Procedural Generation of Maps and Narrative Inclusion for Video Games

João Ulisses[1], Ricardo Gonçalves[1,2], António Coelho[1,2]

[1] DEI, FEUP
Rua Dr. Roberto Frias s/n 4200-465, Porto, Portugal

[2] INESC TEC
Rua Dr. Roberto Frias s/n 4200-465, Porto, Portugal

jpulisses@gmail.com, refg@fe.up.pt, acoelho@fe.up.pt

Abstract. One of the biggest pitfalls digital games have is the lack of replay-ability and consequently a limited lifespan. The reason behind this problem is that in most cases, when the player reaches the end of a game, there is nothing more to explore. To address this issue many methods have been researched and implemented. These methods are mainly procedural content generation algorithms (PCG), non-linear stories and the ability for players to create their own content. The problem however remained as the integration of this type of content and their quality is not easy or not done as desirable, especially narratives. Due to the advantages PCG algorithms, they are being integrated into project Orion, a serious game, along with a narrative that aims to create a versatile tool independently of the programming language. This promotes replay-ability as more content can be generated and will help students in their studies since the game may last longer, as well as teachers to use this tool dynamically according to what they want to teach. In addition, since narrative is also an extremely important component of game content, an XML schema has been created to store static stories that will be integrated into the generated game levels seamlessly. As a result, Project Orion will show the implemented dungeon generation algorithm and narrative integration and how it is possible to generate interesting multi-level dungeon games that incorporate all elements in a narrative world which will arch for the player to follow in a 3D virtual world where any content can be easily added, even by people that do not have programming skills.

Keywords
Procedural Content Generation in Games; Serious Games; Dungeon Crawler; Level Generation; Dynamic Generated Content; Narrative

1. Introduction

Serious games are becoming an alternative method for imparting knowledge and skills, both in academic and enterprise environments. This technology, as the name implies, uses game characteristics such as rules, challenges, rewards, interaction and feedback, and wraps them together to solve a problem. By achieving a balance between fun and the problem addressed, these types of games can become a motivational force to engage learners to learn the desired knowledge and skills.

In spite of the intrinsic and extrinsic motivational factors inherent to any game, whether serious or not, one of the biggest engagement pitfalls has always been their longevity and replay-ability. Once the novelty wears off, players rarely feel the need to come back, especially if it is a single player game. This is where the procedural generation of content in Games (PCG-G) may help, by generating content in real-time, either as the game evolves or every time a new game is initiated. Of course content has a very broad meaning, however in this work we are particularly addressing the generation of game levels or maps and integrating it with content that may be generated in multiple ways even external to the game, such as the narrative. The inclusion of narrative in the game is also extremely important since storytelling has always been a crucial element in human development [11], and also functions as an intrinsic motivational factor [10].

In addition to the replay-ability potential that PCG algorithms may introduce, there is also a great advantage of reducing the workload on the design teams by generating to a point where artists can pick the generated assets and polish and add further detail, enriching the overall quality of the application and reducing the time and cost it would take if content was done from the beggining [1]. Doing integration between the generated content and the narrative is a problem, but with methods similar to Natural Language Processing it is possible to achieve this, but also have the type of interaction between the writer directly to the program that allows development teams to save time.

This paper is organized as follows. Section 2 defines the context of PCG and presents the problematic of narrative integration and the proposed solution, integrated with all other generated content showed in the following sections for a serious game for the teaching of computer programming [2]. In section 3 narrative integration is explained and how it is achieved, methods will be shown such as the use of an XML file to describe a narrative so that the generated levels tell a story, as well other content.

In section 4 it is given a context of PCG base algorithms for dungeon generation and the algorithm used is specified thoroughly showing it is strong points and why it was used. In section 5 the dynamically-generated world in Project Orion with all the results from all the integrations, showing advantages of utmost importance in the way it was made allowing not only better results, better team integration and other advantages, this section also includes demonstration. Finally, the conclusions and future work are presented in section 6.

2. State of the Art

This section will begin to explain what procedural content generation in games is, then what type of content can be generated as well as the distinction between adaptive and non-adaptive PCG algorithms. Finally, a few PCG algorithms will be explored.

Procedural Content Generation in games (PCG-G) can be defined as the automatic creation of content, which is achieved through the implementation of an algorithm tailored to a specific end [3]. This process can be achieved through a random process or through deterministic reconstruction based on parameters which will create the same object every time. Since the main focus of this article is the creation of continuous new experiences in game environments, the focused PCG-G algorithms will have some degree of stochasticity.

Even though content is intended to be greatly different in every generation iteration to avoid the feeling of repeating in the player, it is a good policy not to be completely random. Instead PCG-G algorithms should implement some degree of stochasticity that adheres to constraints, which can be affected by associated parameters, of what content can be generated and how [4]. By generating content randomly within that range of constraints, the result will be adequate and within the scope of the game. Furthermore, by allowing the player to act, directly or indirectly, upon the parametrization of the algorithm the results will be tailored to the choices he made [5]. These algorithms fall under the category of adaptive-PCG and can be useful, for instance, to adjust the difficulty of a generated level to the player's proficiency.

But what type of game content can be generated by PCG algorithms? Theoretically, almost anything can be generated content. Hendrikx et al. [1] defines four classes of game content that can be generated procedurally:

- Game Bits include textures, item properties, sounds, vegetation, buildings, behavior, weather and natural elements.
- Game Space pertains to the environment present in-game, whether they are indoor (dungeons; rooms; houses; etc.) or outdoor (forests; space; underwater; etc.).
- Game Systems such as ecosystems, road networking, urban environment development and entity behavior management.
- Game Scenarios define puzzles; story and level concept.

Another important aspect to take into consideration is whether PCG-G algorithms are run online or offline [3]. A PCG-G online algorithm is usually executed in run-time, as the game progresses, generating new content on the fly. For example Minecraft[1] is continuously generating the environment as the player moves into uncharted territory.

Online algorithms may be adaptive and have two critical requirements. The first of these requirements is speed of execution because if these algorithms take too long doing the necessary computations it will have a detrimental effect on the player's immersion. As for the second requirement, the algorithm needs to ensure that a minimum level of quality is attained.

On the other hand, offline algorithms are usually used during development and subject to specific changes to meet the desired results and as such, these are almost always non-adaptive.

3. Narrative Integration

[1] https://minecraft.net/, 2014

Having a new map to explore every time we play a particular game is without a doubt exciting (at the least the first few times) and can be a source of motivation for the player to explore the game once again. This effect can be significantly amplified if the map also presents a story for the player to follow. But since the maps keep changing every new playthrough, it becomes necessary to have some mechanism that allows the new map to contain the narrative content and plot points that compose the story. The game Diablo[2] is fairly renowned for achieving this effect - while the maps keep changing every new playthrough, the storyline (quests), key objects, places and characters remain the same.

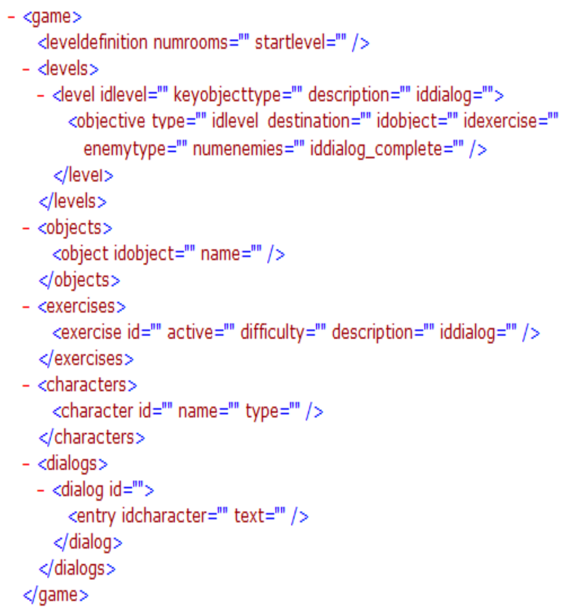
In order to complement the generated maps with a story we had to somehow store the narrative information, as well as the programming exercises associated with each map/level. By storing this information we could then adapt our map generation to match the needs of the story as well as of the exercises.

To this end we created a data format based on XML, as shown in figure 1. This format allows the definition of each of the individual levels, associating with them dialogs, descriptions and objectives. These objectives are associated with exercises, the level they unlock (useful to establish links between levels) and other information related with the narrative and gameplay mechanics, such as object collection, enemy encounters, etc.

The exercises definition is fairly simple but requires that the identity field matches the ones in the exercise evaluator module, for more information about this module refer to previous work [2].

Finally, to give story more life, characters and dialogs are elements also present in the format, although at this stage in a very rudimentary form. Through these tags are the main character, player controlled, as well as other non-player controlled characters can be defined and have dialogs associated with them.

[2] http://eu.blizzard.com/en-gb/games/d2/ , 2014



```
- <game>
    <leveldefinition numrooms="" startlevel="" />
  - <levels>
    - <level idlevel="" keyobjecttype="" description="" iddialog="">
        <objective type="" idlevel_destination="" idobject="" idexercise=""
          enemytype="" numenemies="" iddialog_complete="" />
      </level>
    </levels>
  - <objects>
      <object idobject="" name="" />
    </objects>
  - <exercises>
      <exercise id="" active="" difficulty="" description="" iddialog="" />
    </exercises>
  - <characters>
      <character id="" name="" type="" />
    </characters>
  - <dialogs>
    - <dialog id="">
        <entry idcharacter="" text="" />
      </dialog>
    </dialogs>
  </game>
```

Figure 1. Base XML format for storing narrative and exercise information.

As this stage of development the dialogues are static and do not allow the player to have different choices (though they can explore in different ways and do any order of exercises), but still means that the story itself is completely linear, however like any content here, it can be changed and improved by the writer having low or none implementation costs and allowing the player to make decisions.

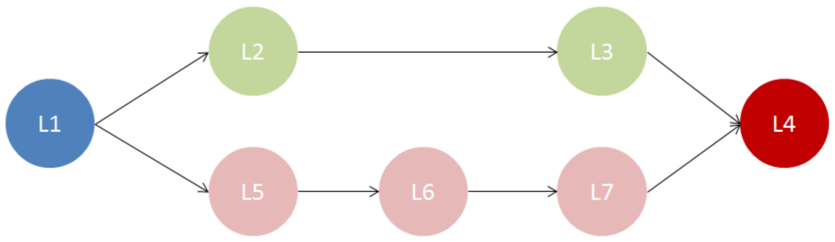


Figure 2. Example of a narrative arch.

4. PCG-G Algorithms for Dungeon Generation

Dungeon level is a instance of a video game level, this instance is often called dungeon because the player has to explore it in order to complete that level.

Game level generation usually have specific constraints that are associated with the type of level being generated which usually require a specific oriented algorithm. In this sense an algorithm for generating an open world with oceans, mountains, etc.

may not be the most adequate to generate a dungeon area. For this reason, in this section, some base algorithms will be presented which are mainly oriented to generate dungeon levels.

4.1. Random Room Placement

The algorithm used consists mainly in populating a designated gridded area with rooms of random size and connecting them through corridors. As parameters this algorithm can receive the number of rooms to generate, the width and the height ratio.

The general workflow of this algorithm is as follows: firstly it starts by parsing the parameters, proceeding to a loop state where rooms are generated. For each room generation loop iteration, the generated room must be first validated before placing it on the map. For example if the room overlaps another or is outside the bounds of the map it is discarded and the room is redone. This process loops until all rooms have been generated or until some other exit condition is met, such as reaching the maximum number of attempts. After all rooms have been successfully generated the next step is the creation of the corridors. This step usually involves using a path finding algorithm such as A* to find the closest path from room to room. When all rooms have been connected, if there are still unconnected doors in some of the rooms, additional dead end corridors may be added or like in Orion non-openable doors can be generated here. Orion also creates doors in spaces between rooms and corridors that are not walls.

4.2. Space Partitioning For Room Placement

In the previous example, during room generation and map placement, it could occur with some frequency that rooms would get discarded. With space partitioning algorithms, such as BSP trees and quadtrees [6] this problem is circumvented because rooms are placed in the empty generated areas and therefore do not overlap other rooms.

BSP tree random placement starts by sub dividing the initial map area into smaller sections by choosing a horizontal and vertical direction. This sub-division is done for *n* iterations. After reaching the n iterations rooms are inserted into the subdivided areas. Note that rooms must be of the same size or smaller than the regions they will be placed on. After all rooms have been successfully placed, the next step is connecting the rooms by looping through all split regions and connecting each immediate neighboring regions ensuring that all rooms get connected.

Similarly to BSP trees, quadtrees start to divide a large area but in this case it divides it in four equal squares. For each new area it is again subdivided until either an area reaches the minimum room size or if the stochastic element of the algorithm stops subdivision. For each leaf a room is placed such that it is entirely contained within the leaf area.

4.3. Cellular Automaton Method

While the two previous algorithms are oriented to generate room-like dungeons, a cellular automaton algorithm is oriented to generate cavern-like areas (closed areas with a opening) [7]. The first step in this algorithm is to fill an area randomly with walls and empty spaces. Subsequently the algorithm parses each of the grid cells and applies the 4-5 rule: a cell *C* at position *p* becomes a wall if at least five (including itself) of the eight (diagonals included) immediate connected neighboring cells are walls. Parsing each of the cells need to be done simultaneously instead of parsing one by one and using the value of the previous to calculate the next. The reason for this is purely to make it will look less machine processed. One of the problems this algorithm has is that it tends to have very inconsistent results such as wide open areas or disjoint areas. This problem can be solved however by adding an additional refinement rule before applying the initially defined rule[3] . This new rule contains the first one plus: a cell *C* becomes a wall if two or less cells within 2 steps from *C* are walls. A mathematical notation of these rules can be for example:

Rule 1: $W'(p) = C_1(p) \geq 5 \text{ or } C_2(p) \leq 2$ Rule 2: $W'(p) = C_1(p) \geq 5$

Where $W'(p)$ represents if there is a wall in a particular position *p*, and $C_n(p)$ represents the number of cells within *n* steps of the position *p* that are walls.

4.4. Algorithm Developed for the serious game Orion

The developed algorithm generates different unique levels for each player each time it is executed. It is a requirement that maps are generated at game run-time.

This algorithm is inspired on a random room placement approach used in the game TinyKeep[4], however with some important differences such as disconnected rooms and multi-level maps. The reason behind this choice was strongly due to the aesthetic look and topology that could be achieved with this algorithm, which feels more varied and less artificial, consequently having more immersion potential which is one of the main goals of the serious game this algorithm [9] is meant to integrate.

As many other algorithms this one also requires some entry parameters to function. These parameters are the number of rooms to be generated, a set of values representing the mean and deviation parameters for a Gaussian distribution function, a maximum room size ratio, the minimum room size and finally the minimum boundary (space) between rooms. These parameters will be further explained as the steps involved in this algorithm are detailed.

The algorithm is divided in different sequential steps that use the data from the previous step to feed the next until the final result is reached. The first step is randomly creating rooms and positioning them over the map area without worrying about overlapped rooms. The room creation is basically a loop that generates as many room as set in the entry parameters. For each room generated, before accepting it, it is verified if it meets certain constraints such as room ratio, to avoid very narrow rooms for example, or if the room has a minimum size, for example to avoid single cell rooms. Rooms random generation is done by using a Gaussian distribution function to ensure that most rooms are within the same size/ratio range and only a few are bigger,

[3] http://pixelenvy.ca/wa/ca_cave.html, 2014

[4] http://tinykeep.com/, 2014

creating a more natural distribution and avoiding having a map filled with only huge rooms.

The next step on this process is to ensure that no rooms are overlapped and to achieve this, a simple separation steering function is used on each room until no room overlaps. This steering process uses the entry parameter minimum border, which can be equal or higher than zero, to increase (or use the default room size if it is zero) to check if two rooms overlap.

The main reason behind this parameter is to prevent rooms to be closely packed, allowing space for later corridor generation. Also, the separation steering process ensures that the room is snapped to the grid to ease the corridor generation. Because this is a multi-level map (each level is a game level which can have two floors), when a new level is added to a generated map in a previous iteration of this algorithm process, some rooms will be selected as level transition rooms.

The selection of these transition rooms is based on their area (the higher the area the more chances it has to be selected as a transition room) and they are meant to establish a transition point between the current level and the next one.

In figure 3, these rooms can be seen as blue if they are transition rooms to the next level, or pink if it is a transition room to the previous level. Upon creating a new map, the previous map transition rooms are passed and integrated within the selected rooms of the new level. These rooms remain fixed during the process of separation steering of the new level to ensure they retain their relative position to the parent's level.

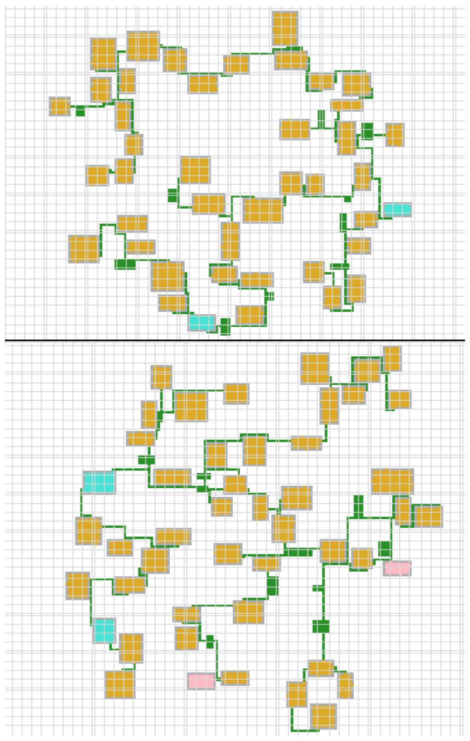


Figure 3. Example of a generated map with two levels. Blue rooms are transition rooms to the next level and pink rooms are transition rooms to the previous level.

With a list of selected rooms the next step is the creation of a graph that connects rooms together. This is achieved by constructing a graph similar to a relative neighborhood graph, except that this one was done in such a way that in some cases a map could have disconnected sub-graphs which could be a potentially way to allow some kind of teleportation mechanic in the game. This can be disabled by simply selecting the two closest nodes of each closest sub-graph and connecting them ensuring this way that every room is reachable by normal means. One of the side effects of this approach was that some rooms were connected by several nodes, creating multiple paths between nodes and to solve this, a minimal spanning tree is generated from the initial graphs, reducing the number of connections between nodes.

The final step is connecting rooms using the graph. To achieve this, a path finding algorithm was used, more specifically A*, which is used to create a route for each graph edge.

To avoid non-upright (zig-zag) corridors the algorithm punishes heavily direction alteration, ensuring that corridors have a straighter format. Also the different cost values are given to cells to ensure that the algorithm will always favor existing corridors, adding more realism and variety to the map, like intersections. Finally, after a path as been found between two rooms, this path is checked against all non-selected rooms and if an intersection occurs, the room is integrated as part of the corridor and no longer being a room adding further detail to the general map layout (so corridors are not always lines connecting rooms).

5. Project Orion and Results

Project Orion[12][2] as shown in figure 4 is a roguelike[5] game, having the main characteristics such as the fact that Orion generated procedurally as shown in figure 5 and 6, the objects have different names and descriptions, and the game was meant to play alone (though it is easy to implement multiplayer because of the way it was dynamically generated by just adding one more player and create a connection between the players).

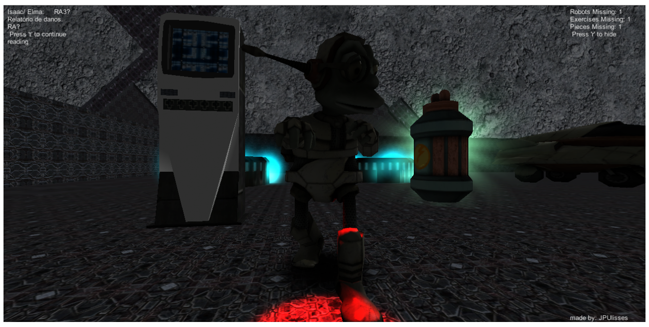


Figure 4. Orion, with different objects to pick and interact.

This project used the algorithm presented in section 4 to generate the map, that information was used to draw 3D elements such as rooms, corridors and placing doors.

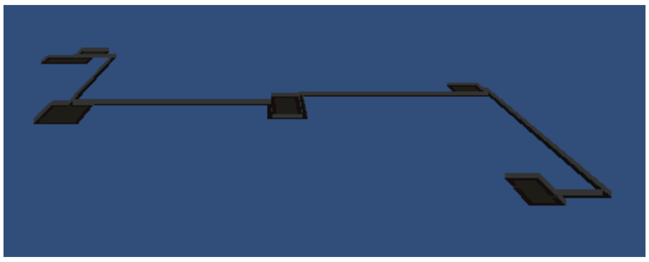


Figure 5. A generated level with a few rooms and doors.

[5] http://www.roguebasin.com/index.php?title=Berlin_Interpretation, 2008

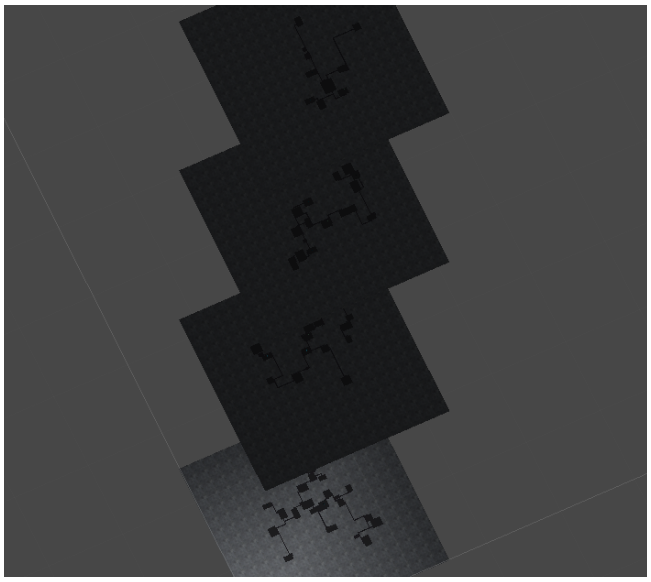


Figure 6. The generated levels in Y axis, each with more rooms than the example given before. The player teleports from one level to another after completing the objectives, physical stairs could be added if they had fit into the theme of the game.

In the game most content is present and editable in a external XML file, such as gameplay mechanics, objects, enemies, programming exercises and the narrative itself. These allow to dynamically create a 3D world with content to play and interact with it in different ways, allowing the player to explore and solve the problems presented. The player can make use of the dialog following the events and the conversation between characters or be guided in the right way, the programming exercises can also offer tips and help. This content can be seen, and edited externally in the demonstration[6].

The way it was programmed dynamically and using Unity3D, allows any content to be replaced at any time by referencing another content. This also allows the content to be adapted to different programming languages. Thanks to Unity3D, these capabilities allow teams to work independently and put together their work at a later stage, and also constantly update and improve it, with low cost of implementation. For example in the demonstration, some objects such as the enemy robot that comes from Unity3D Platform Tutorial, have a JavaScript code, however it is instantiated and asked to be destroyed by C# code.

[6] https://feupload.fe.up.pt/get/grRy5yI2kcOcK0a, 2014

6. Conclusions and Future work

From the aesthetic and functional point of view, the generated maps achieve their goals, to generate dynamic content as shown, however it is possible to go further and not only use this to speed the process of generating large content, but also allow multi-disciplinary teams to work together with ease and update their work on later stages of development with low implementation cost. The extra capability of the platform used (Unity3D) to instantiate and communicate with objects of other programming languages enhances this capability even further.

PCG lay the foundation for a rich dungeon environment and ensure that there are several pathways that can be taken to reach the end of each map (level). Coupled with the XML narrative description file, these maps are brought to life which acts as motivation factor for the player to explore the map and discover the story, consequently having to solve the proposed exercises.

Everyday the games that use PCG methods require people from multidisciplinary areas, that not only must know how to program, but they must know the generated content in order to predict it, limit the way it will be generated if needed and generate it in a pleasant way for the user or with some degree of expected quality. This claim should be changed as it is not always true as shown, teams can work separately, each with different areas of knowledge, however the team in charge of putting all together, generating the content based on the work of the other teams, must have some knowledge of that area in order to create the best result.

For the narrative at this stage it is still done manually, however in the future we hope to generate the stories and provide a complete 'new' experience each playthrough, as well as varying exercise parameters so that these are also unique to each student. Additionally the XML format needs to be further enhanced to allow player dialog choices, multi-dependent objectives, non-player character interaction, non-serious side quests and elements, events, locations, etc. However at this stage it is possible to have player-created content as shown in the demonstration, while this exists for entertainment games, it is harder to do on serious games as we did, this can create new experiences and promote experience sharing between the students.

The algorithm for dungeons is not finished however, for the moment all rooms are rectangular which may suit most cases, but to increase immersion some rooms could be improved, for example, by using predefined models that add more detail to the map. Another evolution is to have a predetermined pathway that is passed to the algorithm and generate around it a map level. This could be useful in the eventuality of wanting to define a set of rooms with specific exercises and narrative plot points.

Preliminary results show that most players pointed out it was easy to get to play and to play Orion. This information includes the game itself, how user friendly it was, but the whole process since downloading the game, which thanks to Unity3D makes the process of building the project much easier, creating a executable file ready to be played. This also shows the game is ready to be tested, however more content is needed for further experiences, these can done either by professors or students if they wish to do so.

Acknowledgements

"The Media Arts and Technologies project" (MAT), NORTE-07-0124-FEDER-000061, is financed by the North Portugal Regional Operational Programme (ON.2 – O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT)."

References

1. Hendrikx, Mark, et al. "Procedural content generation for games: a survey." ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), 2013.
2. Coelho, António; Kato, Enrique; Xavier, João; Gonçalves, Ricardo: Serious Game for Introductory Programming. In Proceedings of the Second International Conference on Serious Games Development and Applications (SGDA 2011), Lisbon, 2011.
3. Togelius, Julian, et al. "Search-based procedural content generation: A taxonomy and survey." Computational Intelligence and AI in Games, 2011.
4. Togelius, Julian, et al. "What is procedural content generation?: Mario on the borderline." Proceedings of the 2nd International Workshop on Procedural Content Generation in Games. ACM, 2011.
5. Georgios, Yannakakis and Togelius, Julian. "Experience-driven procedural content generation." Affective Computing, IEEE Transactions, 2011.
6. Togelius, Julian; Shaker, Noor; Nelson, Mark J. "Procedural Content Generation in Games: A Textbook and an Overview of Current Research", 2014
7. Johnson, Lawrence; Georgios, Yannakakis; Togelius, Julian. "Cellular automata for real-time generation of infinite cave levels." Proceedings of the 2010 Workshop on Procedural Content Generation in Games. ACM, 2010.
8. Valtchanov, Brown, Joseph. "Evolving dungeon crawler levels with relative placement." Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering. ACM, 2012.
9. Doull, Andrew. "The death of the level designer." Internet: http://pcg. wikidot. com/the-death-ofthe-level-designer, last accessed in 2008.
10. Pintrich, Paul R., et al. "Reliability and predictive validity of the Motivated Strategies for Learning Questionnaire (MSLQ)." Educational and psychological measurement, 1993.
11. Schank, Roger C., and Robert P. Abelson. "Knowledge and memory: The real story.", 1995.
12. Ulisses, João; Coelho, António: "Solução de geração procedimental de níveis de jogo", Oporto, June 2014.
13. Togelius, Julian, et al. "Procedural Content Generation: Goals, Challenges and Actionable Steps", 2013.