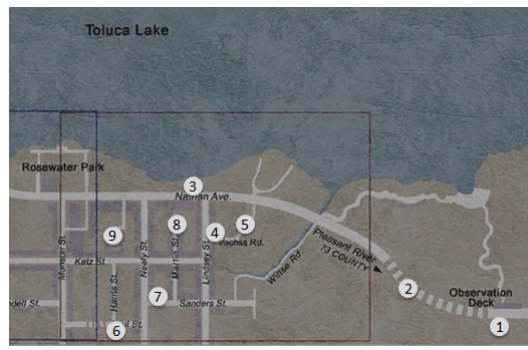


Figure 10. Virtual world areas of the first level of Silent Hill II

In Figure 11, the formal representation of the topological world of the first level of Silent Hill II is presented. Each region of the map is represented by places of the same name in the state graph. The transitions represent the boundaries between adjacent areas. The token in place *Observation Deck* represents the current location of the player at the beginning of the game. In the topological map, firing a transition means that the player moves from one area to another.

Figure 12 shows the same topological world of Figure 11, but adapted to the Colored Petri net of the CPN Tools. In Figure 12, all the areas of the game have the type *PLAYER* and the token is represented by the inscription *1'player* at the place *Observation Deck*.

On the topological map, a transition firing means that the player moved from one area to another. He can pass through areas quickly or slowly. Thus, minimum and a maximum durations for passing from one area to another have to be considered in the time model. To simulate the player's movement on the map, a random time function is then added to each transition of the state graph.

The function $uniform(a, b)$ produces a random number between parameters *a:real* and *b:real*. The probability distribution is uniform, i.e., any value between parameters a and b has the same probability to accur. For $b > a$, $uniform(a, b)$ produces then a value from a uniform distribution with mean $(a + b)/2$ [15]. The

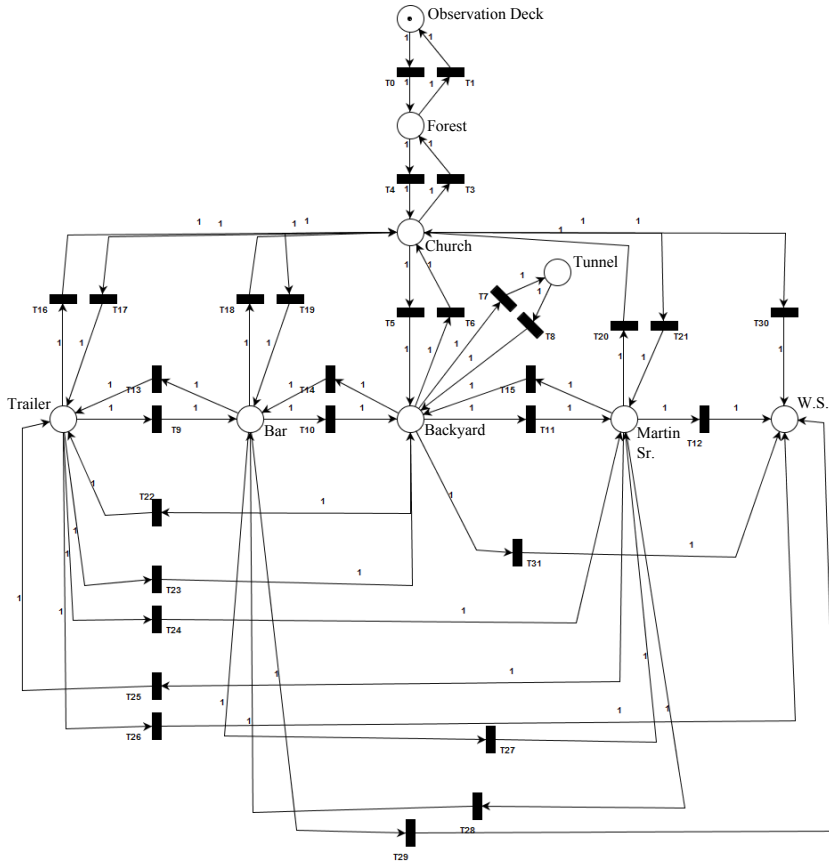


Figure 11. Topological map of Silent Hill II represented by a State Graph

uniform function seems suitable to represent the duration that exists on the topological map as it has a minimum and maximum parameter, which seem consistent with how long a player will spend while moving from one area to another.

Figure 13 illustrates the time topological model. In order to keep the model as clean as possible, the inscription $@++Time()$ was associated with each transition of the topological model. $Time()$ corresponds to a simplified notation for the function $uniform(a, b)$ and assigns a delay to the token used to fire the transition. The two parameters, a and b, and the random value produced, are all real numbers. The chosen parameters for this example are[2] $a = 0.1$ and $b = 1.0$. In that way, in case of

[2] These parameters can be updated according to the specifications of the game designer.

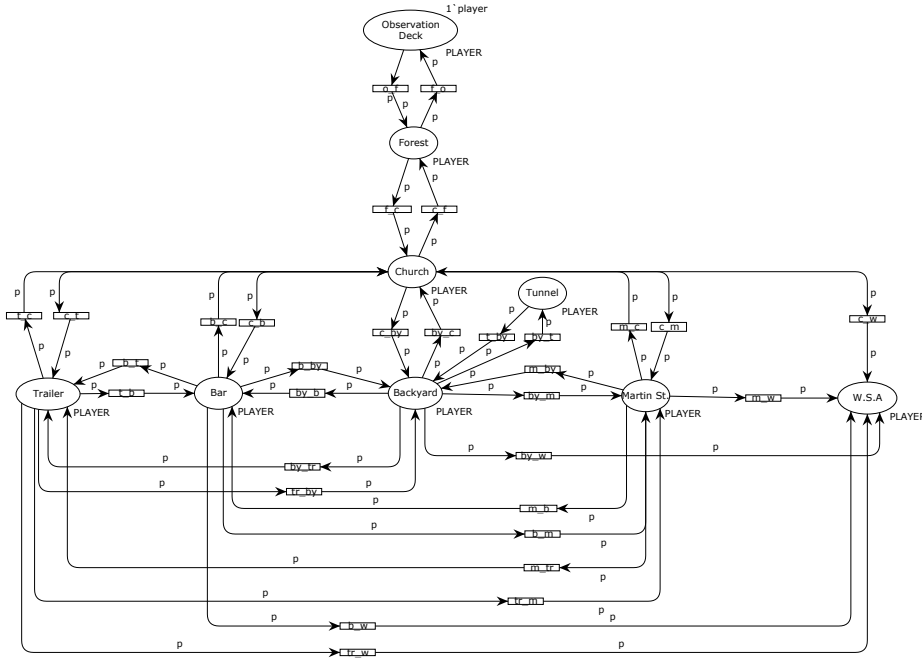


Figure 12. Topological map of the first level of Silent Hill II adapted to CPN Tools

adjacent areas, the player will spend between 0.1 and 1.0 time units to move from one area to another.

4.3 Communication Mechanisms

Generally, at the beginning of the game the player cannot access all the connected areas. To pass from one area to another the player has to respect some requirements (conditions) that will depend on some of the activities associated to the logical model of the level. Such conditions in most of the cases will correspond to find an object (like a key for example) necessary for passing from one area to another. Eventually, some conditions will simply correspond to the completion of a task the player must perform (like kill a creature for example), not necessarily producing a specific game item. In any case, such situations will imply in a kind of communication between the logical model and the topological model.

For example, in the game Silent Hill II, to pass from the area *Observation Deck* to the area *Forest*, the player must find the map of Silent Hill. Thus, the first activity on the logical model is *Find the map*. The player can only performed this activity if he is in the corresponding area, which is *Observation deck*. Thus, it is necessary to establish a communication mechanism between both models in order to guarantee that the player is at the right place (where he must be to perform the activity) at

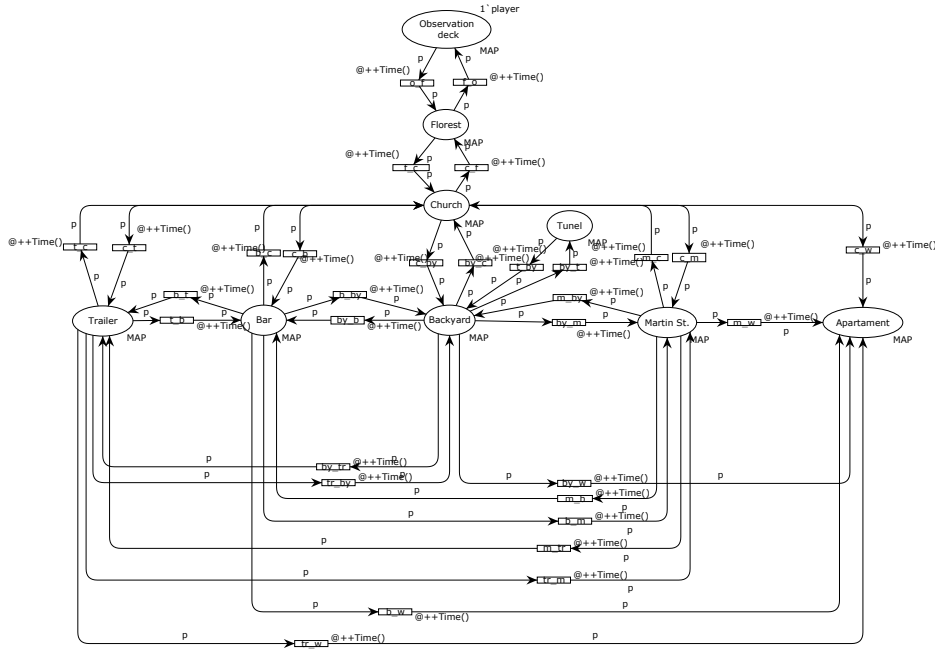


Figure 13. Timed topological model of the first level of Silent Hill II

the right time (when he must perform the activity in accordance with the logical model which fixes the sequence of activities of the level).

Figure 14 is an example of communication mechanism between the logical model and the topological model. To execute the first activity *Find map of Silent Hill*, the player needs to be on the area *Observation deck* (the place *Observation deck* must be marked). After the completion of the activity *Find map of Silent Hill*, the first condition of the level is verified. The first condition of the game is represented by the places *Map found* (in the logical model) and *Condition 1: find the map* (in the topological model). Once with the map, the player will be able to pass from *Observation Deck* to *Forest*. After a condition becomes true (a token produced in a condition place), the corresponding condition place will continue marked all the time. This means that once a passage between two adjacent areas is liberated, it will continue open until the level is completed.

Such a procedure to link both models can be seen as a kind of synchronous communication mechanism. In order to keep both models distinct (at least visually), an interesting approach is to use the fusion place concept proposed by the CPN Tools. Fusion places in our approach are used to visually implement the communication mechanisms between the logical model and the topological map, as shown in Figure 15. Figure 15 is then equivalent to Figure 14 using the CPN Tools concept of fusion places, maintaining both models distinct.

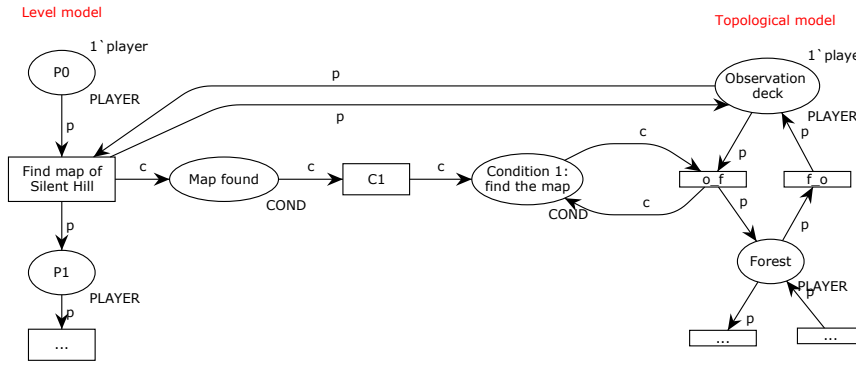


Figure 14. Communication mechanism between the logical model and the topological model

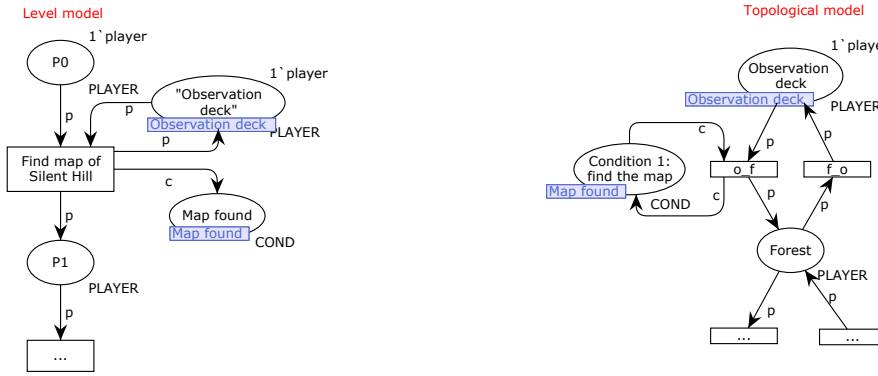


Figure 15. Communication mechanism implemented with the fusion place concept

To distinguish condition places from other places in the model, the color set *COND* was created. This type is associated with all places that represent a condition in the topological model. In Figure 15, the places *Map found* and the place *Condition 1: find the map* have the type *COND*. These places are marked with the tag *Map found* (represented by a blue rectangle). The members of a fusion set have the same fusion tag. When a token is produced in one of the places of a same fusion set, the same token is then produced in all other places of the fusion set. For example, in Figure 15, after the firing of transition *Find map of Silent Hill*, a token is produced in the place *Map found* of the logical model. The same token is then automatically reproduced (cloned) in the corresponding place *Condition 1: find the map* of the topological model.

Figure 16 represents both models (logical and topological) with their synchronous communication mechanisms of the first level of Silent Hill II. It is important

to notice that both models, as well as the communication mechanisms, use a same formalism (a Colored Petri net); then, it will be possible to implement some kind of qualitative and quantitative analysis techniques on the global model (logical + topological) of the level of the game.

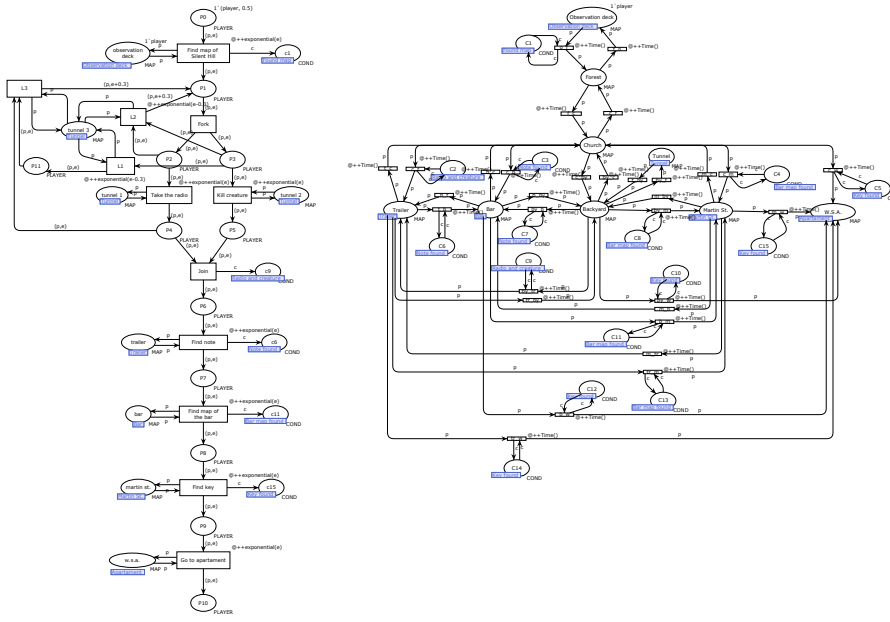


Figure 16. Full communication between the logical model and the topological model. The left side of the figure represents the activities of the first level of Silent Hill II. The right side represents the areas of the virtual world.

5 GAMEPLAY ANALYSIS

One of the advantages of using Petri nets is the existence of various analysis techniques. It is possible to analyze the state space of a Petri net by applying reachability analysis and to use some techniques to investigate some good properties of a Petri net. Simulation can also be used, and the emphasis in that kind of validation is to detect errors and to increase confidence in the correctness of a system [6].

In order to analyze the model of a game, two analysis techniques are used in this work. The first (qualitative analysis) is based on the analysis of the state space of the model. The second one (quantitative analysis) is based on simulation.

5.1 Soundness Verification

The idea behind a state space is to construct a graph which has a node for each reachable marking and an arc for each occurring binding element [5]. In other words, a state space represents all possible executions of the system under consideration and can prove that the system contains a certain formally specified property [6].

To construct the state space, the approach proposed by [19] (presented in Section 2.3) was used. This approach consists of creating an extended Petri net $\overline{PN}$ obtained by adding an extra transition $t*$ which connects the end places of the models to the start places of the same models. If $\overline{PN}$ is live and bounded, then the corresponding PN model can be considered Sound from the point of view of game modeling. The Soundness criterion will correspond then to the correctness of the model. Thus, for applying such a method of analysis, some modifications to the model presented in Figure 16 are needed.

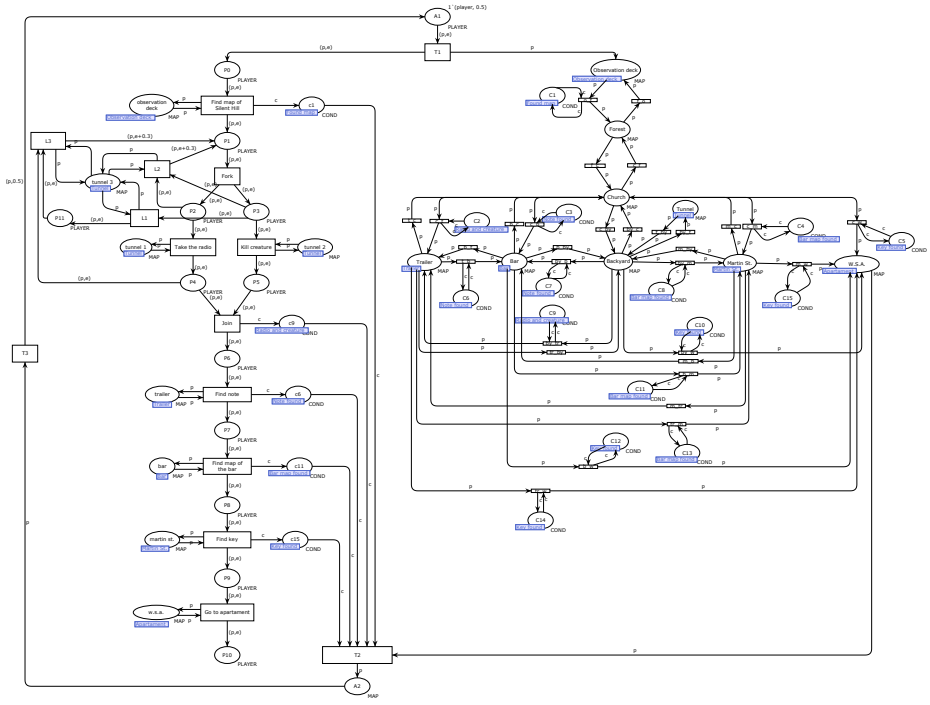


Figure 17. Global model for analysis

The modified model for analysis is presented in Figure 17. The time functions were omitted from the model since the notion of time is unnecessary for traditional state space exploration in the Petri net theory. Common *start* and *end* places, named *A1* and *A2* respectively, are created. The transition *T1* is a fork which produces one token in the start place of the level model (*P0*) and one token in the place *Observation deck* that represents the area of the game where the player will be at the beginning of the level. The transition *T2* is a join which has the purpose to consume the tokens that, in the Sound case, must be present at the end place of the logical model, at the place representing the area where the player will be at the end of the level, and in the marked condition places common to both models (used as places communication between the logical model and the topological model). The firing of transition *T3* will reinitiate the augmented model only if the non augmented model is Sound (all activity transitions of the logical model can be achieved by the player who can normally explore the various areas of the topological map). In practical terms, the augmented model will be reinitiated if there is no dead transition and no token duplication.

The qualitative analysis of the models is implemented using the state space analysis functionality of the CPN Tools. This functionality allows to record the results of the analysis in a report file. The first part of the report corresponds to the statistical information obtained after applying the state space analysis on the augmented model of Figure 17. The SCC Graph (Strongly Connected Components) indicates that there exists only one strongly connected component in the obtained reachability graph after applying state space analysis. The second part of the state space report contains information about the boudedness property. According to the boundedness report, the game model (logical model + topological map) is bounded. The last part of the report indicates the live transition instances. All the transitions of the augmented model are live.

According to the state space report, the augmented Petri net model is bounded and live. Thus, according to the theorem presented in Section 2.3, the non augmented model is Sound. From the point of view of the game, it means that all activities will be performed eventually and all areas of the topological map will be accessed by the player, what is as a matter of fact expected from a video game. The second analysis of the models is based on simulation.

5.2 Simulation

Simulation supports validation [15]. It can be used to explore a finite number of executions of the system under consideration. Thus, simulation is suitable for detecting errors and for obtaining increased confidence in the correctness of a system [6]. The CPN Tools simulator supports interactive and automatic simulation. An interactive simulation provides a way to investigate different scenarios in detail and check how the model works. In the automatic mode, simulation can be performed in play mode or fast forward mode. The end of a simulation is determined by simulation stop criteria and produces a simulation report.

In this paper, the automatic simulation is used in order to estimate the time a player will need to complete a level of a game. All the simulations presented are performed automatically 10 times. Therefore, a replication functionality of the CPN Tools, expressed by the inscription *CPN'Replications.nreplications 10*, is used.

First, the logical model, presented in Figure 9, is simulated alone (without considering the topological model). Figure 18 shows the simulation report. The minimum duration for performing all activities of the level corresponds to approximately 8.54 time units, and the maximum duration corresponds to approximately 23.41 time units. The mean time to performed all tasks of the level corresponds to approximately 13.76 time units.

```
Simulation no.: 1
Steps.........: 9
Model time....: 8.54950963137
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 2
Steps.........: 9
Model time....: 15.30329904
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 3
Steps.........: 9
Model time....: 9.77368666598
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 4
Steps.........: 15
Model time....: 17.0892393767
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 5
Steps.........: 17
Model time....: 23.4159232825
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 6
Steps.........: 9
Model time....: 13.9048727582
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 7
Steps.........: 15
Model time....: 8.68495576436
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 8
Steps.........: 13
Model time....: 15.4814450027
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 9
Steps.........: 9
Model time....: 15.6245852868
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds

Simulation no.: 10
Steps.........: 9
Model time....: 9.91201704388
Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds
```

Figure 18. Simulation report of the Timed Logical Model

Figure 19 illustrates the simulation report of the topological model alone (presented in Figure 13). The topological model represents all the areas of the first level of Silent Hill II. According to the simulation report, the minimum duration to cover all the areas of the map corresponds to approximately 1.17 time units. In contrast, the maximum duration to cover all the areas of the map corresponds to approximately 10.50. The mean duration corresponds to approximately 5.01 time units.

In a game, the player can only perform an activity if he is in the appropriate area. And some areas are accessed only after performing a specific activity. It is necessary then to keep both models together with the existing interaction represented by the communication mechanisms to estimate the global duration of the level in

```
Simulation no.: 1                          Simulation no.: 6
Steps.........: 7                          Steps.........: 18
Model time....: 3.10039547966              Model time....: 10.5031058175
Stop reason...: No more enabled transitions! Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds          Time to run simulation: 0 seconds

Simulation no.: 2                          Simulation no.: 7
Steps.........: 4                          Steps.........: 8
Model time....: 1.17036978075              Model time....: 4.20277828118
Stop reason...: No more enabled transitions! Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds          Time to run simulation: 0 seconds

Simulation no.: 3                          Simulation no.: 8
Steps.........: 11                         Steps.........: 4
Model time....: 6.05321861787              Model time....: 1.83511203897
Stop reason...: No more enabled transitions! Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds          Time to run simulation: 0 seconds

Simulation no.: 4                          Simulation no.: 9
Steps.........: 11                         Steps.........: 15
Model time....: 5.32683792618              Model time....: 8.49571707258
Stop reason...: No more enabled transitions! Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds          Time to run simulation: 0 seconds

Simulation no.: 5                          Simulation no.: 10
Steps.........: 7                          Steps.........: 12
Model time....: 4.17942236777              Model time....: 5.24148291281
Stop reason...: No more enabled transitions! Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds          Time to run simulation: 0 seconds
```

Figure 19. Simulation report of the Timed Topological Model

a more realistic way. The timed global model presented in Figure 16 is simulated. According to the replication report in Figure 20, the minimum duration of the level corresponds to approximately 59.13 time units, and the maximum duration corresponds to approximately 210.44. The mean duration of the level corresponds to approximately 87.16 time units.

When the models are simulated separately, the estimated duration refers only to the property of a specific model that expresses only one vision of the game (activities or map). This does not happen when considering the global model because one model influences the other. For example, when the topological model is simulated alone, all areas of the map can be accessed. This is not possible when the global model is simulated because the player needs to perform certain activities to obtain access to some areas. When the activity model is simulated alone all the activities can be performed almost immediately, which again is not possible when the global model is simulated. In fact, the player needs to be in a specific area of the map to perform a specific activity of the game and need a certain time to move from one area to another. In this way, both models need to be considered together by means of a communication mechanism in order to produce a realist simulation and obtain an accurate estimate time when considering the level of a video game.

```
Simulation no.: 1                    Simulation no.: 6
Steps.........: 124                  Steps.........: 368
Model time....: 74.8600106922        Model time....: 210.443008263
Stop reason...: No more enabled transitions!   Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds    Time to run simulation: 0 seconds

Simulation no.: 2                    Simulation no.: 7
Steps.........: 153                  Steps.........: 87
Model time....: 94.7624598182        Model time....: 59.6436906794
Stop reason...: No more enabled transitions!   Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds    Time to run simulation: 0 seconds

Simulation no.: 3                    Simulation no.: 8
Steps.........: 97                   Steps.........: 174
Model time....: 59.1337012618        Model time....: 105.075118921
Stop reason...: No more enabled transitions!   Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds    Time to run simulation: 0 seconds

Simulation no.: 4                    Simulation no.: 9
Steps.........: 117                  Steps.........: 110
Model time....: 70.8650821122        Model time....: 65.2153802654
Stop reason...: No more enabled transitions!   Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds    Time to run simulation: 0 seconds

Simulation no.: 5                    Simulation no.: 10
Steps.........: 114                  Steps.........: 109
Model time....: 71.4459040729        Model time....: 60.1723798904
Stop reason...: No more enabled transitions!   Stop reason...: No more enabled transitions!
Time to run simulation: 0 seconds    Time to run simulation: 0 seconds
```

Figure 20. Simulation report of the Global Model

6 CONCLUSION

This article presented the modeling and analysis of video games using the formalism of Petri nets. A video game is composed of activities and a virtual world. To represent the activities of a game the WorkFlow net was used. The areas of the virtual world (topological map) was represented by a state graph. A timed version of the model was presented too. In particular, random time functions were used to simulate the time a player needs to complete a level of a game. The fact of using a same formalism (Petri Nets) for both models (logical model + topological model) permitted to consider a kind of synchronous communication mechanism used to show the influence of one model over the other. The software CPN Tools was used to implement the approach.

Two forms of analysis were presented. A qualitative analysis based of a state space construction was presented to verify the Soundness of the model. The main purpose of the state space analysis is to show that all the activities of the level model can be executed eventually and that all the areas of the virtual world can be accessed by the player. A quantitative analysis was presented to calculate the game play duration. The simulation results show in particular the influence that a model has over the other, thus making it possible to produce the effective duration a player will need to complete a specific game level.

Comparing this approach with other works dealing with game modeling, its main advantage is that the use of the same formalism to specifying different views (logical + topological) of a game allows the use of the CPN Tools to implement a kind of qualitative and quantitative analysis of the gameplay. In this approach,

there is no need to translate the model to another formalism or deal with the problem of state explosion. In addition, such an approach has the advantage to verify the correctness of a video game still in the level design phase (before its implementation). In that way, the CPN Tools was an interesting option as it is capable of producing graphical models and provides efficient analysis functionalities.

As a future work proposal, it will be interesting also to investigate the behavior of multiplayer games and to model the interactions that exist between several players that collaborate to reach a common goal.

REFERENCES

[1] Ang, C. S.—Rao, G. S. V. R. K.: Designing Interactivity in Computer Games: A UML Approach. International Journal of Intelligent Games and Simulation, Vol. 3, 2004, No. 2, pp. 62–69.

[2] Araújo, M.—Roque, L.: Modeling Games with Petri Nets. Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory (DiGRA 2009), London, UK, 2009.

[3] de Oliveira, G. W.—Julia, S.—Passos, L. M. S.: Game Modeling Using Workflow Nets. 2011 IEEE International Conference on Systems, Man, and Cybernetics, 2011, pp. 838–843, doi: 10.1109/icsmc.2011.6083757.

[4] Gal, V.—Le Prado, C.—Natkin, S.—Vega, L.: Writing for Video Games. Proceedings Laval Virtual (IVRC), 2002.

[5] Jensen, K.: An Introduction to the Practical Use of Coloured Petri Nets. In: Reisig, W., Rozenberg, G. (Eds.): Lectures on Petri Nets II: Applications (ACPN 1996). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1492, 1998, pp. 237–292, doi: 10.1007/3-540-65307-4_50.

[6] Jensen, K.—Kristensen, L.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, Berlin, Heidelberg, 2009, doi: 10.1007/b95112.

[7] Konami: Silent Hill 2. Computer Game, Developed and Published by Konami, 2001.

[8] Milner, R.—Harper, R.—MacQueen, D.—Tofte, M.: The Definition of Standard ML (Revised Edition). The MIT Press, 1997, doi: 10.7551/mitpress/2319.001.0001.

[9] Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, Vol. 77, 1989, No. 4, pp. 541–580, doi: 10.1109/5.24143.

[10] Natkin, S.—Vega, L.—Grünvogel, S.: A New Methodology for Spatiotemporal Game Design. In: Mehdi, Q., Gough, N. (Eds.): Proceedings of 5th Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004), 2004, pp. 109–113.

[11] David, R.—Alla, H.: Discrete, Continuous, and Hybrid Petri Nets. Springer, 2010, doi: 10.1007/978-3-642-10669-9.

[12] Reuter, C.—Göbel, S.—Steinmetz, R.: Detecting Structural Errors in Scene-Based Multiplayer Games Using Automatically Generated Petri Nets. Proceedings of

the 10th International Conference on the Foundations of Digital Games (FDG 2015), Pacific Grove, USA, 2015.

[13] Reyno, E. M.—Cubel, J. A. C.: Automatic Prototyping in Model-Driven Game Development. Computers in Entertainment, Vol. 7, 2009, No. 2, Art. No. 29, doi: 10.1145/1541895.1541909.

[14] Rucker, R. V. B.: Software Engineering and Computer Games. Pearson Education, 2003.

[15] van der Aalst, W.: Timed Coloured Petri Nets and Their Application to Logistics. International Symposium on Physical Design, 1992.

[16] van Der Aalst, W.—van Hee, K. M.: Workflow Management: Models, Methods, and Systems. MIT Press, 2004.

[17] van der Aalst, W. M.: Structural Characterizations of Sound Workflow Nets. Computing Science Reports, Vol. 96, 1996, No. 23, pp. 18–22.

[18] van der Aalst, W. M.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (Eds.): Application and Theory of Petri Nets 1997 (ICATPN 1997). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1248, 1997, pp. 407–426, doi: 10.1007/3-540-63139-9_48.

[19] van der Aalst, W. M.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers, Vol. 8, 1998, No. 1, pp. 21–66, doi: 10.1142/s0218126698000043.

[20] van der Aalst, W. M.—Stahl, C.: Modeling Business Processes: A Petri Net-Oriented Approach. The MIT Press, 2011, doi: 10.7551/mitpress/8811.001.0001.

[21] van der Aalst, W. M.—ter Hofstede, A. H.: Verification of Workflow Task Structures: A Petri-Net-Based Approach. Information Systems, Vol. 25, 2000, No. 1, pp. 43–69, doi: 10.1016/s0306-4379(00)00008-9.

Franciny M. Barreto works at the Federal University of Jataí as Assistant Professor. She received her Bachelor degree in computer science in 2012 from the Federal University of Goiás (Brazil), her Master degree in computer science in 2015, and her Ph.D. in 2020, both of them from the Federal University of Uberlândia (Brazil).

Stéphane Julia received his Diploma in electrical and automatic control engineering in 1992, his Master degree in automatic and industrial computing in 1993 and his Ph.D. in industrial computing in 1997, all of them from the Paul Sabatier University of Toulouse (France). He is currently Full Professor of computer science at the Federal University of Uberlândia (Brazil). His research interests include the use of Petri nets in software engineering and the modeling and analysis of workflow management systems.