

Using Petri Nets for Analysis of Navigation Paths in Constrained Graphs – Application to Roguelike Games

Luis Gomes^{1,2,3,*}, José Ribeiro-Gomes⁴ and João-Paulo Barros^{2,3,5}

¹NOVA University Lisbon, School of Science and Technology, Portugal

²Center of Technology and Systems (CTS), UNINOVA, Portugal

³Intelligent Systems Associate Laboratory (LASI), Portugal

⁴Instituto Superior Técnico, Portugal

⁵Polytechnic Institute of Beja, Beja, Portugal

Abstract

In many areas of application, graphs have been used to support path planning addressing navigation challenges in dynamic environments. Applications target very diverse areas, ranging from manufacturing and robotics to video games, including georeferencing applications in our daily lives considering real-time traffic conditions. This paper focuses on applications where navigation through a predefined map consisting of distinct areas (rooms or nodes) connected by pathways needs to be defined. Roguelike games, a genre defined by procedurally generated environments and random map creation, are used to validate the proposal. Navigation is constrained by resource availability and specific conditions that allow or block movement between adjacent areas. Players can collect specific resources in visited areas, which can be used to allow/unlock passages. The main goals include determining whether, given a specific randomly generated map/graph, along with all associated constraints and resources, the game is inviable (when it is not possible to reach the final area of the game), non-viable (when depending on the player's options, it is possible or not to reach the final goal), or viable (when the ultimate goal area can always be reached regardless of user's choices over time). This paper proposes a strategy for translating the map and associated graph into a Petri net model. Then, the viability level of the game can be determined by constructing and analyzing the associated reachability graph from the Petri net model. A set of examples are presented illustrating typical situations.

Keywords

Procedurally generated games, Graph Analysis, Reachability graphs, Petri nets

1. Introduction

In several areas of application, selecting a strategy for navigation is of paramount importance. In this sense, path planning is a crucial consideration in navigation. A significant number of strategies (probably the most used ones) rely on a characterization of the environment by a set of nodes and known costs associated with traversing from one node to another. This directly supports the construction of a graph, facilitating the determination of the optimal path from an initial node to a destination node.

Graph-based algorithms, being easily processable [1], are extensively employed in addressing path-planning challenges. In these algorithms, the space is characterized as a collection of cells, and each connection between cells is assigned a weight representing its associated cost. These costs may account for various constraints, such as distances, energy, time, resources, or other types of resources, depending on the specific area of application.

Dijkstra introduced a pioneering algorithm in 1959 [2], which remains highly relevant for determining a specific path with minimized associated costs. Various adaptations of Dijkstra's algorithm have been proposed. Best-First-Search [3] associates cost with the distance to the goal. A* [4] is an extension that combines Dijkstra's algorithm and Best-First-Search, evaluating paths based on both cost and the distance of the node to the goal. Dynamic A* [5] extends A* to dynamic environments by accommodating changing costs over time.

PNSE'24, International Workshop on Petri Nets and Software Engineering, 2024

*Corresponding author.

✉ lugo@fct.unl.pt (L. Gomes); josepgomes@tecnico.ulisboa.pt (J. Ribeiro-Gomes); joao.barros@ipbeja.pt (J. Barros)

🆔 0000-0003-4299-8270 (L. Gomes); 0000-0002-0097-9883 (J. Barros)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Several alternative approaches have been proposed to tackle path planning in different application areas. Some still consider static scenarios, while others cope with dynamic scenarios (such as geo-referenced traffic applications). Some are bio-inspired, while others consider some physics-related features.

In the first group considering bio-inspired alternatives, particular reference is given to the ant colony optimization algorithm, which mimics the behavior of ant colonies [6, 5]. This algorithm seeks to find the shortest path from a nest to a food source and back by emulating ant behavior, which involves the release of pheromones. These pheromones are detected by other ants, guiding them towards the established path.

In the second group emphasizing physics-inspired approaches, one can find the artificial potential fields method introduced in 1985 [7]. This model utilizes potential fields, where the selected path for the movement is influenced by two forces: attraction towards the goal and repulsion from obstacles. The combined effects of these forces determine the direction to follow.

As in Dijkstra's algorithm and other referred algorithms, the focus is on the shortest path between two specific nodes; one node in the path is visited once at most. In applications where additional constraints (other than costs) need to be considered, those approaches would need severe changes (if not wholly inadequate). This is the case when the movement along the path needs the possession of a specific resource, which in turn can be obtained in some node not belonging to the shortest path to the destination. For instance, while at node A, it may be necessary to visit some other node to bring a specific resource, which will be used afterward to unlock some movement between nodes and return to node A to proceed toward the goal. In this sense, immediate applicability of the referred algorithms is not possible.

This is the case for the application area of roguelike games, where the game's map is randomly generated and can be represented using a graph. Movements between rooms in the map could be constrained by some doors, which can be unlocked using specific resources (keys, weapons, or others); those resources can be obtained in some other room (node). Fig. 1 shows two simple maps suffering from an ill-formed condition.

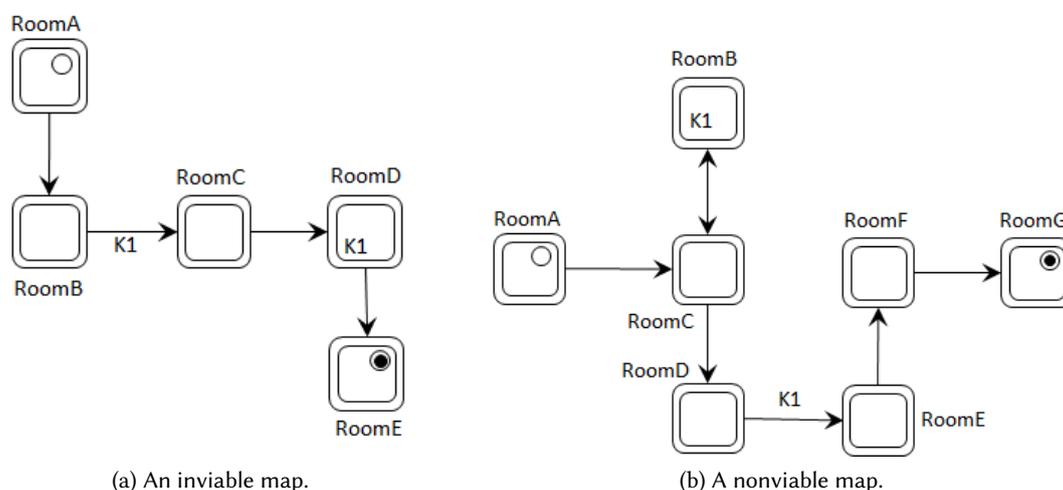


Figure 1: Examples of ill-formed maps.

In Fig. 1(a), to go from RoomA to RoomE, it is necessary to move along RoomB to RoomC. However, for this movement, the player needs to have a specific resource (key K1, in this case), which in turn can only be granted when reaching RoomD. In this case, the game is inviable, as it cannot be concluded. Another annoying situation is illustrated in Fig. 1(b), where the movement from RoomA to RoomG needs the possession of key K1 (to move from RoomD to RoomE). However, if the player chooses to move away from RoomC without visiting RoomB, they will be blocked in RoomD. In this case, the final room might not be reached depending on the player's options (actions over time). This is considered as a nonviable map. Only maps allowing the player to reach the final room regardless of their options are

acceptable (as in roguelike games, resetting the game halfway would lead to an entirely new generated map/graph, resulting in loss of player progression).

This paper analyzes potential navigation paths within a randomly generated map, where navigation is constrained by particular conditions associated with specific movements between rooms. This scenario is particularly relevant in roguelike games, which are procedurally generated, and where a significant challenge lies in ensuring that, following the random generation of maps and constraints, players can always reach the ultimate goal without encountering deadlock situations.

The proposed approach involves translating the constrained map into a Petri net model, a strategy previously suggested in other works [8], and complementing and extending the work presented in [9]. By constructing the Petri net model associated with the map/graph of the game, the associated reachability graph is generated, and its analysis answers the questions mentioned earlier. To the best of our knowledge, this is the first work tackling the classification of possible navigation paths between two nodes in graphs, which in turn were generated as representing constrained maps.

The paper is structured as follows. In Section 1, the motivation for the work identifying the challenge to be addressed is presented, while Section 2 briefly describes different approaches to building a roguelike game map/graph, as well as introducing a simple map/graph random generator used for validation of the approach afterward. Section 3 presents the proposed approach to perform the analysis of the map characteristics, while Section 4 is devoted to presenting the steps to obtain the Petri net model from the map/graph, and Section 5 focus on the analysis strategies. Section 6 is dedicated to the validation using a set of examples, and Section 7 concludes.

2. Building the map and the graph of the roguelike game

The proposed approach relies on a graph representation of the map, so it can be translated to a Petri Net representation and benefit from associated analysis techniques.

Though relying on a graph representation may seem like a too-limiting requirement, in this section, we present a non-exhaustive list of map generation techniques used in roguelikes and show that this is not the case and how they can be mapped to a graph representation. In fact, some map generation techniques already rely on defining a map graph as a starting point, and others may be translated to a graph representation afterwards. Consequently, we aim to show that the proposed approach is suitable for diverse implementations of map generation.

We structure the section into techniques that rely on defining a map graph beforehand and then translating it into a map, and techniques that define a map straight away but may then be represented as graphs. All shown techniques have been used in games, and none consider including conditions to constrain movements between nodes/rooms (such as the need to unlock doors between rooms).

2.1. Graph First

The following approaches start by defining a graph structure that represents the map only after populating the actual playable map.

One approach is Binary Space Partitioning [10, 11], which recursively divides a space into smaller spaces. The random subdivisions can be interpreted as rooms with different layouts, sizes, and connections between them. They inherently have a hierarchical, tree-like structure.

Another technique consists of defining a Voronoi diagram using a set of points to subdivide the map [11]. The regions inside the diagram might correspond to different areas of interest, and the edges may correspond to connections or transitions.

Lastly, some games have a pre-defined, manually created graph specifying the map's layout. This approach usually relies on a set of pre-made rooms that are randomly selected to instantiate the graph. One such game is *Enter the Gungeon*¹.

¹<https://enterthegungeon.com>

2.2. Map First

Contrary to the approaches above, the following approaches focus on generating the map straight away. Some approaches may be translated to a graph afterward, while others may be more difficult.

A common approach is to use cellular automata [11]. It considers that the map is a grid of cells, each belonging to a specific group or state, such as room, door, wall, water, . . . Starting from an initial cell, it iteratively considers the neighboring cells in order to populate the world in a way such that a series of rules are followed.

Perlin noise is also prevalent for map generation. It is a gradient noise function that produces smooth, continuous patterns. Randomness is introduced through noise generation, creating natural-looking variations in terrain. An example of a very popular game that uses Perlin noise to define its world is *Minecraft*².

Random Walk and Drunkard's Walk [11] are both approaches that involve placing tiles in the world by iteratively moving through the map in a random direction at each step. These techniques create winding paths, favored for cave systems, paths, roads, or rivers.

Yet another approach is Wave Function Collapse³, which uses an input pattern (such as an image or small region of what the world such look like) and a set of constraints. One game that uses this technique is *Caves of Qud*⁴.

2.3. Map/graph random generator used for validation

In order to validate the proposed approach on randomly generated maps/graphs associated with roguelike games, a simple map/graph generator was used. It is used in the examples in the validation section. As stated, there is no loss of generality, as all implementations that produce a graph map are compatible with the proposed approach.

The adopted technique for the implemented simple map/graph generator was inspired by an online tutorial⁵, and can be seen as an extension of cellular automata. The code of the developed map/graph random generator is available at <https://github.com/miguelmtsvitoria/RandomMap>, which is a result of [12].

In this simple map/graph generator, a grid map was considered, where cells may contain a room. These rooms may be connected to adjoining rooms by means of doors placed North, East, South, and/or West, so that no room may have a door that connects to nothing. Furthermore, each room may have more than one door, provided there is only one door in each direction and that all rooms are connected.

The implemented map generator starts by defining a central room with four doors, one on each side. Afterward, it iteratively checks each available door and adds a compatible room. For example, for the East door, we must add a room that has a West door and may have additional doors. Each room added may contain new doors, which allow the map to develop and expand from that central node.

One key concept in this approach is that we may define a passage between two rooms as bidirectional or unidirectional or as requiring a key (or other constraint) to open a door.

The size of the map, the overall number of connections per room, restrictions on doors, and so on may be controlled to change the difficulty of the game.

2.4. Representing the graph

The representation of the map layout randomly generated by the application described in the previous sub-section is a simple XML file storing the list of rooms (dungeons) and the list of transitions between rooms, as presented in Table 1, which stores the information associated with the (very simple) map of Fig. 2. In this example, one key is available (at RoomB), which is needed to unlock the transition from RoomD to RoomE (the final room).

²<https://www.minecraft.net>

³<https://github.com/mxgmn/WaveFunctionCollapse>

⁴<https://www.cavesofqud.com>

⁵https://www.youtube.com/playlist?list=PLB1b_auVtBwA-qr2-WnWX0LjZXkqKu5Aj

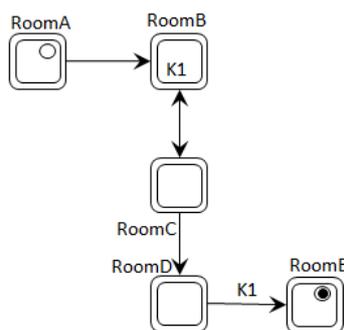


Figure 2: A simple map.

```

<map>
  <layout>
    <dungeons>
      <room roomName="RoomA" state="Initial" key="" />
      <room roomName="RoomB" state="Intermedium" key="K1" />
      <room roomName="RoomC" state="Intermedium" key="" />
      <room roomName="RoomD" state="Intermedium" key="" />
      <room roomName="RoomE" state="Goal" key="" />
    </dungeons>
    <transitions>
      <transition source="RoomA" destination="RoomB"
        bidirectional="False" keyNeeded="" />
      <transition source="RoomB" destination="RoomC"
        bidirectional="True" keyNeeded="" />
      <transition source="RoomC" destination="RoomD"
        bidirectional="False" keyNeeded="" />
      <transition source="RoomD" destination="RoomE"
        bidirectional="False" keyNeeded="K1" />
    </transitions>
  </layout>
</map>

```

Table 1

XML file associated with map of Fig. 2

3. Proposed approach for analysis of the game's map/graph

The proposed approach to support the analysis of the map/graph randomly generated for a roguelike game starts with the translation into a Petri net model.

The player (there is only one player) starts the game in one specific room (the initial room), and the goal is to reach the final room. Some rooms may contain some items (called keys in this paper), which can be collected by the player when visiting the room. These items can unlock some movements between rooms, which means that some passages between rooms can be constrained by the possession of a key.

Petri nets modeling was selected due to its support for the representation of resources (the keys, in this case) and further evolution of movements between rooms constrained by those resources in a very natural way (benefiting from local evaluation when executing the Petri net model).

Considering the type of maps/graphs to analyze, the state space associated with the Petri net model is finite. This means that building the reachability graph is a viable solution to fully analyze the map/graph's properties.

Fig. 3 presents the proposed flow to allow identification of the viability level of the generated map/graph, considering the following possible classifications:

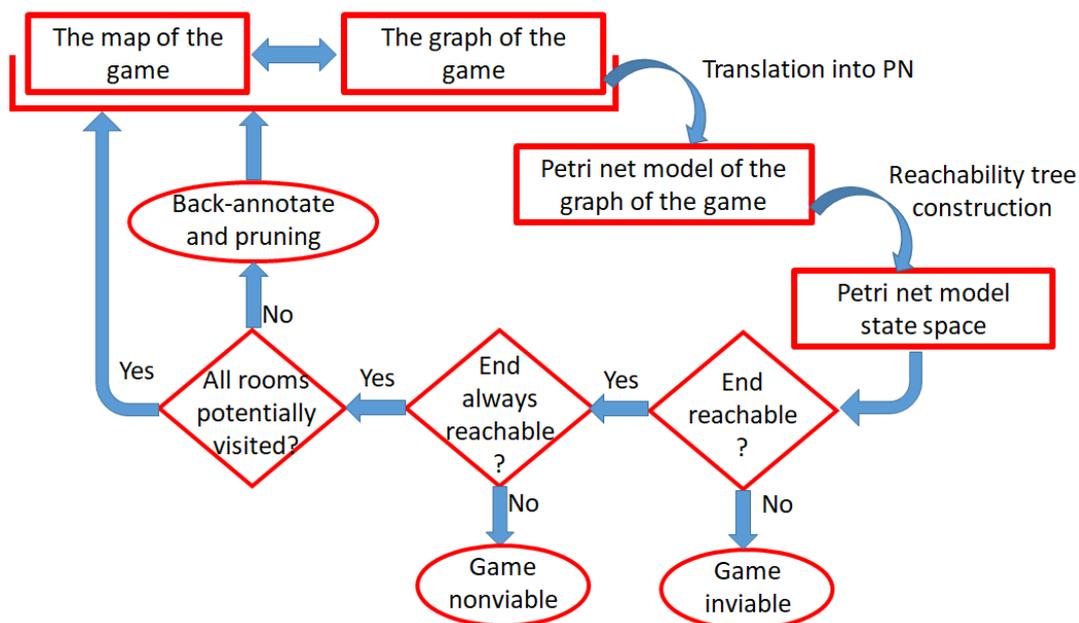


Figure 3: The proposed flow for analysis of the game's map/graph.

- inviable game: where the final room cannot be reached.
- nonviable game: where the final room can be reached or not depending on the player's options (for instance, if the player chooses to follow a one-way direction dead-end, they will be stuck there, and the final room will not be reached).
- viable game with inaccessible rooms: where the final room is always reachable regardless of the player's options, but there are rooms that cannot be visited (which represent resource consumption without purpose); in this case, pruning those rooms from the map is recommended.
- viable game: where the final room is always reachable regardless of the player's options, and all rooms can be visited.

4. Generating the Petri net model

The first step of the proposed approach is the generation of the Petri net model, which describes the possible player movements through the map. Two steps are considered to obtain the Petri net model:

- Applying a set of translation rules to each of the map/graph characteristics into Petri nets constructs. The rules are applied separately to each map/graph characteristic, leading to a Petri net model composed of several disjoint sub-models.
- Applying net addition operation of the disjoint sub-models to obtain the global Petri net model.

The Place-Transition Petri net class is considered for the models to be produced.

The translation rules are divided into two groups:

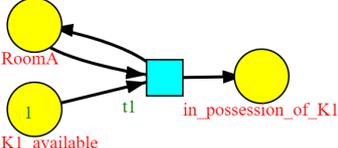
- One group focused on translating the rooms (the graph nodes). Table 2 shows (four) rules applied to:
 - the initial room.
 - the final room (the goal of the game).
 - an intermediary room.
 - a room holding some resource (for instance a key, or other resource used in the game)
- Another group focused on translating the movements between rooms/nodes. Table 3 shows five rules applied to:

- the transition between two rooms (unidirectional).
- the transition between two rooms (bidirectional).
- the transition between two rooms (unidirectional) constrained by the possession of a resource (e.g., a key).
- the transition between two rooms (bidirectional) constrained by the possession of a resource (e.g., a key).
- the asymmetric transition between two rooms (bidirectional) constrained by the possession of a resource in one direction and unconstrained in the other.

More translation rules could be defined to model additional features of map/graph transitions. Examples may include other types of asymmetric transitions or consider a cost on a specific resource instead of the possession of a key (leading to the consumption of parts of the resource).

Table 2

From map components to Petri net sub-model: translation rules for rooms; (a) Initial Room; (b) Final Room; (c) A Room, (d) A Room with a resource/reward K1

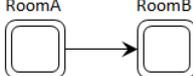
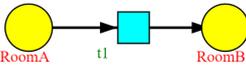
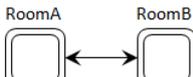
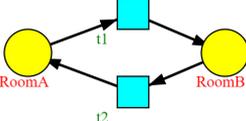
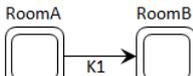
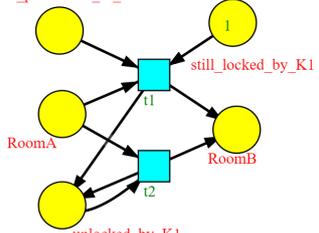
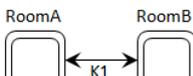
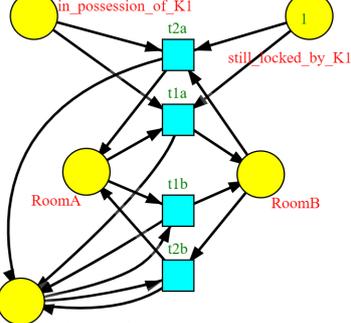
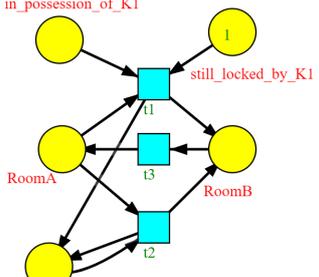
<i>Description</i>	Map component	Petri net sub-model
(a)	Initial_Room 	 InitialRoom
(b)	Final_Room 	 FinalRoom
(c)	A_Room 	 ARoom
(d)	Room_with resource_K1 	

After applying the set of translation rules to all components of the map/graph, a set of disjoint sub-models is obtained. The generation of the final Petri net modeling of the whole map/graph can be summarized in the following steps, illustrated in Figure 4:

- Generate a sub-model for each room considering the rules of Table 2. While places will keep the names, the names of the transitions needed in the sub-models will be unique and sequentially generated (using a sequence number, such as t1, t2, ...). These steps are numbered from 1 to 5 in Figure 4.
- Generate a sub-model for each transition considering the rules of Table 3. While the places associated with rooms of the instance of the sub-model are updated with the names of specific rooms, transitions will be generated, keeping the unique names strategy (continuing the sequence numbering). These steps are numbered 6 to 9 in Figure 4.
- the last step in the process of generating the final Petri net model associated with the randomly generated graph/map (step 10 in Figure 4) uses the net addition operation [13], where all places with equal names will be fused into one with the same name holding a marking that is the sum of all tokens in the fusing set (in this sense, only place fusion is used in the referred net addition operation).

Table 3

From map components to Petri net sub-model: translation rules for transitions; (a) move from room A to room B (unconditionally); (b) move from room A to room B and from room B to room A (unconditionally); (c) move from room A to room B constrained by K1; (d) move from room A to room B and from room B to room A constrained by K1; (e) move from room A to room B constrained by K1 and from room B to room A unconditionally

Description	Map component	Petri net sub-model
(a)		
(b)		
(c)		
(d)		
(e)		

5. Analysis of the map

The overall strategy to determine the viability level of the randomly generated map/graph can be described using Algorithm 1. As previously mentioned, the generation of map/graph examples for validating the approach is performed using the application described in sub-section 2.4, which generates the XML file describing the map, as well as the associated PNML file associated with the generated Petri net model after applying the translation rules presented in Section 4. The IOPT-Tools web-based framework [14] is used to read the PNML file and to support the remaining steps to classify the viability level of the generated map/graph as described through the Algorithm 1. The IOPT-Tools framework,

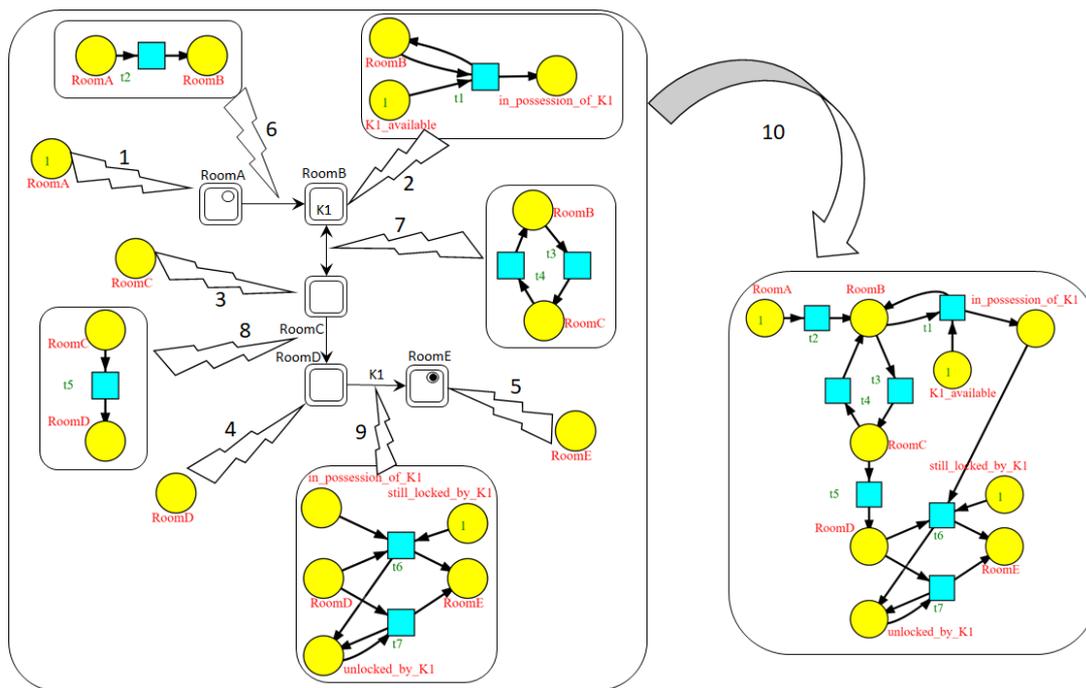


Figure 4: Building the Petri net model associated with a map/graph.

available at <http://gres.uninova.pt/IOPT-Tools/>, uses one specific class of Petri nets, the Input-Output Place-Transition nets (IOPT-nets) [18], which was selected for convenience of the authors, but other tools could also provide similar support for implementation of Algorithm 1 (Note: the Petri net models presented were edited using the IOPT-Tools framework, so instead of using a black dot to represent a token, the number "1" is used instead).

The proposed strategy heavily relies on the construction of the reachability graph, which is a directed graph that represents all reachable states, considering a specific initial marking of the Petri net model. The nodes of this graph are all the reachable markings (global marking states), and the arcs connecting those nodes have an annotation referring to the set of transitions that produced that change in state.

After importing the PNML file to the IOPT-Tools framework, the associated reachability graph containing all reachable markings is obtained (using the "Generate State Space" tool option [16]) (line 2 of Algorithm 1). After that, the IOPT-Tools query system [17] is used to obtain the remaining answers. More specifically, a set of queries is formulated (using the "Query Editor" tool option), and associated answers are obtained (using the "Query Results" tool option).

In particular, the following queries are presented, and the following answers are obtained:

- the first query is $RoomXX = 1$, where $RoomXX$ is the final room for the map/graph under analysis; the result will provide a list with the numbers of the states of the reachability graph that contains $RoomXX$ marked (line 3 of Algorithm 1, providing the *SuccessMarkings* set). If the result is the empty set, then the map/graph is **inviabile** (if statement in line 4 of Algorithm 1).
- a second set of queries has the form $REACH(GlobalMarking)$, where $GlobalMarking$ is the number of each of the reachability graph states obtained as a result of the previous query; this corresponds to the loop in line 11 of Algorithm 1; the result will provide a list with the numbers of the states of the reachability graph that are in a path leading to $GlobalMarking$ (line 12 of Algorithm 1).

After knowing the list of reachability graph states that have a path to a global marking of success (where the final room is marked), comparing with the total number of reachability graph states (line 14 of Algorithm 1) it is possible to conclude if the map/graph is **nonviable** (line 16 of Algorithm 1) or **viable** (else in line 17 of Algorithm 1).

Finally, when the map/graph is viable and the reachability graph is obtained (using the "Generate State Space" tool option), the information of the maximum number of tokens in all places can be known,

Algorithm 1 Checking the viability of the map and the reachability of the game's final objective**Require:** $MarkedPetriNet = (P, T, F, M = P \rightarrow \mathbb{N})$ **Require:** $F \subseteq P \times T \cup T \times P$ **Require:** $Rooms \subseteq P$ **Require:** $FinalRoomPlace \in Rooms$

```

1: function CHECKMAP( $MarkedPetriNet, Rooms, FinalRoomPlace$ )
2:    $ReachabilityGraph \leftarrow BUILDASSOCIATEDREACHABILITYGRAPH(MarkedPetriNet)$ 
3:    $SuccessMarkings \leftarrow \{m \in NetMarkings(ReachabilityGraph) :$ 
       $m(FinalRoomPlace) > 0\}$ 
4:   if  $SuccessMarkings = \emptyset$  then
5:      $\triangleright$  The game is inviable (impossible to reach FinalRoom)  $\triangleleft$ 
6:     return  $GAME\_IS\_INVIABLE$ 
7:   else
8:      $\triangleright$  game final room (objective) is reachable  $\triangleleft$ 
9:      $\triangleright$  success markings are net markings where place modeling the final room is marked  $\triangleleft$ 
10:     $MarkingsToTheGoal \leftarrow \emptyset$ 
11:    for all  $m \in SuccessMarkings$  do
12:       $MarkingsToTheGoal \leftarrow MarkingsToTheGoal \cup$ 
       $NodesInPath(InitialMarking(ReachabilityGraph), m)$ 
13:    end for
14:    if  $|MarkingsToTheGoal| < |Markings(ReachabilityGraph)|$  then
15:       $\triangleright$  The game is nonviable (concluding the game depends on player options)  $\triangleleft$ 
16:      return  $GAME\_IS\_NONVIABLE$ 
17:    else
18:       $Visited \leftarrow \emptyset$ 
19:      for all  $placeRoom \in Rooms$  do
20:        if  $\exists m \in MarkingsToTheGoal : m(placeRoom) > 0$  then
21:           $Visited \leftarrow Visited \cup placeRoom$ 
22:        end if
23:      end for
24:      if  $|Visited| = |Rooms|$  then
25:         $\triangleright$  FinalRoom can always be reached  $\triangleleft$ 
26:        return  $GAME\_IS\_VIABLE\_AND\_ALL\_ROOMS\_VISITED$ 
27:      else
28:         $\triangleright$  The game is viable (FinalRoom can always be reached), but at least one room is not
       $reachable, so the map/graph should be pruned$   $\triangleleft$ 
29:        return  $GAME\_IS\_VIABLE\_NOT\_ALL\_ROOMS\_VISITED$ 
30:      end if
31:    end if
32:  end if
33: end function

```

allowing a decision based on all rooms having been visited (if statement in line 24) in the map/graph. Either all were visited (viable map with all rooms visited, line 26 of Algorithm 1) or only some (viable map with some rooms not reachable, line 29 of Algorithm 1).

Coming back to the simple map/graph of Figure 2, the associated reachability graph is presented in Figure 5 (using a tree representation, where some leaves denoted by squares are in fact representing loops returning to some already existing state), where two deadlock states are presented colored red: the state 9, associated with goal achievement (having final marking at places *RoomE* and *unlocked_by_K1*),

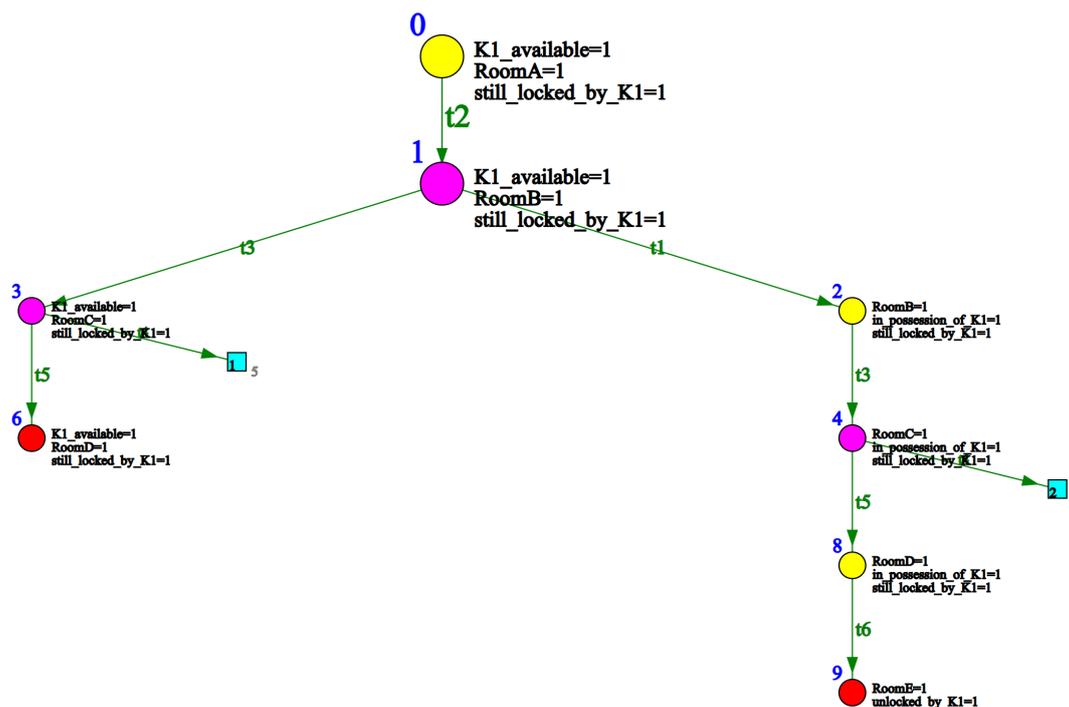


Figure 5: State space (reachability graph) associated with the Petri net generated from the map/graph of Figure 2.

and state 6, associated with a situation where final goal is not reachable (due to the player did not take the K1 key when visiting RoomB and moved afterward to Room D, where it is not possible to return to Room B).

Using the IOPT-Tools framework and the query "RoomE = 1", we got the answer "State 9". Afterward, for the query "REACH(9)", the answers were "States 0, 1, 2, 3, 4, 8, 9", which means that it is possible to reach State 9 from 7 reachable states. For the query "NOT REACH(9)", the answer was "State 6", which means that from State 6, the final goal is not reachable (states 5 and 7 are dummy states, as they only redirect to states 1 and 2). The cardinality of the set *MarkingsToTheGoal* is 7, while the cardinality of the set *Markings(ReachabilityGraph)* is 8 (important to note that even with eight states in the reachability graph, IOPT-Tools can associate higher numbering to reachable states due to dummy states (enclosed in squares), which represent pointers to reachable states). In this sense, the simple map of Figure 2 is nonviable (lines 14-16 of Algorithm 1).

6. Validation

In this section, small examples of maps/graphs randomly generated by the application described before are used to validate the correctness of the proposed approach, considering four typical situations:

- a viable map
- a viable map where some rooms are not reachable
- a nonviable map
- an inviable map

For the four analyzed situations, a figure will be used to present the map randomly generated by the application described before and the Petri net graph obtained using the IOPT-Tools framework, which in turn benefits from the usage of the open-source graph visualization software Graphviz [15].

6.1. Viable map

In this example, the map presented in Fig. 6(a) was generated containing 14 rooms and 3 keys, from which the Petri net model of Fig. 6(b) was generated, leading to a reachability graph composed by 288 states.

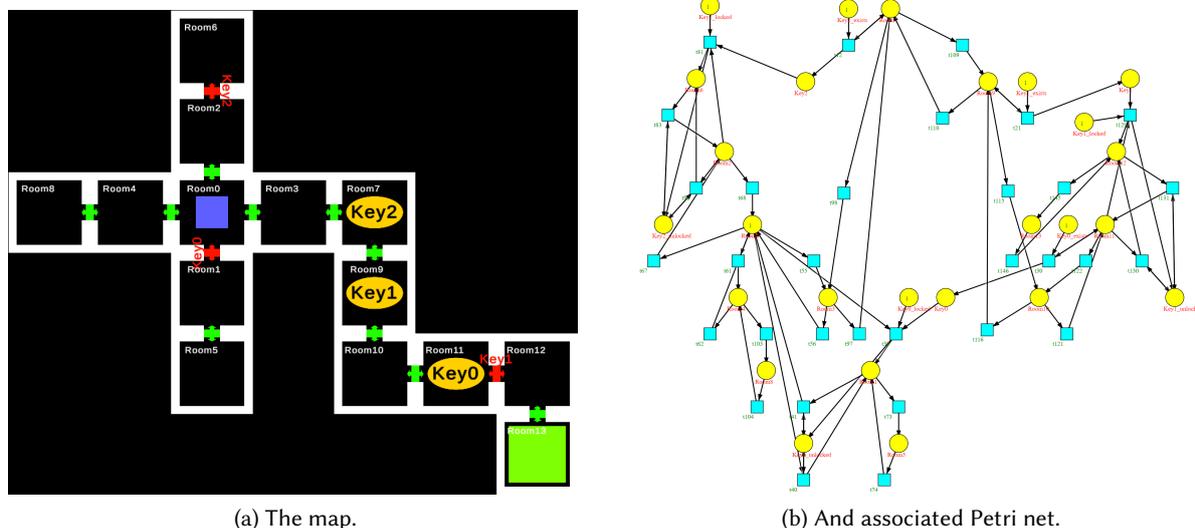


Figure 6: A viable map.

The query ($Room13 = 1$) received an answer identifying 9 states of the reachability graph where the place Room13 is marked: states 71, 82, 91, 109, 332, 375, 525, 531, and 571. Note: as previously referred, the numbering of the states in the reachability graph obtained from the IOPT-Tools accommodates the numbering of the reachable states as well as the numbering of dummy states, which only point to actual reachable states (that is the reason why, in this example, the reachability graph contains 288 reachable states, and it is possible to find states with a number greater than that).

From the 9 queries $REACH(stateX)$, where $stateX$ are the 9 states referred above, the answers were 29, 58, 58, 116, 90, 180, 93, 186, 288, accordingly (for example, for the query $REACH(71)$, 29 reachable states were identified). From this it is possible to conclude that all 288 states of the reachability graph have a path leading to (at least) one of the success states.

Finally, from the state space generation report, it is possible to conclude that all rooms are reachable. So, the conclusion is that the map is viable.

6.2. Viable map with rooms not reachable

In this example, the map presented in Fig. 7(a) was generated containing 15 rooms and 2 keys, from which the Petri net model of Fig. 7(b) was generated, leading to a reachability graph composed by 30 states.

The query ($Room14 = 1$) received an answer identifying 4 states of the reachability graph where the place Room14 is marked: states 21, 25, 46, and 47.

From the queries, $REACH(21)$, $REACH(25)$, $REACH(46)$, and $REACH(47)$, the answers identifying the number of states that can reach the referred state were 12, 13, 21, and 22, accordingly. Considering the answers provided, it is possible to conclude that all 30 states of the reachability graph have a path leading to (at least) one of the success states.

Finally, from the state space generation report, it is possible to conclude that some rooms are not reachable, namely Room2, Room6, and Room10 at the top of the figure. More specifically, to move from Room0 to these rooms, possession of Key1 is needed, but Key1 is available at the bottom of the map after moving to Room9, from where it is impossible to return.

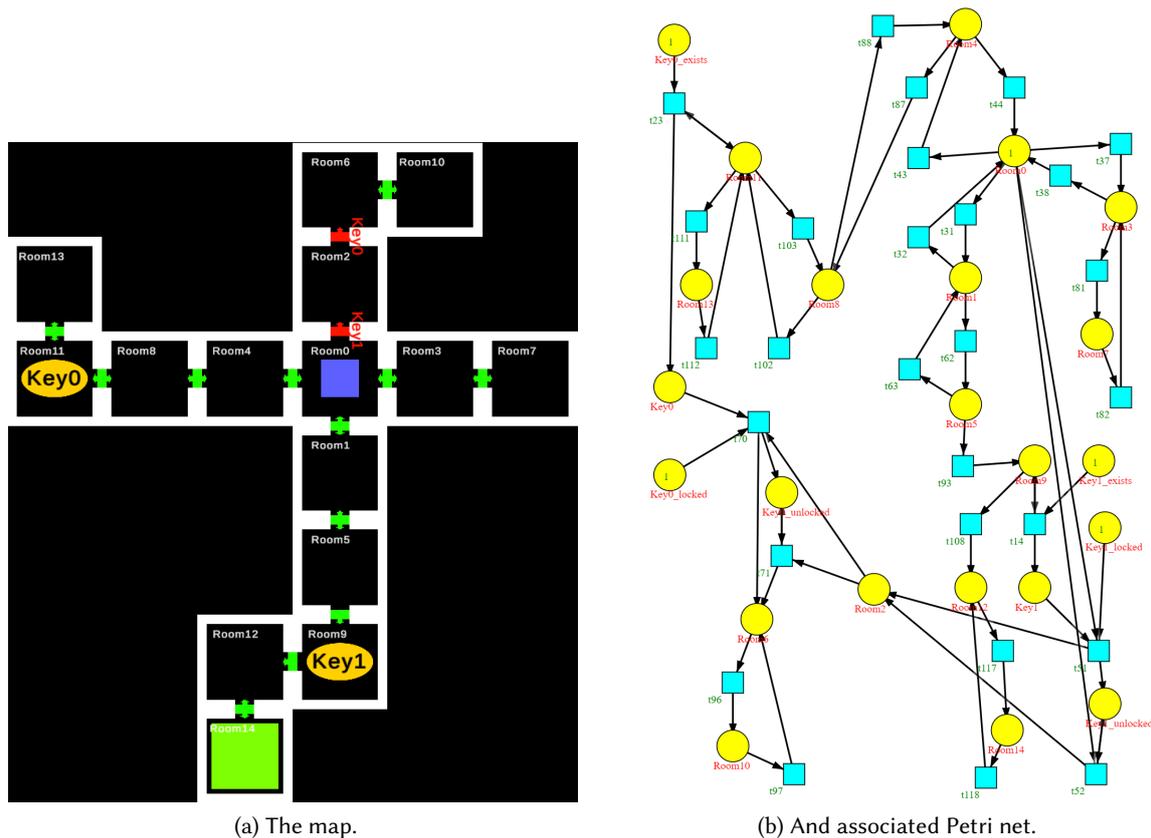


Figure 7: A viable map with rooms not reachable.

So, the conclusion is that the map is viable, but contains non reachable rooms. As an additional step before providing the map for the game to proceed, it is recommended to prune the map from Room2, Room6, and Room10.

6.3. Nonviable map

In this example, the map presented in Fig. 8(a) was generated containing 17 rooms and 2 keys, from which the Petri net model of Fig. 8(b) was generated, leading to a reachability graph composed by 129 states.

The query (Room16 = 1) received an answer identifying 3 states of the reachability graph where the place Room16 is marked: states 131, 139, and 181.

From the queries, $REACH(131)$, $REACH(139)$, and $REACH(181)$, the answers identifying the number of states that can reach the referred state were 37, 74, and 120, respectively, from where it is possible to conclude that only 120 states of the reachability graph have a path leading to one of the success states.

So, the conclusion is that the map is nonviable: if the player moves to Room15 on the top, they will not be able to return as the transition between Room13 and Room15 is unidirectional.

6.4. Inviable map

In this example, the map presented in Fig. 9(a) was generated containing 21 rooms and 3 keys, from which the Petri net model of Fig. 9(b) was generated leading to a reachability graph composed by 18 states.

The query (Room20 = 1) received an answer confirming that Room20 is unreachable.

So, the conclusion is that the map is inviable.

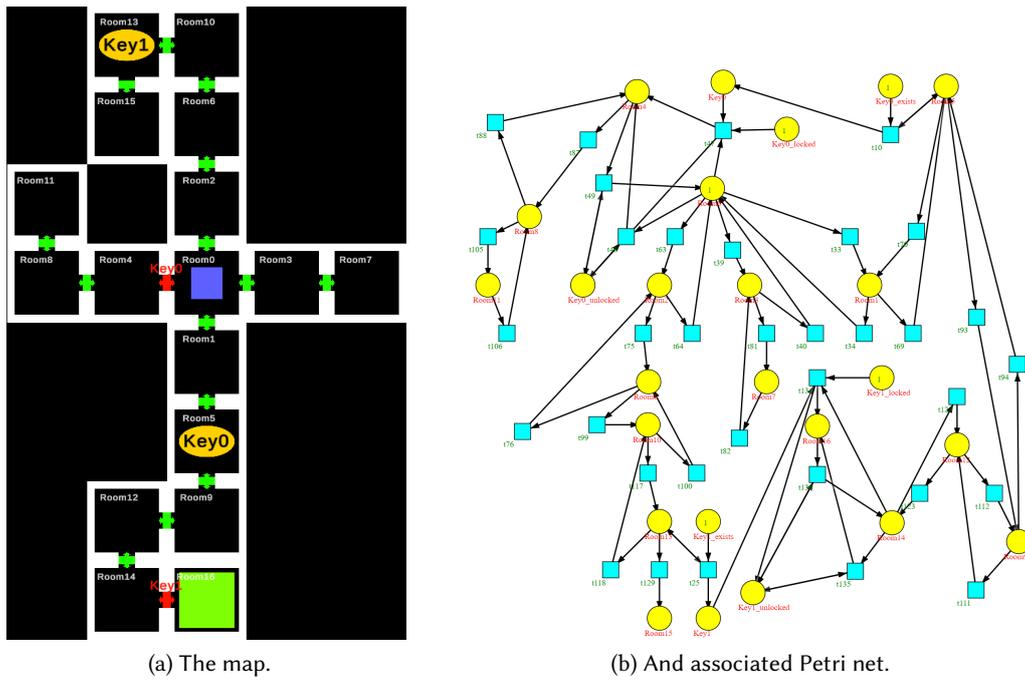


Figure 8: A nonviable map.

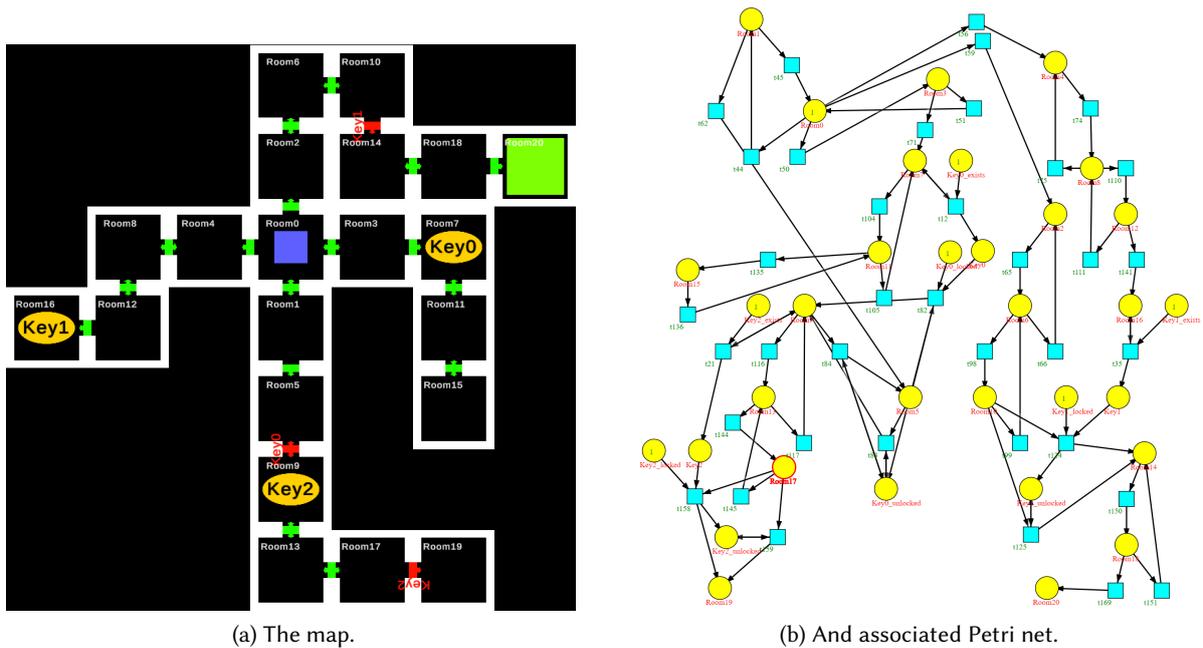


Figure 9: An inviable map.

From the state space generation report, it is also possible to conclude that some rooms are unreachable, namely Room9, Room13, Room14, Room17, Room18, Room19, and Room20.

7. Conclusions and future work

The proposed approach proved to correctly identify the viability level of randomly generated maps/graphs, which can be used as the base for roguelike games. Maps, where the final room cannot be reached, are classified as inviable, while maps, where the reachability of the final room depends on the

movement options of the player along the game, are classified as nonviable. Viable games (those where it is always possible to find a path leading to the final room) are successfully classified, and rooms that cannot be visited are identified and can be removed from the map, saving resources when the execution of the game is concerned.

The proposed approach can be used to validate more sophisticated games, namely other types of role-playing games (RPG), e.g., games with 2D or 3D grids, with any number of transitions per room, or multi-player support. To support multi-player extensions, the Petri net class to be used must consider distinguishable tokens, meaning an adequate class of high-level Petri nets needs to be selected.

In future works, it is foreseen to set up a complete environment to generate fully playable roguelike games, integrating the proposed approach to validate the viability of the randomly generated map (where all the manual steps referred to in this paper will be integrated, allowing automatic execution of the whole flow).

Acknowledgments

This work was financed by Portuguese Agency FCT – Fundação para a Ciência e Tecnologia, in the framework of project UIDB/00066/2020, and under the PhD scholarship with the reference PRT/BD/154920/2022. The authors would like to thank Miguel Vitória for making available the application that allowed the generation of the maps used to validate the presented work.

References

- [1] Daniel, K., Nash, A., Koenig, S., & Felner, A. (2010). Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39(January), 533–579. <https://doi.org/10.1613/jair.2994>
- [2] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numer. Math.* 1, 269–271 <https://doi.org/10.1007/BF01386390>.
- [3] Russell, S. J., Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson Education, 3rd edition, 2003. ISBN: 9780136042594.
- [4] Hart, P., Nilsson, N. & Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions On Systems Science And Cybernetics*. 4, 100-107 (1968)
- [5] Alves, J. M. M. (2017). Path Planning and Collision Avoidance Algorithms for Small RPAS, MSc Thesis; Instituto Superior Técnico, Portugal. https://fenix.tecnico.ulisboa.pt/downloadFile/1126518382183333/JulianaAlves_Thesis.pdf
- [6] Reshamwala, A. (2013). Robot Path Planning using An Ant Colony Optimization Approach : A Survey. pp. 65–71.
- [7] Khatib, O. (1985). Real-time obstacle avoidance for manipulators and mobile robots. *Proceedings - IEEE International Conference on Robotics and Automation*, 500–505. <https://doi.org/10.1109/ROBOT.1985.1087247>.
- [8] Barreto, F. M., Julia, S. (2021). Formal Approach Based on Petri Nets for Modeling and Verification of Video Games. *Computing and Informatics* vol. 40, number 1, (216–248)
- [9] Gomes, L., Ribeiro-Gomes, J. (2023). Analysing navigation paths in constrained graphs using Petri nets, in: *ISIE'2023 – 32nd IEEE International Symposium on Industrial Electronics*; 19-21 June 2023, Aalto University, Helsinki-Espoo, Finland; DOI: 10.1109/ISIE51358.2023.10228055
- [10] Fan, X., Li, B., Sisson, S. (2018). The Binary Space Partitioning-Tree Process. *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*
- [11] Wolverson, H. (2001). *Hands-on Rust - Effective Learning through 2D Game Development and Play*. ISBN: 9781680508161. <https://pragprog.com/>
- [12] Vitoria, M. (2023). Geração de Mapas para Jogos Tipo "Roguelike" e sua Validação Utilizando Redes de Petri. MSc Thesis. NOVA University Lisbon, Portugal
- [13] J. P. Barros, L. Gomes (2004). Net Model Composition and Modification by Net Operations: a

- Pragmatic Approach. INDIN'2004 – 2nd IEEE International Conference on Industrial Informatics; 24-26 June 2004; Berlin, Germany. DOI: 10.1109/INDIN.2004.1417350
- [14] Pereira, F., Moutinho, F., Costa, A., Barros, J.P. Campos-Rebelo, R., Gomes, L. (2022) IOPT-Tools – From executable models to automatic code generation for embedded controllers development. PETRI NETS 2022 - 43rd International Conference on Application and Theory of Petri Nets and Concurrency; June 19 - 24, 2022, Bergen, Norway; in L. Bernardinello and L. Petrucci (Eds.): PETRI NETS 2022, LNCS 13288, pp. 127–138, 2022. https://doi.org/10.1007/978-3-031-06653-5_7
- [15] J. Ellson, E. Gansner, L. Koutsofios, S. North and G. Woodhull, "Graphviz – Open Source Graph Drawing Tools", in editors P. Mutzel, M. Jünger and S. Leipert, "Graph Drawing", 2002, Springer Berlin Heidelberg, pages 483–484.
- [16] F. Pereira, F. Moutinho, L. Gomes, J. Ribeiro, R. Campos-Rebelo, "An IOPT-net State-Space Generator Tool", INDIN'2011 - 9th IEEE International Conference on Industrial Informatics, July 26-29, 2011, Caparica, Lisbon, Portugal; pp. 383 - 389; ISBN 978-1-4577-0434-5; DOI 10.1109/INDIN.2011.6034907
- [17] F. Pereira, F. Moutinho, L. Gomes, "Model-checking Framework for Embedded Systems Controllers Development using IOPT Petri Nets", ISIE'2012 – 2012 IEEE International Symposium on Industrial Electronics; 28-31 May 2012, Hangzhou, China
- [18] L. Gomes, J.-P. Barros, "Refining IOPT Petri Nets Class for Embedded System Controller Modeling", IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, 2018, doi=10.1109/IECON.2018.8592921