UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

VINICIUS MILANI RODRIGUES DE FREITAS

Procedural generation of cave-like maps for 2D top-down games

Porto Alegre
2021

VINICIUS MILANI RODRIGUES DE FREITAS

Procedural generation of cave-like maps for 2D top-down games

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering

Advisor: Prof. Dr. Dante Augusto Couto Barone
Coadvisor: Dr. Leonardo Filipe Batista Silva de Carvalho

Porto Alegre
2021

CIP — CATALOGING-IN-PUBLICATION

Milani Rodrigues de Freitas, Vinicius

Procedural generation of cave-like maps for 2D top-down games / Vinicius Milani Rodrigues de Freitas. – Porto Alegre: 2021.

55 f.

Advisor: Dante Augusto Couto Barone

Trabalho de conclusão de curso (Graduação) – Universidade Federal do Rio Grande do Sul, Escola de Engenharia. Curso de Engenharia de Computação, Porto Alegre, BR–RS, 2021; Coadvisor: Leonardo Filipe Batista Silva de Carvalho.

1. Video game. 2. Procedural generation. 3. Dungeon. 4. Unity. 5. Cave. I. Couto Barone, Dante Augusto, orient. II. Batista Silva de Carvalho, Leonardo Filipe. III. Título.

Dedico este trabalho à minha família, amigos e namorada.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dante Augusto Couto Barone for accepting my work proposal, for guiding me through the many changes we had, for connecting me to the right people and for his incredible availability and for being always helpful. I also would like to thank my co-advisor Leonardo Filipe Batista Silva de Carvalho for the great support and crucial help.

I'd like to thank Anderson Ferrugem and Raphael Campos for participating in some of our meetings and providing essential information for this work.

Even though I ended up changing the work idea I started with them, I'd like to thank my former advisor Lisandro Zambenedetti Granville and my former coadvisor Lucas Bondan for their understanding.

Finally, I want to thank my parents Nei and Nelci for always being supportive of me, helping and encouraging me to achieve all my goals in life. I'm also thankful for the emotional support and encouragements of all of my friends, and last but not least, my girlfriend Débora Alexia, who was always there for me during the development of this work.

“Video games are bad for you? That’s what they said about rock-n-roll.”

— SHIGERU MIYAMOTO

ABSTRACT

Procedural Content Generation (PCG) has been extensively used in game design. With it, game content can be created automatically with limited or indirect user input. This application of PCG is important in lowering the cost and time-consumption of game design, specially in current years since the video-game industry has been experiencing a dramatic growth over the last decades. The objective of this work is to create a system that generates cave-like maps for 2D top-down games, which can serve as a blueprint for game designers to build upon on the future. Additionally, the work intends to evaluate if the generated maps have a set of qualities to make them be perceived as good or desirable by players. The system was developed in the Unity Game Engine, utilizing the C# language and a combination of algorithms, that are presented and explained in this work. In order to evaluate the maps, criteria on what makes a good map were researched, and then applied a survey to test if the generated maps satisfied said criteria. A pretest questionnaire was made and answered by 23 participants, its results were used to develop an improved version of it, which was then answered by 163 participants. Finally, 9 out of 12 questions of the survey have reached their desired result.

Keywords: Video game. procedural generation. dungeon. Unity. cave.

Geração procedural de mapas de cavernas para jogos 2D top-down

RESUMO

A Geração Procedural de Conteúdo (PCG) tem sido amplamente utilizada no design de jogos eletrônicos. Com sua utilização, o conteúdo de um jogo pode ser criado automaticamente, através do uso de entradas de usuário limitadas ou indiretas. Esta aplicação de PCG faz-se importante para reduzir o custo e o consumo de tempo do design de jogos, especialmente nos anos atuais, já que a indústria de videogames sofreu um crescimento significativo nas últimas décadas. O objetivo deste trabalho é criar um sistema que gere mapas semelhantes a cavernas para jogos 2D top-down, que podem servir como uma base de desenvolvimento para designers de jogos. Além disso, o trabalho pretende avaliar se os mapas gerados possuem um conjunto de qualidades para que sejam percebidos como bons ou desejáveis pelos jogadores. O sistema foi desenvolvido utilizando-se do motor de jogos Unity, da linguagem C# e de uma combinação de algoritmos, que são apresentados e explicados neste trabalho. Para avaliar os mapas, pesquisou-se por critérios sobre o que constitui um bom mapa e, em seguida, uma pesquisa foi realizada através de um questionário para testar se os mapas gerados satisfazem esses critérios. Um questionário de pré-teste foi aplicado e respondido por 23 participantes, seus resultados foram utilizados no desenvolvimento de uma versão aperfeiçoada deste, que foi respondida por 163 participantes. Por fim, 9 das 12 questões alcançaram seus resultados desejados.

Palavras-chave: Jogo eletrônico, geração procedural, calabouço, Unity, caverna.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
CA	Cellular Automata
DFS	Depth-First Search
IMMS	Instructional Materials Motivational Survey
NPC	Non-Playable Character
PCG	Procedural Content Generation
RPG	Role-Playing Game

CONTENTS

1 INTRODUCTION

Designing a game involves a conjunction of many different disciplines, for example: art, animation, sound, story, character creation, etc. Although the term game design is widely used in the industry to refer only to the design of the gameplay aspect, all the separate parts of design must properly work together in order to provide the player with a good experience (ZUBEK, 2020).

In this chapter we are going to present the necessary background information in order to understand what motivated this work. First, we will discuss the concept of game design and how it evolved through the ages. Then we will present some information about the game industry and the costs of developing a game. Lastly we will explain what procedural content generation (PCG) is and show a division of categories to better understand it.

After the motivation of the work is explained, we will present our objectives and preface the content of the next chapters.

1.1 Game design

The importance of game design has increased throughout video game history. Designers of early games like Pong, shown here on Figure 1.1, or Spacewar! had very limited computing resources to work with, therefore it makes sense that they are simple and straightforward. Such games had little to no audio output, very primitive graphics and simple gameplay, sometimes borrowing ideas from well-established games, such as Pong, which is an electronic version of the Tabletennis sport (WOLF, 2007).

In time, the technological advances made possible for more complex games to be created. Designing a game became much more nuanced and time consuming.

1.1.1 Evolution of the video game industry

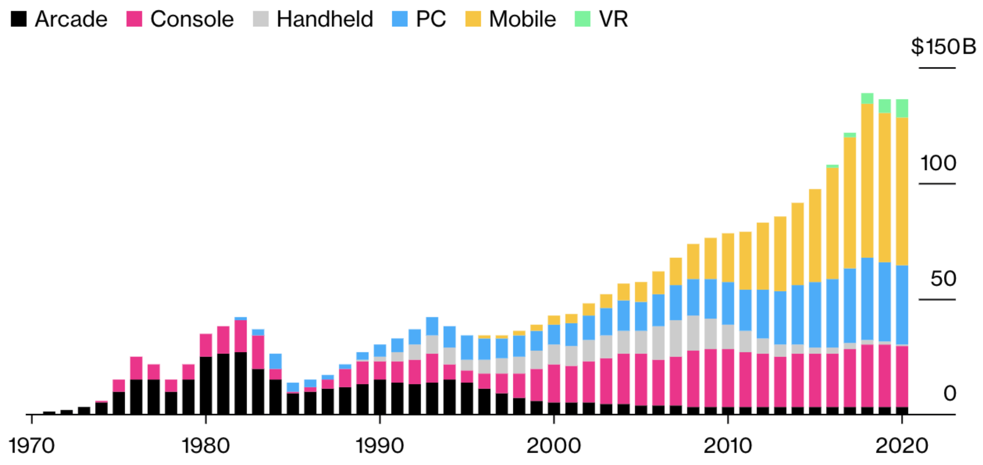
In 2020, the video game industry was already bigger than the movie industry and North American sports combined. The gaming industry has also experienced a big growth thanks to the COVID-19 pandemic (WITKOWSKI, 2020), since players are having more time at home for their hobby. But even before the pandemic, the industry was already in

Figure 1.1 – Atari's PONG arcade released in 1972

Source: (PONGMUSEUM, 2021)

fast-grow along the last decades, as shown in Figure 1.2.

Figure 1.2 – Video game industry revenue through the ages



Source: (SMITHERS, 2019)

On the other hand, the cost of making games has increased dramatically as well. The cost of making Final Fantasy 7 remake, which was released in April 2020, for example, was roughly $200 million, around $120 million more than the original Final Fantasy 7, which was released in January 1997 (KOUMARELAS, 2021).

Triple-A games, which are games that are produced by mid-sized or major publishers (STEINBERG, 2007), currently require the work of hundreds of people over the period of years to develop a single game. This is resulting in games not being as profitable for some developers, as few companies can afford developing long, diverse and polished games (SHAKER; TOGELIUS; NELSON, 2016).

One of the methods to reduce the cost of game designing is Procedural Content Generation (PCG), which uses algorithms to generate automatic content for the game.

1.2 Procedural content generation

In the context of video games, PCG is defined as "the algorithmic creation of game content with limited or indirect user input" (TOGELIUS *et al.*, 2011). In other words, PCG refers to software that is able to create game content by itself.

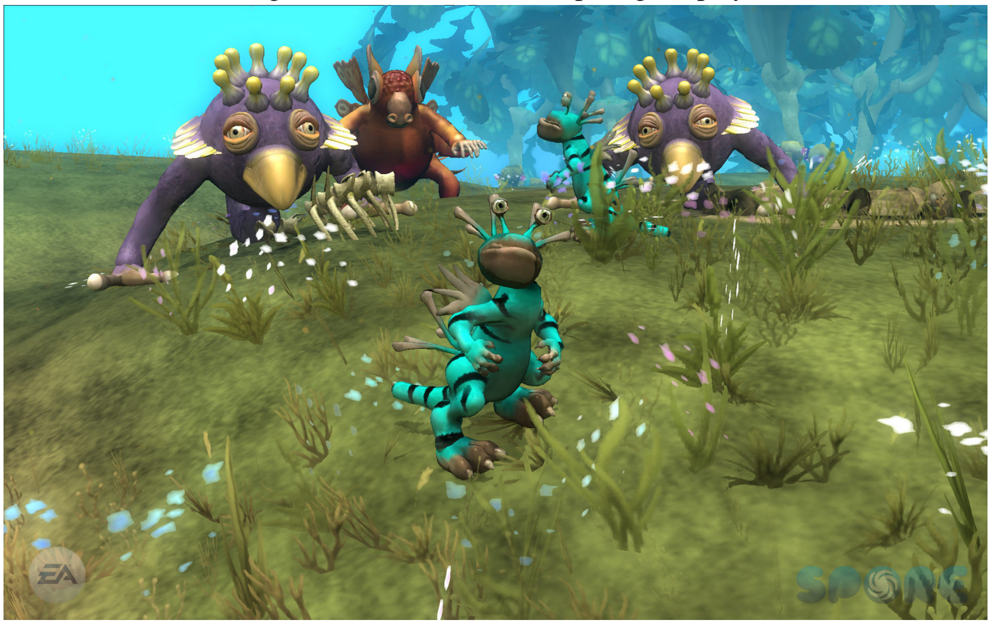
According to Doull (2008) there are 7 categories of PCG in games:

- **Runtime random level generation**: the generation of game levels while the game is being played. This is what people often think of when PCG in games is mentioned. In this category, an algorithm is responsible for generating random or pseudo-random levels for the game.
- **Design of level content**: in this method, the automatically generated content is used at the level design stage to supplement human design skills. An algorithm can, for example, populate an environment created by the designer rapidly. Or the designer may choose specific generated levels to expand on.
- **Dynamic world generation**: This technique is used to dynamically grow the environment that the player interacts on by using random seeds. In this case, the generated maps are never held in memory except as temporary structures to display.
- **Instancing of in game entities**: in order to reach a statistically insignificant chance of repetition, in-game entities, such as like monsters, items, non-playable characters (NPCs), have some of their properties procedurally generated. These properties may be, for example, the position of the entity, its size, structure, etc.
- **User mediated content**: this is a type of procedural generation where the user is in control. The technique offers a range of possibilities to users, who are responsible for putting them together in order to generate content.
- **Dynamic systems**: some real-world systems such as group or weather behavior can be modelled using PCG techniques. This is widely used in combination of artificial intelligence (AI) in order to, for example, make NPCs react differently according to certain weather conditions.

- **Procedural puzzles and plot generation**: this category is about using PCG in order to generate individual puzzle elements to increase replayability, e.g. changing door codes. Games that have its plot generated by PCG are also included in this category.

An example of a commercial game that was developed utilizing different PCG techniques is Electronic Arts's Spore. In this game, the player's objective is to evolve its own species, starting all the way from a microscopic organism until interstellar exploration. Entities on the game are generated by other players using in-game editors, which is an example of user mediated content generation. Worlds and galaxies are created through a combination of dynamic world generation and runtime random level generation. Moreover, some of the character animations are created procedurally (WRIGHT, 2007). Figure 1.3 shows user-generated characters standing on a procedurally generated world.

Figure 1.3 – Screenshot of Spore gameplay



Source: (ELECTRONIC ARTS, 2008)

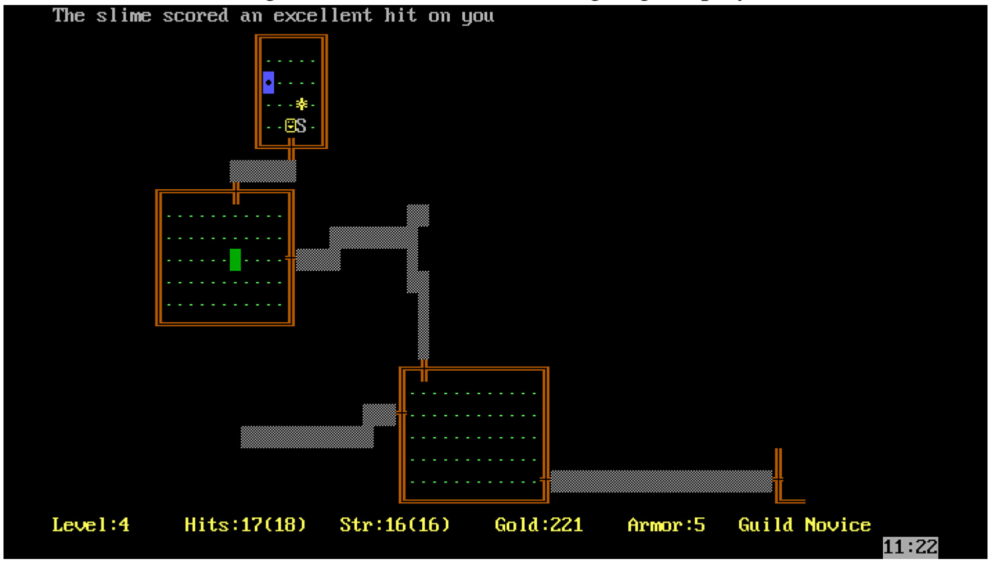
1.3 Dungeons in games

One of the most notable applications of PCG in games is the creation of procedurally generated dungeons. A dungeon is a labyrinthic environment that the player can explore, as well as collect items, slay monsters, fall into traps, etc. Although originally the term "dungeon" refers to a labyrinth of prison cells, nowadays, there are dungeons representing caverns, castles, forests, underwater environments, etc (SHAKER; TOGELIUS; NELSON, 2016).

Procedural generation of dungeons is also an extensive area of research, as is shown in the survey from Viana e Santos (2019). The survey shows that there are still challenges to be overcome and areas to develop, such as the generation of 3D dungeons.

This current definition of a dungeon can be probably tracked to the Dungeons and Dragons game, which is a tabletop Role-Playing Game (RPG) that had a huge influence in the development of video games throughout history. In fact, a specific type of computer RPG has spawned from the idea of exploring randomly generated dungeons: the roguelike genre. The name "roguelike" comes from the 1980's game Rogue, which featured procedurally generated dungeons that the player had to explore in search of an amulet (BREWER, 2016). A screenshot of a typical Rogue session can be seen in Figure 1.4.

Figure 1.4 – Screenshot of Rogue gameplay



The slime scored an excellent hit on you

Level:4 Hits:17(18) Str:16(16) Gold:221 Armor:5 Guild Novice

11:22

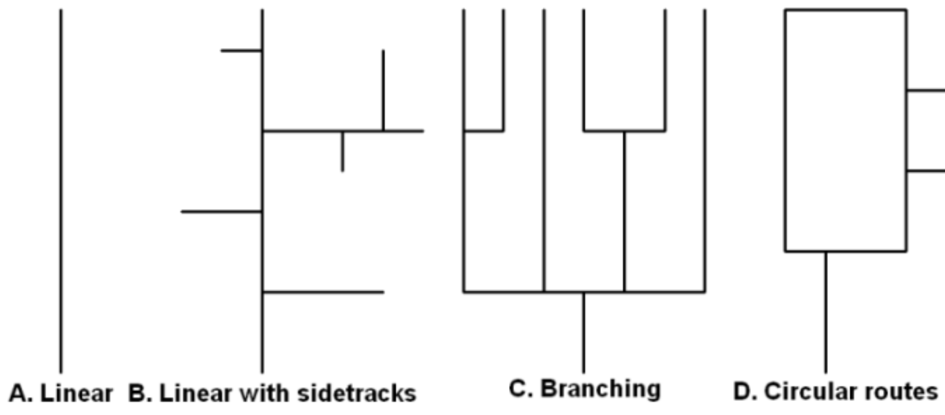
Source: (EPYX, 1985)

In Melan (2006) it is suggested that the design of the dungeon structure is very important to the creation of a good dungeon map. According to the author, a good map design is one that embodies the factors that make playing on a dungeon fun: exploration, decision making, consistent pace of action, discovery of secrets. Four different basic forms of dungeon maps, created from the author experience with RPG, were presented in Melan (2006), shown here on Figure 1.5.

1.4 Objectives

Taking motivation from the information presented in the previous sections, this work has been elaborated with the objective of creating a PCG system to design cave-like

Figure 1.5 – Basic dungeon structures



Source: (MELAN, 2006)

dungeon maps for 2D top-down games.

More specifically, we will:

- Generate varied and cave-like dungeon maps from the combination of different algorithms.
- Congtribute to the PCG category of 'design of level content' by generating leves that can serve as a blueprint for game designers to build upon on the future.
- Provide maps based on a set of criteria found in bibliography for what is considered a good dungeon map.
- Evaluate if the generated maps match the criteria through a survey applied to video-game players.

1.5 Organization of this work

Next, on Chapter 2, we will cover works that are related to this one. After that, on Chapter 3 our proposal will be expanded on and some background information will be provided in order to fully understand it. Following that, on chapter 4 we will provide details on our development and implementation. Chapter 5 will cover the creation of the survey and its results. Lastly on Chapter 6 we will discuss the conclusions of our work.

2 RELATED WORK

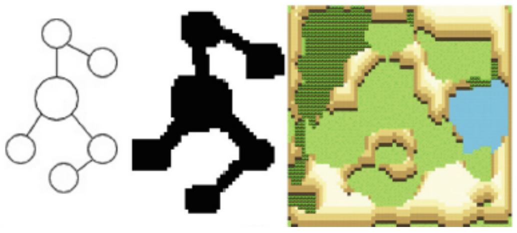
While looking for works related to this one, we searched for researches that focused specifically on dungeon or on content generation for games and also, on works that had their focus on cave generation.

2.1 Conditional Convolutional Generative Adversarial Networks Based Interactive Procedural Game Map Generation

This paper by Ping e Dingli (2020) suggests the usage of Conditional Generative Adversarial Network and Convolutional Neural Network to create a game design system. The system takes a gameplay area map defined by the user as input and generates a complex map with the same design pattern.

In Figure 2.1 we see the image designed by the user in the first two steps and, in the last step, the generated map. The variety of the resulting maps depend on the training of the network as well.

Figure 2.1 – Example of the system's usage



Source: (PING; DINGLI, 2020)

2.2 Cellular automata for real-time generation of infinite cave levels

In this paper a simple cellular automata (CA) algorithm is used in order to generate, in real time, infinite cave-like levels. The algorithm was tested in a game called Cave Crawler, which is a top-down view game where the players have to traverse the generated tunnels while defeating waves of enemies.

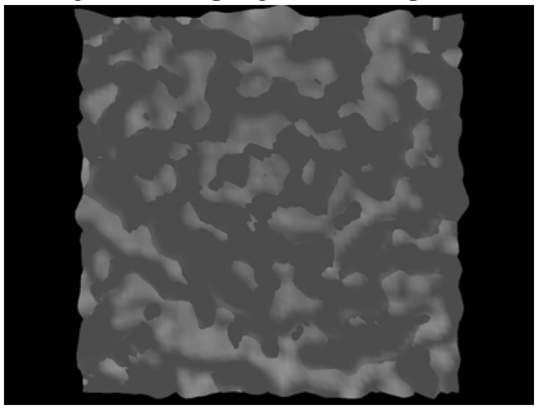
Much like this paper, our work also uses a simple CA algorithm as one of the steps of the system in order to generate the cave levels. However, this paper focus mostly on the real-time and performance aspect of the map generation (JOHNSON; YANNAKAKIS; TOGELIUS, 2010).

2.3 Procedural creation of 3D solution cave models

The focus of this article by Boggus e Crawfis (2009) is to use PCG algorithms to create 3D cave models based on real-world cave patterns. The authors present the usage of surface images of cave patterns and use them to create models of cave systems. The surface images can be generated, for example by utilizing fractal algorithms, or real world terrain data.

After the surface image is provided, the algorithm creates a heightmap out of it and then simulate the flow of water through this map, since this is ultimately the way solution caves, which are specific kinds of caves, are formed. Figure 2.2 shows a cross section view of a generated cave, where lighter areas represent walls (2009).

Figure 2.2 – Spongework cave pattern



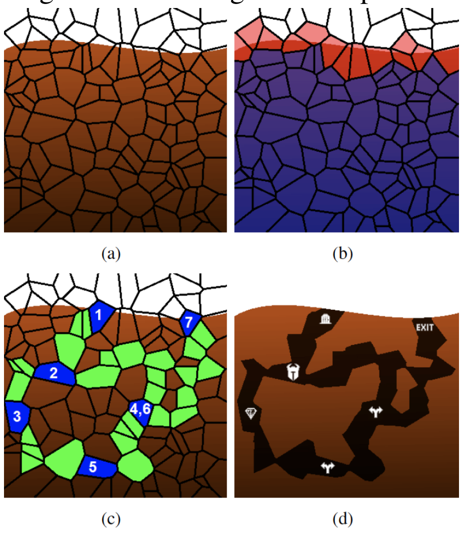
Source: (BOGGUS; CRAWFIS, 2009)

2.4 Procedural Playable Cave Systems based on Voronoi Diagram and Delaunay Triangulation

This paper by Santamaría-Ibirika *et al.* (2014) presents a new method for the generation of 2D or 3D playable cave systems. The user-defined input for their method are a list of Points Of Interest (POI) that will be set inside the cave and the relationship that these POI share. The paper defines a list of parameters that compose the POI like location, depth, number of branches, etc.

Figure 2.3 shows their cave generation process. In (a) a Voronoi diagram is generated on a terrain representation. (b) shows the classification of the Voronoi cells. (c) shows seven POI assigned to their respective cells in the diagram and (d) shows the final generated cave (2014).

Figure 2.3 – Cave generation process



Source: (SANTAMARÍA-IBIRIKA *et al.*, 2014)

2.5 Analysis and Development of a Game of the Roguelike Genre

The work of Gonçalves *et al.* (2015) describes the development of a prototype game of the roguelike genre. It also presents an analysis focused on the different techniques used on the AI for enemies and on the procedural generation of the dungeons. The compared PCG algorithms include: Kruskal and Prim, both being genetic algorithms; depth-first search, used to generate perfect mazes and cellular automata for cave-like structures. Some of the gathered results show that techniques of basic interaction iteration and BSP trees have good scalability while cellular automata and depth-first search need optimization or restriction in other to increase scalability.

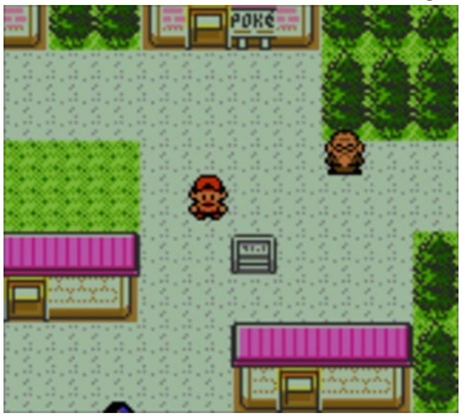
3 PROPOSAL

In this chapter we will define concepts that will be necessary to fully understand our proposal, which was developed in order to achieve the objectives presented in section 1.4. In the last section we will expand on the criteria for what makes a good video-game map.

3.1 Video-game maps

In the context of video games, a map corresponds to a game scenery inhabited by the player and other entities like monsters, NPCs, objects, etc (CARVALHO, 2011). The specific type of map that we have selected for this work is the 2D top-down type. This map perspective, also sometimes referred to as overhead view, bird's-eye view, Godview, etc. have a camera angle that shows the player and the area around them from above. An example of a game with an overhead view is shown in Figure 3.1.

Figure 3.1 – Screenshot of Pokémon Gold gameplay

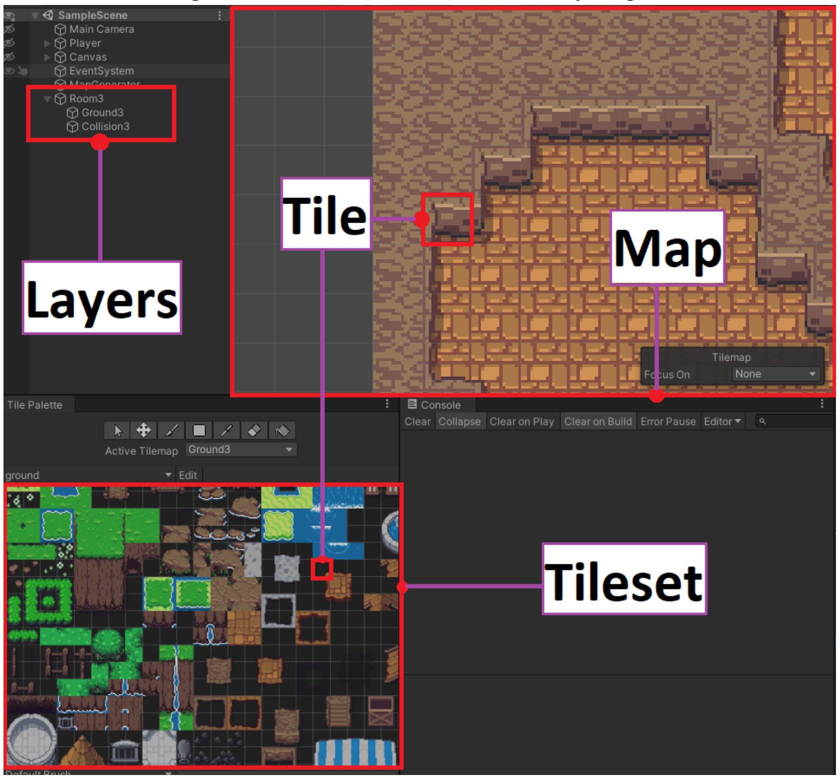


Source: (GAME FREAK, 1999)

3.1.1 Map elements

The tool chosen to develop our solution is the Unity game engine, which was released in 2005 and developed by Unity Technologies (UNITY TECHNOLOGIES, 2021). Game engines are softwares that provide the core functionalities needed for a game to run, for example a rendering engine, a physics engine, a collision engine, etc. Figure 3.2 shows a screenshot of this tool with some important elements that will be described on the next items of this section. The description of such elements have been adapted from Carvalho (2011).

Figure 3.2 – Screenshot of the Unity engine



Source: Image provided by author

3.1.1.1 Map

The map is represented here by a three-dimensional mesh of width w_{map} and height h_{map}, with $\{w_{map}, h_{map} \in \mathbb{N} | (w_{map} > 0) \wedge (h_{map} > 0)\}$. Additionally, this mesh has a depth of l_{map} layers, with $\{l_{map} \in \mathbb{N} | l_{map} \geq 1\}$, in our specific case, $l_{map} = 2$.

The map is composed by a number of $w_{map} \times h_{map}$ cells, in our case, each cell has one or two tiles (which will be explained in a subsequent section) one per layer, creating what is called a tiled map.

3.1.1.2 Layers

The layers are the different levels of the map where it's elements are distributed. In the scope of this work, there are two types of elements that will compose our maps, which are: ground and walls. The ground is the path where the player can walk on, while the walls are what define the structure of the cave maps. The player cannot walk on walls. In addition, these two elements are organized in two separate layers, here called ground layer and collision layer. The collision layer is named as such because, in order to avoid letting the player walk on the walls, a collider[1] component was added to this layer.

3.1.1.3 Tiles

A tile is the entity that occuppies the cells of the map mesh. Each tile is an image that represents some object to be inserted in the game, this image is of dimensions $(i * w_{cel}) \times (j * h_{cel})$ with w_{cel} and h_{cel} representing, respectively, the width and height of a cell in the map, while i and j are constants where $\{i, j \in \mathbb{N} | (1 \leq i < w_{map}) \wedge (1 \leq j < h_{map})\}$. In our specific case, $w_{cel} = h_{cel} = 16pixels$ and all tiles used have $i = j = 1$.

3.1.1.4 Tileset

A tileset is, as the name suggests, an image containing the set of tiles that will represent the game objects.

The Unity game engine divides any given tileset into a number of tiles that can be used by the developer to fill the map mesh and create an envinronment. The tileset used on this work is the "Zelda-like tileset", which was posted by ArMM1998 (2017) and is licensed in the public domain. Figure 3.3 shows this tileset, although only the tiles needed for walls and ground were used in this work, which will be shown in the next chapter.

3.2 Cave patterns

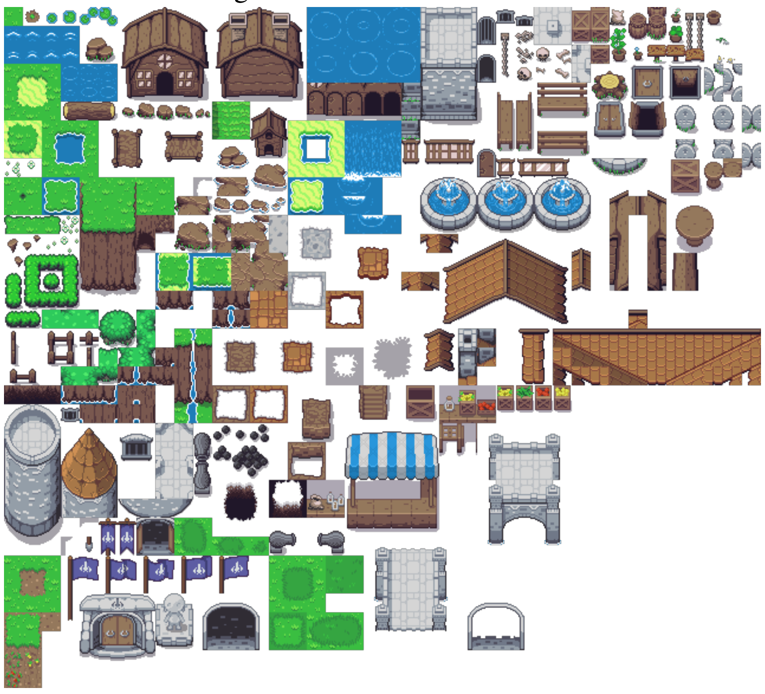
As will be discussed in the next section, one of the characteristics that make a good map is for it to have a natural look. Since we focused on cave-like maps, we researched for what would be understood as a natural cave representation in a 2D top-down view.

According to Audra e Palmer (2011), caves are formed by having different types of sources of water interacting with different kinds of soil. Figure 3.4 show some of the most common cave patterns.

It is outside of the scope of this work to give a larger description to each of the different cave patterns shown here, however it is important to note that this is an accurate representation of real-world caves and that they were used as a reference for the generated maps.

[1]In the Unity engine, the collider component is added to GameObjects where there's need to simulate physical collision.

Figure 3.3 – Zelda-like tileset



Source: (ARMM1998, 2017)

3.3 What makes a good generated map

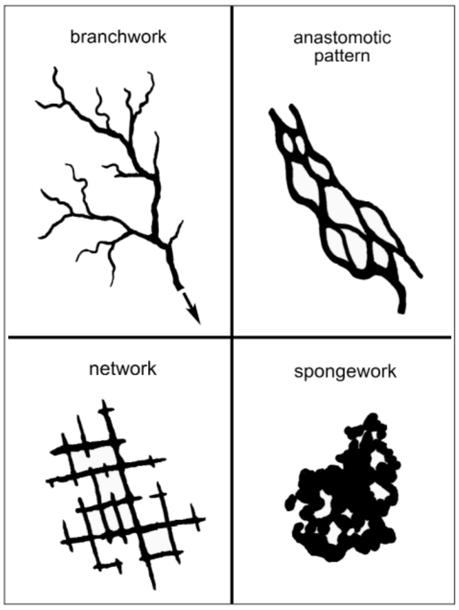
At last, an important part of our proposal is to generate maps that have a set of qualities that make them be perceived as good or desirable maps by players.

During the bibliographic research for these criteria, most of the findings related not only to the map but to the entire game level entirely, for example in Preuss, Liapis e Togelius (2014) the placement of enemies and treasures are also taken into consideration.

However, there were some criteria found on such works that made reference only to the map itself. Here is a list of the qualities of a good generated map, according to our bibliographic research:

- The generated maps should have a natural look, they shouldn't look as if they were generated by an algorithm (SHAKER; TOGELIUS; NELSON, 2016) (ADAMS; MENDLER, 2002).
- Dungeon maps should have a point of entrance, a point of exit (which might be the same as the point of entrance), and a path between them (SHAKER; TOGELIUS; NELSON, 2016).
- The Generated maps should look different from each other, to provide the player with an unique experience each time (GONÇALVES *et al.*, 2015) (ZAPATA, 2014).

Figure 3.4 – Common cave patterns



Source: (AUDRA; PALMER, 2011)

- The generated maps should encourage players to explore them (PING; DINGLI, 2020).
- Borders are necessary on the generated maps, in order to prevent players and monsters from escaping them (RODRIGUEZ TORRADO *et al.*, 2020).

To evaluate if the generated maps have the qualities listed here, we created a survey based on the one found on Carvalho (2016), which was used to evaluate players' motivation to explore game levels, developed based on the Instructional Materials Motivational Survey (IMMS), that uses the ARCS model by Keller (KELLER, 1987), (KELLER, 1993). Therefore, some of the elements for a good map that we will be evaluating are also adapted from the components of the ARCS model:

- **Attention**:
    - The design of the generated maps should be able to keep the player attention on the game.
    - The generated maps' design should be attractive to the players.
- **Relevance**:
    - The player should want to play a game that includes the generated maps.
- **Confidence**:
    - The generated maps should not appear to be too hard for the player.

    - When looking at the map, the player should be able to comprehend it easily.

- **Satisfaction**:

    - The player should want to explore the map.

More information about the survey will be given in Chapter 5.

4 DEVELOPMENT

On this chapter we will be explaining how the system was implemented, which algorithms that were used, the reason for choosing them, and what were the challenges faced during development.

The system was developed in the Unity Game Engine, utilizing the C# language. The basis for this work was started by following the video tutorial available at Taft (2019), about creating a 2D top-down game in Unity. This tutorial laid our foundation for important concepts, like importing the tileset, understanding how to draw a map, the basics of collision, etc.

It was after this point that the creation of the PCG system really started. To better understand its implementation, we will divide it in 7 steps and discuss each one in a different Section of this Chapter. The last Section will cover all of the user inputs that can be provided to the system in order to tune the map generation. However, first, there are still some information that are useful to understand all the steps from this point forward:

- The generated map will be represented by a global matrix of integers with width w_{map} and height h_{map}.
    - Its important to notice that w_{map} and h_{map} are user-defined parameters.
- Each individual position of this matrix will be referred to as a *cell*.
- A cell of value 1 represents a *wall*, a cell of value 0 represents the absence of a *wall*, or a path.
- The Figures where the map is shown are created by the use of Unity's Gizmo debugging tool. On these figures, the white color represents a *path* cell and the black color represents a *wall* cell.
- A group of one or more connected *path* cells is here defined as a room.

4.1 Create a randomly-filled base map

The first step taken was to have a randomly generated base map to start building upon. The generation of random or pseudo-random content is a part of most PCG techniques. In our case, the importance of the randomness is a reflection of one of the items for what is considered a good map, pointed on Section 3.3: "the Generated maps should look different from one another".

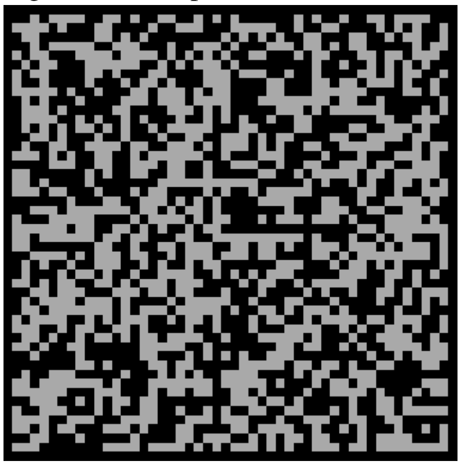
The creation of this base map is accomplished by the steps listed here in Algorithm 1.

Algorithm 1: Randomly filling of the map

```
int map[$w_{map}, h_{map}$]
Function FillMapRandomly():
    for all cells in the map do
        random = randomly generated number between 0 and 100
        if random < fillPercent then
            map cell = 1
        else
            map cell = 0
```

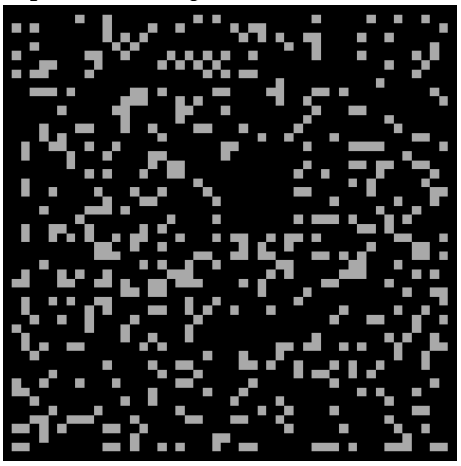
The variable $fillPercent$ specifies how much of the map will be populated by *walls* or *paths*, this variable is a user-defined parameter. The attribution of different values to it will generate different maps at the end of the process. Figure 4.1 shows an example of a base map with $w_{map} = h_{map} = 50$ and $fillPercent = 50$, while Figure 4.2 shows the same map but with $fillPercent = 80$.

Figure 4.1 – Map with fillPercent = 50.



Source: Image provided by author

Figure 4.2 – Map with fillPercent = 80.



Source: Image provided by author

The random number generator used to fill the map can also be controlled by a user-defined parameter. The user can choose to write a *seed* for the generator, which is a text that will tune the generator. Different executions of the generator with the same *seed* will always give the same results. In this case the generated map will always be the same for each given *seed*. Otherwise, the *seed* will be automatically chosen based on the execution time of the algorithm, making each generation different from the previous ones.

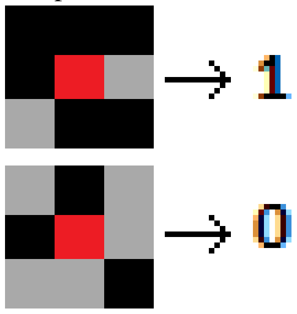
4.2 Cellular automata

A CA is a discrete model of computation that was first discovered by Stanislaw Ulam and John von Neumann; it can be described, in a simplified manner, as an n-dimmensional grid, with a set of states and a set of transition rules. The grid is composed by cells, each of them able to be in one of several states; in its most basic form, cells can be 1 (on) or 0 (off) (WOLFRAM, 1983).

Although CA has found application in many different areas of study, it was the 1970s Conway's Game of Life which brought its attention to beyond academia. This was a zero-person game where cells from a 2D grid would live or die (hence, they had only two states, 1 or 0) based on how many neighbors they had around them. This neighborhood is defined as the eight (8) cells surrounding the central cell currently being analyzed. Some of the rules from the Game of Life have been applied in dungeon generation because of the cave-like appearance that the algorithm creates on the grid (SHAKER; TOGELIUS; NELSON, 2016).

In our specific case, the rule used is that, if there are more than 4 *walls* around the current cell, then the cell becomes a *wall*, if there are less than 4 *walls* around the cell, it becomes a *path*. The method was also chosen because it generates images that are similar to the representation of *spongework* caves, shown in Figure 3.4. Figure 4.3 serves as a visual representation to this rule, where the red cell is the one under current analysis, while the number on the right represents is resulting state after the CA execution.

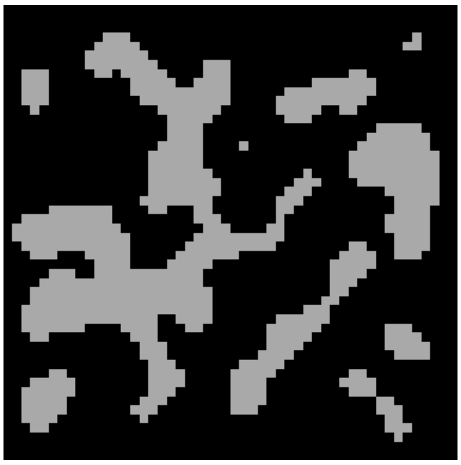
Figure 4.3 – Representation of the CA rule used.



Source: Image provided by author

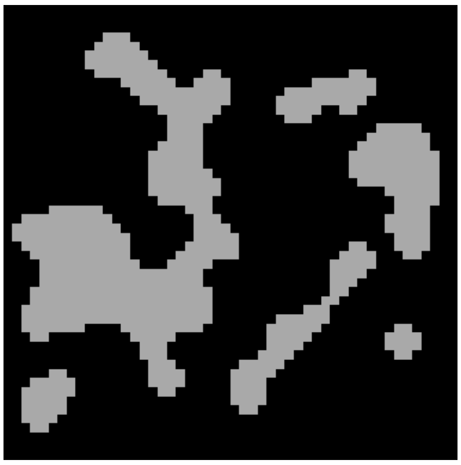
In Figure 4.4 we can see the base map generated in Figure 4.1 after 5 iterations of the CA. In addition, Figure 4.5 shows the map after 20 iterations. The number of iterations is a user-defined parameter for the system.

Figure 4.4 – Map after 5 iterations of CA.



Source: Image provided by author

Figure 4.5 – Map after 20 iterations of CA.



Source: Image provided by author

4.3 Creating a list of rooms

At this stage, the algorithm has finished letting us with one or more rooms scattered throughout the map. On code, we represent the rooms as a separate *Room* class that contains the coordinates of all of its cells. It is important to note that the *Room* class also contains other important information used by a following step of the algorithm (Section 4.5) defined by a flag indicating if the *Room* was visited, a list of *Rooms* connected to this *Room* and a method that connects two *Rooms*.

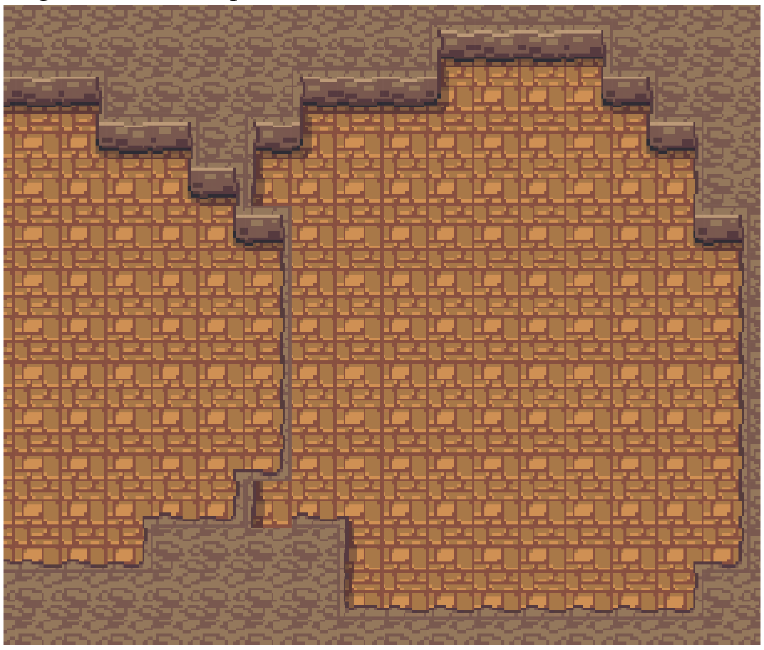
There are two sets of user-defined inputs that will determine if there will be the creation of entrance and exit rooms and their location on the map. After that is completed, the algorithm goes through the map grid and searches for *paths*. If a *path* was found we use the flood-fill algorithm in order to define where this room begins and where it ends, and then, we create a *Room* containing all the coordinates of its cells. After the creation of the *Room* it is added to a list containing all *Rooms* on the map.

4.4 Checking if the generated map is possible

One of the issues that came up during development is that not all generated maps can translate well to a 2D top-down tile-based map. This happens because, for a visually well-formed wall to be generated, there needs to have at least 2 *walls* between 2 *paths*. On that respect, Figure 4.6 exemplifies a malformed wall between 2 *Rooms*.

To solve this matter, in this step we go through each cell of each *Room* and check

Figure 4.6 – Example of a malformed wall between two *Rooms*



Source: Image provided by author

if there are *Rooms* that have a distance of less than 2 cells from each other. If there are, then the whole process is started again until the generated *Room* can be well translated to a tile-based map. Algorithm 2 describes this process.

Algorithm 2: Checking if the generated map is possible

```
while map is not possible do
    Randomly fill the map
    Run cellular automata
    Create a list of Rooms
    Check if the generated map is possible
```

4.5 Connection between rooms

To better explain how we create connections between separated rooms we will divide this section in two: ensuring connectivity through graph algorithms and drawing the connection between rooms using Bresenham's line algorithm.

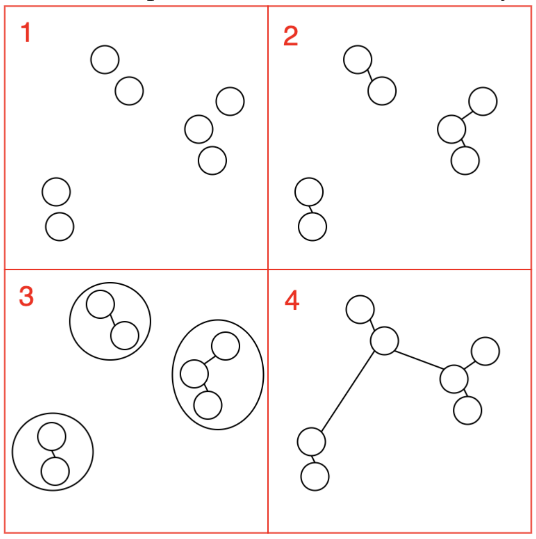
4.5.1 Ensuring connectivity

Ensuring the connectivity between all rooms is important for, as mentioned on Section 3.3: "Dungeon maps should have a point of entrance, a point of exit (which might be the same as the point of entrance) and a path between them". For our specific case, there is one more reason why this step is important: making maps that are similar to the *branchwork* or *network* cave patterns previously seen on Figure 3.4.

As the guideline above recalls, we need to make sure that every pair of *Rooms* are connected, i.e., there is a way to reach one *Room* by starting on any other *Room*. Each *Room* in the map can be viewed as a node in a graph, that way, we made use of a popular algorithm in graph theory to check if there's connectivity: Depth-First Search (DFS). DFS is an algorithm that starts at an arbitrary node of the graph and explores as far as possible along each branch before backtracking (EVEN, 2011).

Figure 4.7 shows a visual representation of the algorithm used in this step, which is described in Algorithm 3. Each node of the graph represents a *Room* in the map. In quadrant 1 we start with a map with fully disconnected *Rooms*. Then, the algorithm begins by connecting each node to its closest node, as seen in quadrant 2, this is done by checking the distance between each two border tiles between all *Rooms* and connecting the closest two; following that, it performs the DFS to check if connectivity is ensured; if not, it handles the sets of connected nodes a a single node, like is shown on quadrant 3, and then, redo the connectivity step, which leads to the result that can be seen in quadrant 4.

Figure 4.7 – Visual representation of the connectivity algorithm.



Source: Image provided by author

Algorithm 3: Ensuring connectivity of the map

```
Graph = represent Rooms as graph
while connectivity isn't reached do
    Connect closest Rooms in Graph
    Check connectivity
    Transform the sets of connected Rooms into new nodes for the new Graph
```

4.5.2 Drawing the connection

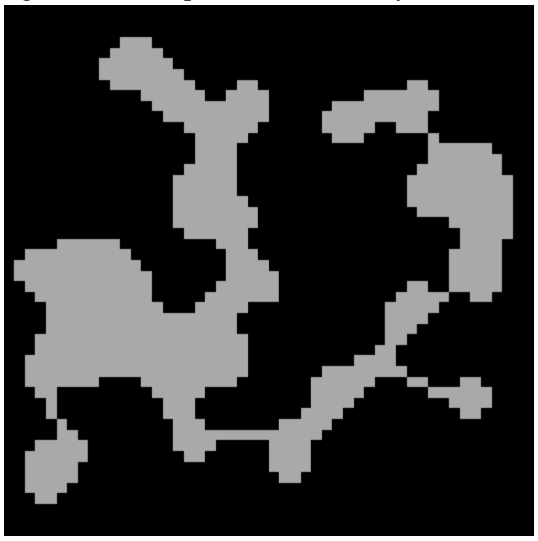
At this point of the algorithm, we have a map filled with *Rooms* that have ensured connectivity, nonetheless, this connectivity is only represented by some elements of the *Room* class, being not represented yet in the map grid as *walls* and *paths*.

This is where the algorithm known as Bresenham's line is used. This algorithm, suggested in 1962 by Jack Elton Bresenham, is used to determine which points on an n-dimensional raster should be selected in order to form a close approximation of a straight line (BRESENHAM, 1965).

Since each *Room* has all of its cell coordinates stored, we can choose the closest cells for each connection and then use the Bresenham's line algorithm to draw this straight line on the map. Figure 4.8 shows the same map seen in Figure 4.5 after the connectivity is ensured and then drawn on the map using Bresenham's algorithm.

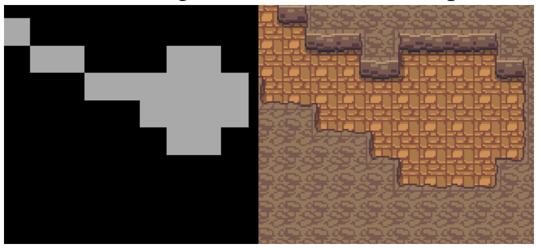
On a first glance, it may seem like these passages will not be wide enough to allow the player to pass through them, but the nature of the chosen tileset ensures that this is indeed possible, as shown in Figure 4.9.

Figure 4.8 – Map after connectivity is ensured.



Source: Image provided by author

Figure 4.9 – Passage before and after tile placement.



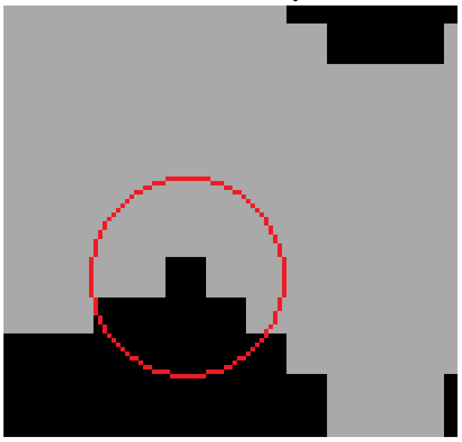
Source: Image provided by author

4.6 Removing inconsistencies from the map

Just like in Section 4.4, maps that are generated with single-spaced *walls* will not translate well to a 2D tile-map. Some of these problematic *walls* are occasionally created on the process of drawing the Bresenham's lines to connect separate *Rooms*.

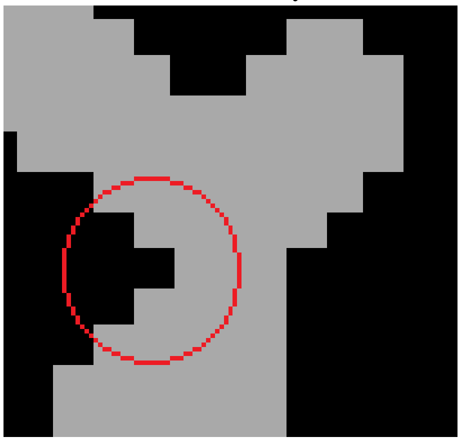
In this step we deal with these *inconsistencies* by going through the entire map grid and checking for *wall* cells that have either a *path* to its right and left, here shown on Figure 4.10; or to its up or down, as shown in Figure 4.11; the inconsistencies are circled in red. After these inconsistent single-spaced *walls* are found, they are replaced by *paths*.

Figure 4.10 – Example of a horizontal inconsistency.



Source: Image provided by author

Figure 4.11 – Example of a vertical inconsistency.

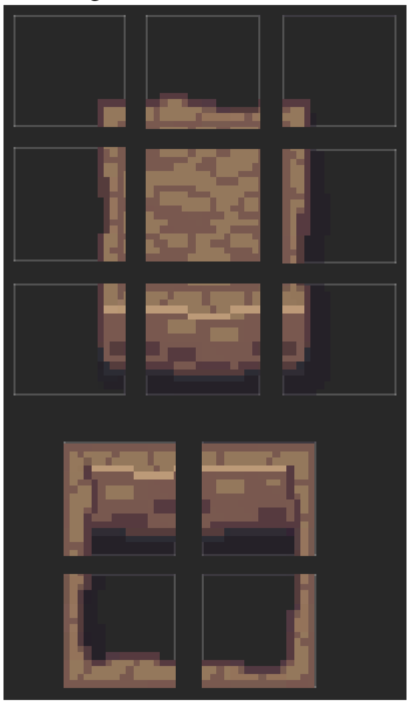


Source: Image provided by author

4.7 Tile placement

Finally, the last step of the generation is to place the tiles to their corresponding spots in the *walls* and *paths*. There are a number of 13 *wall* tiles needed to create the map, which are shown here on Figure 4.12. There is only a single tile that corresponds to a *path*, which is visible on the resulted map in Figure 4.14.

Figure 4.12 – Wall tiles.



Source: Image provided by author

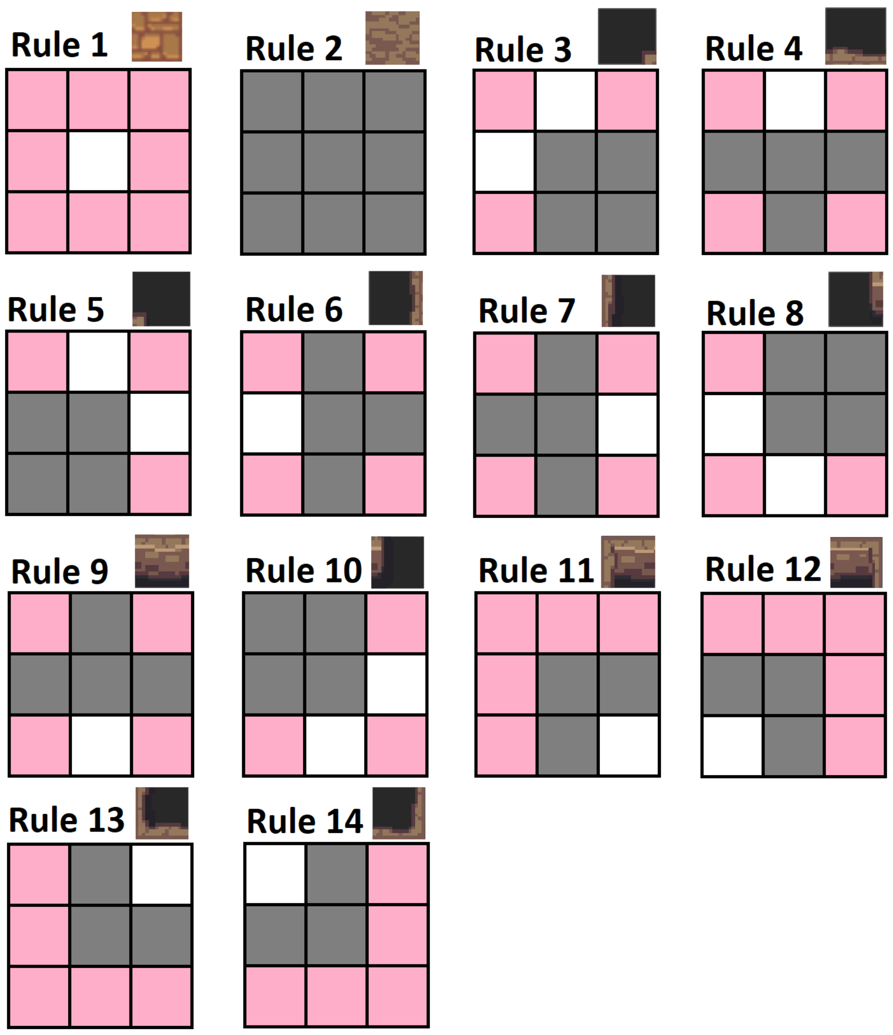
The first idea we had was to try to use a custom CA ruleset, where each cell had 14 possible states, one for each needed tile. However, after we assessed that a single execution of the ruleset through the map grid was enough to place all the tiles correctly, it was our understanding that our algorithm could be better described as a morphological operation.

This morphological operation, containing 14 rules, is here shown on Figure 4.13. The next item list serves to better understand the content of this Figure:

- Similar to Figure 4.3, the center cell is the one currently being analyzed.
- Gray cells represent *wall* cells.
- White cells represent *path* cells.
- Pink cells represent either a *wall* or a *path*, meaning that the content of this cell is not taken into account, i.e., it won't be analyzed by this rule.
- The tile next to the rule number is what the rule will return for the cell being analyzed.

For example, *Rule 3* will check if the current cell, the one at the center of the grid, is a *wall*, then if the cell above it is a *path*, then if the cell to the left of it is a *path*, then if the cell to the right of it is a *wall*, then if the cell below it is a *wall* and finally if the cell to the bottom right of it is a *wall*; if all of these conditions are met, the rule will return the corner tile shown on Figure 4.13 for *Rule 3* and place it on the analyzed cell.

Figure 4.13 – Ruleset.



Source: Image provided by author

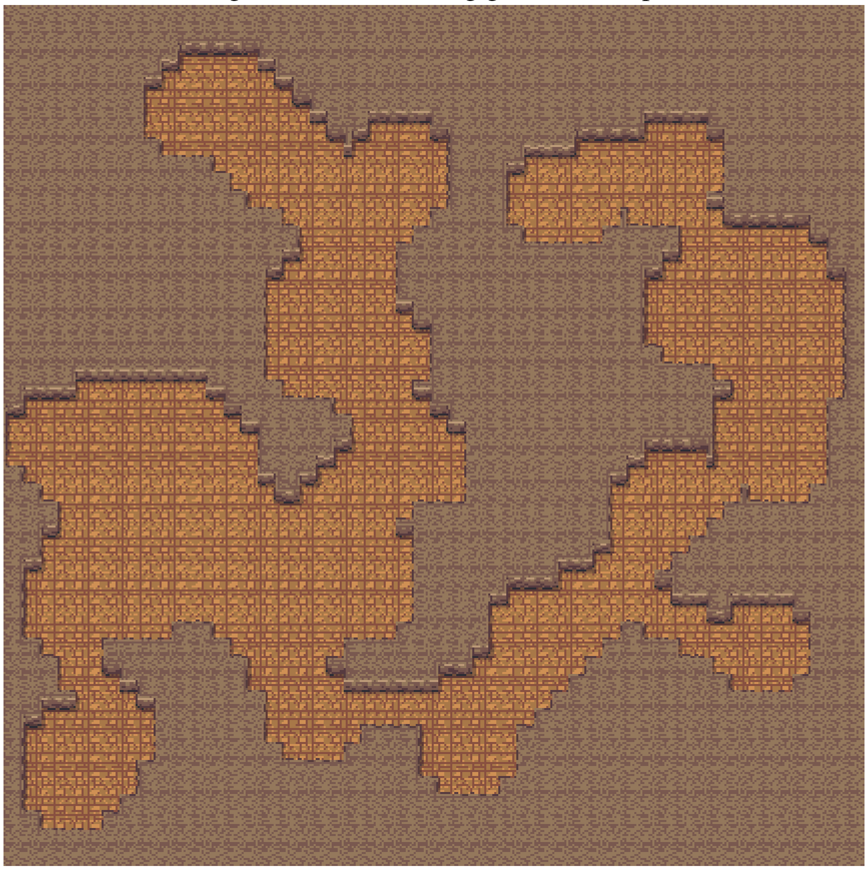
After the algorithm has gone through the whole map grid, the result will be a fully-formed 2D tile-based top-down map. Figure 4.14 shows the same map from Figure 4.8 after the tile placement.

4.8 User-defined parameters

On this section we will be showcasing all of the user-defined parameters that can be used to tune the generation of the map.

Figure 4.15 shows a snippet of the Unity tool, showing all the parameters used to generate the map that was seen on Figure 4.14: $w_{map} = h_{map} = 50$; the creation of a start and an end room was disabled, meaning that the cave used as an example was

Figure 4.14 – Resulting generated map.



Source: Image provided by author

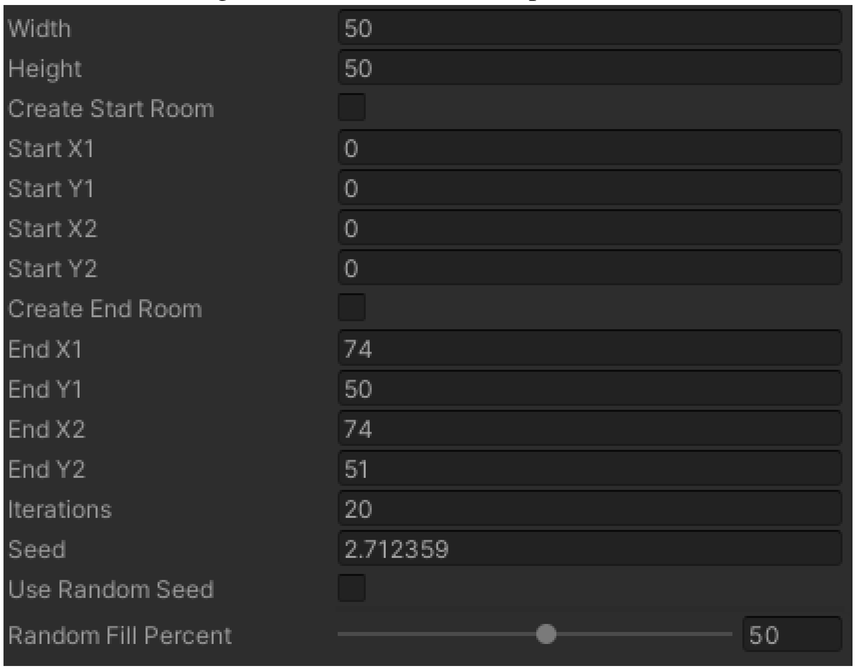
created without entrance and exit; the number of iterations for the CA step was set to 20; a specific *seed* was set; and the fill percent of the first step was set to 50.

The parameters can be explained as such:

- **Width**: the number of cells on a row of the generated map, or w_{map}.
- **Height**: the number of cells on a column of the generated map, or h_{map}.
- **Create Start/End Room**: toggles the creation of an entrance/exit for the generated cave map.
- **Start/End X/Y 1/2**: the coordinates of the 2 cells that will compose the entrance/exit of the cave map.
- **Iterations**: number of times that the CA algorithm will execute on the map.
- **Seed**: a text that will tune the random number generator, can be set to always generate the same map.
- **Use Random Seed**: this will define if the Seed from the above parameter will be used or if the system will choose a random one.
- **Random Fill Percent**: define the value of the variable $fillPercent$, explained on

Section 4.1.

Figure 4.15 – Available user parameters.



Width	50
Height	50
Create Start Room	☐
Start X1	0
Start Y1	0
Start X2	0
Start Y2	0
Create End Room	☐
End X1	74
End Y1	50
End X2	74
End Y2	51
Iterations	20
Seed	2.712359
Use Random Seed	☐
Random Fill Percent	50

Source: Image provided by author

5 METHODOLOGY OF VALIDATION

This chapter will discuss the creation of the questionnaire that was applied to video-game players, the conduction of its pretest, leading to its revision and the application of its final version. After that, we will discuss the results obtained.

As mentioned on Section 3.3, a survey was developed in order to test if a number of our generated maps can achieve the criteria we defined for what is considered a good generated map, which was built by adapting the measured tool of the ARCS model available at Carvalho (2016), originally used to assess the motivation of players to play a game. Both the questionnaire and its pretesting version were made available through Google Forms in Brazilian Portuguese. They make use of the typical five-level Likert scale (LIKERT, 1985 - 1932):

- Strongly disagree.
- Disagree.
- Neither agree nor disagree.
- Agree
- Strongly agree.

Both questionnaires start with a question about the participant age group, and a question about how familiar they are with 2D top-down games. Also in both cases, participants were asked to carefully visualize eight different maps generated by our system. These images are available in Appendix A. The main concern during the generation of these maps was to create a variety of results by tuning the user-defined parameters of the system.

English versions of the questionnaires are available in Appendix B and Appendix C. The justification of each question is also presented. Details about both of them will be discussed in the next sections, as well as their results.

5.1 Pretest questionnaire

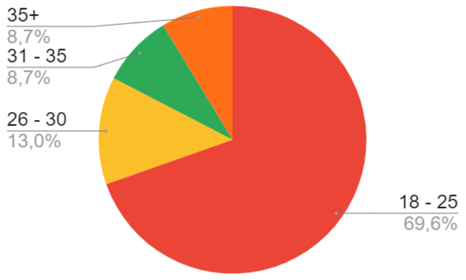
Before sharing a definitive version of the questionnaire, a pretest was made with a smaller group of participants, in order to determine the strengths and weaknesses of our survey concerning question format and wording.

The pretest had 23 participants. These were the first two questions, made with the purpose of understanding the profile of the respondents:

- What is your age group (in years)?
  - *(18-)* Less than 18.
  - *(18 - 25)* Between 18 and 25.
  - *(26 - 30)* Between 26 and 30.
  - *(31 - 35)* Between 31 and 35.
  - *(35+)* More than 35.
- How familiar are you with 2D top-down games? (games like Zelda, Pokémon, Bindings of Isaac, Hotline Miami etc)
  - *(1)* Never heard of it.
  - *(2)* Heard of it, but never played.
  - *(3)* I've seen videos of people playing this type of games.
  - *(4)* I've played at least one game in this style.
  - *(5)* I've played more than one game in this style.

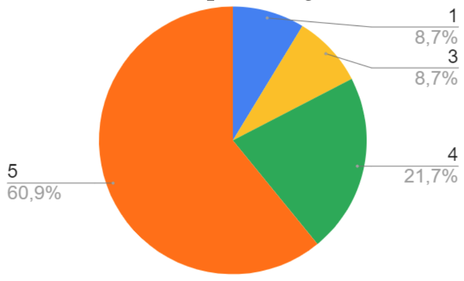
The results of these questions are shown here in Figure 5.1 and Figure 5.2. We had the participation of mostly young people who were already familiar with the concept.

Figure 5.1 – Age group of participants.



Source: Image provided by author

Figure 5.2 – Participants' familiarity with 2D top-down games.



Source: Image provided by author

After these initial questions, as mentioned before, participants were asked to carefully visualize the map images shown in Appendix A, then they responded the 15 questions regarding the map, available in Appendix B. Also, Appendix B contains the next part of the pretest, which involves answering five subjective questions about the questionnaire itself, shown in Table B.2.

5.1.1 Pretest results

For the pretest, the results we analyzed are the ones about the questionnaire itself, i.e., the ones in Table B.2. All of the points listed here have been taken into account when developing the second version.

5.1.1.1 Question 1

Question 1 reads: "Do you think that the questions from this questionnaire were easy to understand? If not, why?".

This question had 12 answers, from which 7 were positive, 3 were negative and 2 were neutral. The most important points we gathered from the neutral and negative answers were:

- Participants weren't sure what the map would be used for, this made it harder for them to answer the questions.
- Participants who were unfamiliar with 2D top-down games found it hard to visualize how the maps could be explored.

5.1.1.2 Question 2

Question 2 reads: "Are there repeated questions in this questionnaire? If yes, which?".

This question had 17 answers, from which 9 were negative and 8 were positive. All of the positive answers referred to questions that were opposites from one another, for example, question 9 (The generated map's design made it difficult for me to keep my attention.) and 10 (The generated cave maps were capable of capturing my attention.).

5.1.1.3 Question 3

Question 3 reads: "Would you include other questions on this questionnaire? If yes, which?".

This question had 16 answers, from which 7 were negative and 9 were positive. Most of these 9 positive answers suggested some type of comparison between the maps. From our understanding, adding questions that compare the generated maps is beyond the scope of this work, therefore we concluded that we need to better explain the objectives of the work before asking the questions.

One of the suggestions was to add a demonstration of how a character would view these maps, how far away would the camera be, how much of the map could be seen etc.

5.1.1.4 Question 4

Question 4 reads: "Would you change the text of one or more questions in this questionnaire? If yes, which?".

This question had 14 answers, from which 7 were negative and 7 were positive. The most important points gathered from the positive answers are listed below:

- Grammatical errors to be corrected.
- Use of a more direct language.
- Change questions 7 and 8 so that, instead of measuring the influence of colors and textures compared to the structure of paths, we can measure the influence of each separately.

5.1.1.5 Question 5

Question 5 reads: "Do you think a question comparing the generated maps to real cave representation is necessary for this questionnaire?".

This question had 19 answers, from which 13 were positive and 6 were negative. Some of the positive answers suggested showing some real cave patterns, arguing that it can be difficult to imagine how the structure of a cave can be represented in 2D.

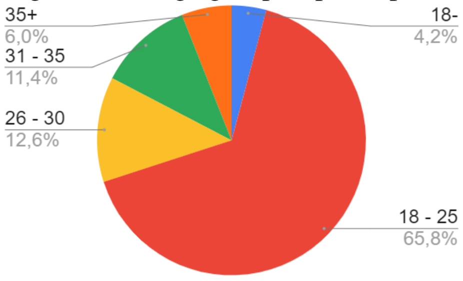
5.2 Second version of questionnaire

After reviewing all the suggestions and important points brought up on the pretest questionnaire, we developed an improved version of it, which can be seen on Table C.1 of Appendix C.

The second version had 163 participants, the first two questions were the same as the ones asked in the pretest and their results can be seen in Figure 5.3 and in Figure 5.4. We can see by these results that the participants' profile is the same as the one the pretest.

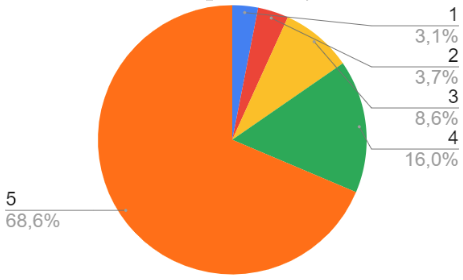
A new question was added with the intent of better identifying the profile of the participants: "Did you ever participate in the development of a video game?". For which

Figure 5.3 – Age group of participants.



Source: Image provided by author

Figure 5.4 – Participants' familiarity with 2D top-down games.

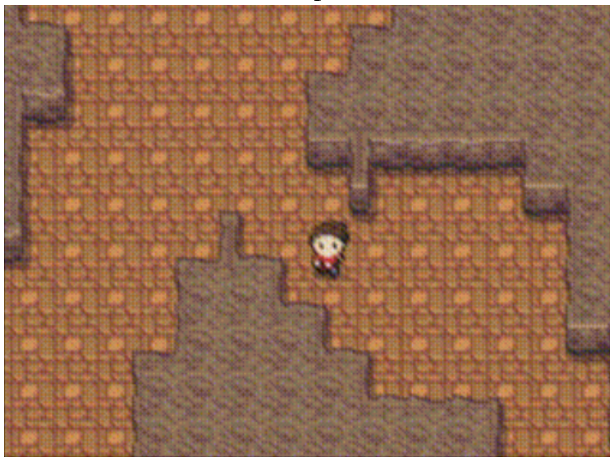


Source: Image provided by author

41,1% answered positively and 58,9% answered negatively. This is relevant since, as discussed on Section 1.2, one of the main objectives of PCG in video games is to assist in game design.

After these three initial questions, the participants were given a summary of the objectives of this work, along with the image from Figure 3.4, to assist in answering the questions to come. Additionally, an animated GIF of a character walking around one of the generated maps was added. A still image of this GIF can be seen here on Figure 5.5.

Figure 5.5 – Still frame of GIF shown to participants, showing a character walking on a generated map.



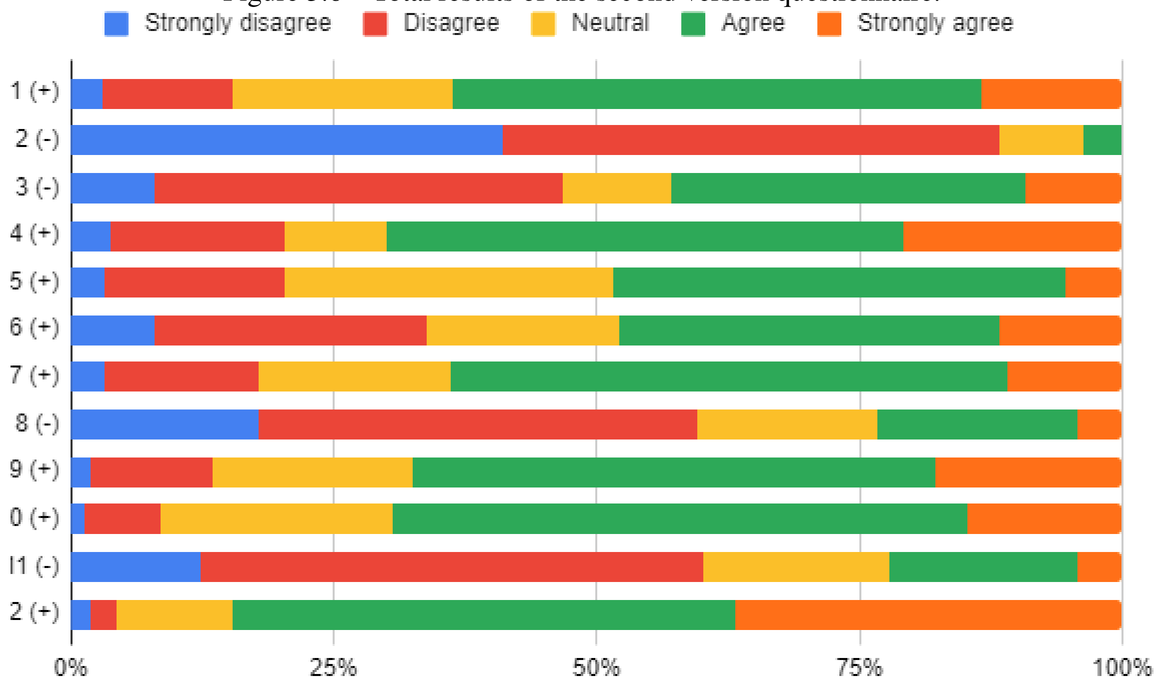
Source: Image provided by author

5.2.1 Analysis of results

The total results can be seen on Figure 5.6, where the questions that have the ideal response as *Strongly agree* are marked with a *(+)* after the question number, and the ones that have the ideal response as *Strongly disagree* are marked with a *(-)*.

To analyze the results, we assigned a weight to each option of the Likert scale,

Figure 5.6 – Total results of the second version questionnaire.



Source: Image provided by author

from 1 (strongly disagree) to 5 (strongly agree). The question's proposition was considered satisfied if the average of answers achieved 60% of the ideal value. For some questions the ideal is to reach 5, for others the ideal is to reach 1. Therefore, considering MAX as the maximum answer, MIN as the minimum answer, and V_{max} and V_{min} as the maximum and minimum satisfactory value, respectively:

$$V_{max} = (MAX - MIN) * 0.6 + MIN = (5 - 1) * 0.6 + 1 = 3.4$$
$$V_{min} = (MAX - MIN) * 0.4 + MIN = (5 - 1) * 0.4 + 1 = 2.6$$

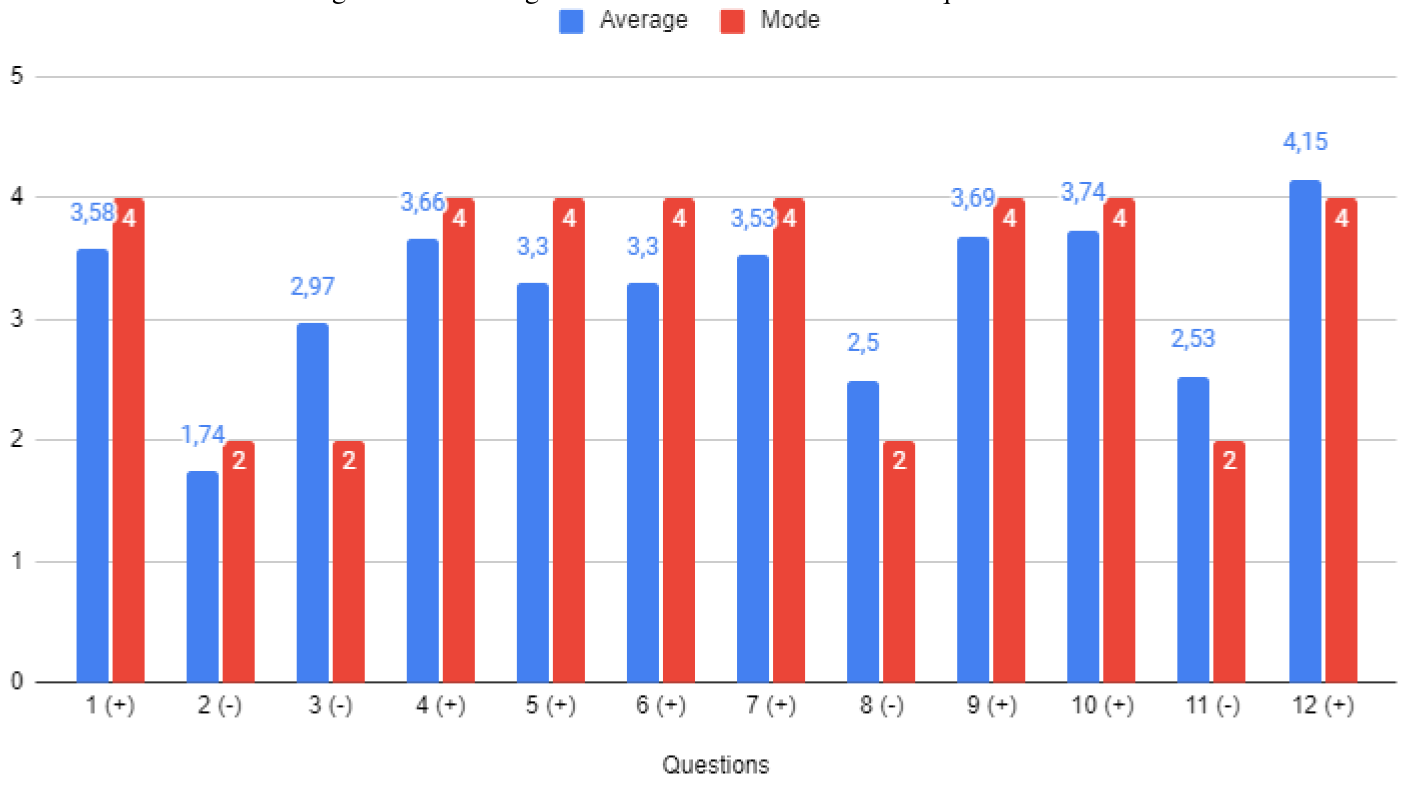
The average and mode of the answers is shown on Figure 5.7.

By these metrics; questions 1, 2, 4, 7, 8, 9, 10, 11 and 12 have reached a satisfactory result; while questions 3, 5 and 6 have not.

Out of the questions that have reached satisfactory results, the most important points gathered were:

- All of the questions related to the ARCS model for measuring motivation have reached satisfactory results.
  - Attention was shown to be the least influential component (questions 1 and 8).
  - Confidence and relevance were the major drivers for motivation (questions 2 and 12).
- The structure of the caves was shown to be slightly more important in achieving the

Figure 5.7 – Average and mode of the second version questionnaire.



Source: Image provided by author

natural look for the map (question 7).

- The variety criteria didn't reach a very expressive result, although still below the 40% mark (question 11).
- The criteria that proposed that the generated maps should encourage players to explore was reached by a good margin (question 9).

Out of the questions that haven't reached satisfactory results, the most important points gathered were:

- It would be easy to get lost on the generated maps (question 3).
  - This could be attributed to the labyrinthic nature of caves.
  - Generating larger maps tend to create many branching paths. This can mean that the system is more effective when generating smaller maps.
- Even though the results of question 4 suggests that the system can generate natural-looking structures, the result of question 5 suggests that the maps still don't quite look like real caves. Although the result was not drastically low.

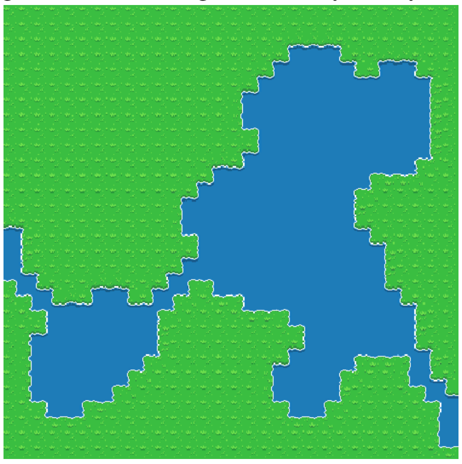
6 CONCLUSION

This work showed how PCG is important in game design and how the price of game development has been increasing throughout the years. To aid in that matter, it proposed the development of a PCG system to generate random cave-like maps that are similar to real-world caves. Finally it evaluated this system through a survey.

One of the biggest challenges faced was the research for criteria on what makes a good map, that can be seen on Section 3.3. Most of the related work we found focused on video-game *levels*, which also contain additional elements like enemies and treasures. Due to this difficulty we decided to also adopt the metrics found in Carvalho (2016) to evaluate players' motivation to explore game levels in educational games, which, in turn, is an adaptation of the ARCS model measurement tool found in Keller (1987). To meet our ends, the tool from Carvalho (2016) was adapted to verify players' perception on our auto-generated caves, which demanded a pre-test of the questionnaire to verify if it indeed attended our needs. In this sense, the evaluation method for generated maps proposed on this work can be further reused by others with little to no adaptation.

The creation of the system required knowledge from different areas and the use and adaptation of many algorithms. Our system has many user-defined parameters, which is a positive trait for map generation, since it allows for a larger variety of maps to be created. Even though the work focused on cave-like maps, by changing the used tiles the system can generate maps like forests, open fields, rivers, etc. Figure 6.1 shows a procedurally generated river, also created with the use of the system.

Figure 6.1 – River generated by the system.



Source: Image provided by author

The survey was answered by 163 participants, most of them from the age group that composes the average gamer and familiar to 2D top-down games. From the 12 questions, 9 achieved satisfactory results while 3 did not. Out of these results we found that, even though the structure of the maps resembles natural caves, the maps were not found to be much similar to 2D representations of real caves.

Finally, as a future works, the system could be turned into an Unity tool, that can then be used to create full games. The generation of rivers can be expanded on, by comparing them to real-life rivers and evaluating with a similar survey. The generation of maps can be changed to a level-generator, where the system wouldn't only generate the map but also place enemies, treasures, hidden passages etc. A deeper analysis of the questions regarding the ARCS model can be done to better evaluate the motivation to explore given to players by our maps.

REFERENCES

ADAMS, David; MENDLER, Michael. **Automatic Generation of Dungeons for Computer Games**. 2002. Bachelor Thesis – University of Sheffield.

ARMM1998. **Zelda-like tilesets and sprites**. 2017. Available from: `https://opengameart.org/content/zelda-like-tilesets-and-sprites`. Visited on: 6 May 2021.

AUDRA, Ph; PALMER, Arthur. The pattern of caves: Controls of epigenic speleogenesis. **Géomorphologie : relief, processus, environnement**, v. 17, p. 359–378, Dec. 2011. DOI: `10.4000/geomorphologie.9571`.

BOGGUS, Matt; CRAWFIS, Roger. Procedural creation of 3D solution cave models. **Proceedings of the IASTED International Conference on Modelling and Simulation**, Jan. 2009.

BRESENHAM, J. E. Algorithm for Computer Control of a Digital Plotter. **IBM Syst. J.**, IBM Corp., USA, v. 4, n. 1, p. 25–30, Mar. 1965. ISSN 0018-8670. DOI: `10.1147/sj.41.0025`. Available from: `https://doi.org/10.1147/sj.41.0025`.

BREWER, Nathan. **GOING ROGUE: A BRIEF HISTORY OF THE COMPUTERIZED DUNGEON CRAWL**. 2016. Available from: `https://insight.ieeeusa.org/articles/going-rogue-a-brief-history-of-the-computerized-dungeon-crawl/`. Visited on: 1 May 2021.

CARVALHO, Leonardo Filipe Batista Silva de. **Aplicação do modelo de Algoritmo Genético Baseado em Tipos Abstratos de Dados (GAADT) na adaptação de cenários bidimensionais de MMORPGs**. 2011. MA thesis – Universidade Federal de Alagoas.

CARVALHO, Leonardo Filipe Batista Silva de. **Explorando os Mitos Nacionais: contribuição ao aprendizado pelo estímulo à motivação a partir dos Serious Games**. 2016. PhD thesis – Universidade Federal do Rio Grande do Sul.

DOULL, Andrew. **The Death of the Level Designer: Procedural Content Generation in Games**. 2008. Available from: `http://roguelikedeveloper.blogspot.com.br/2008/01/death-of-level-designer-procedural.html`. Visited on: 1 May 2021.

ELECTRONIC ARTS. **Steam Page of Spore**. 2008. Available from: `https://store.steampowered.com/app/17390/SPORE/`. Visited on: 1 May 2021.

EPYX. **Steam Page of Rogue**. 1985. Available from: https://store.steampowered.com/app/1443430/Rogue/. Visited on: 1 May 2021.

EVEN, Shimon. **Graph Algorithms**. 2nd. USA: Cambridge University Press, 2011. ISBN 0521736536.

GAME FREAK. **Official Pokémon website**. 1999. Available from: https://www.pokemon.co.jp/game/other/gbc-gs/. Visited on: 5 May 2021.

GONÇALVES, C. *et al.* **Analysis and Development of a Game of the Roguelike Genre**. 2015. Trabalho de Conclusão de Curso – Universidade Federal do Rio Grande do Sul.

JOHNSON, Lawrence; YANNAKAKIS, Georgios; TOGELIUS, Julian. Cellular Automata for Real-Time Generation of Infinite Cave Levels. *In*. PROCEEDINGS of the 2010 Workshop on Procedural Content Generation in Games. Monterey, California: Association for Computing Machinery, 2010. (PCGames '10). ISBN 9781450300230. DOI: 10.1145/1814256.1814266. Available from: https://doi.org/10.1145/1814256.1814266.

KELLER, John. Development and Use of the ARCS Model of Motivational Design. **Journal of Instructional Development**, v. 10, Jan. 1987.

KELLER, John. Motivation by design. Unpublished manuscript, Florida State University, Florida. [*S. l.*], 1993.

KOUMARELAS, Robert. **Final Fantasy VII Remake Is Only Getting More Expensive**. 2021. Available from: https://www.cbr.com/final-fantasy-vii-remake-cost/. Visited on: 30 Apr. 2021.

LIKERT, Rensis. **A technique for the measurement of attitudes / by Rensis Likert.** New York: [s.n.], 1985 - 1932. (Archives of psychology ; no. 140).

MELAN. **Dungeon Mapping**. 2006. Available from: https://www.darkshire.net/jhkim/rpg/dnd/dungeonmaps.html. Visited on: 1 May 2021.

PING, Kuang; DINGLI, Luo. Conditional Convolutional Generative Adversarial Networks Based Interactive Procedural Game Map Generation. *In*. [*S. l.: s. n.*], Feb. 2020. p. 400–419. ISBN 978-3-030-39444-8. DOI: 10.1007/978-3-030-39445-5_30.

PONGMUSEUM. **Pong Museum - History**. 2021. Available from: http://pongmuseum.com/history/. Visited on: 30 Apr. 2021.

PREUSS, Mike; LIAPIS, Antonios; TOGELIUS, Julian. Searching for good and diverse game levels. *In*. 2014 IEEE Conference on Computational Intelligence and Games. [*S. l.: s. n.*], 2014. p. 1–8. DOI: 10.1109/CIG.2014.6932908.

RODRIGUEZ TORRADO, Ruben *et al.* Bootstrapping Conditional GANs for Video Game Level Generation. *In*. 2020 IEEE Conference on Games (CoG). [*S. l.: s. n.*], 2020. p. 41–48. DOI: 10.1109/CoG47356.2020.9231576.

SANTAMARÍA-IBIRIKA, Aitor *et al.* Procedural Playable Cave Systems Based on Voronoi Diagram and Delaunay Triangulation. *In*. 2014 International Conference on Cyberworlds. [*S. l.: s. n.*], 2014. p. 15–22. DOI: 10.1109/CW.2014.11.

SHAKER, Noor; TOGELIUS, Julian; NELSON, Mark J. **Procedural Content Generation in Games**. [*S. l.*]: Springer, 2016. ISBN 9783319427140.

SMITHERS, Pelham. **Peak Video Game? Top Analyst Sees Industry Slumping in 2019**. 2019. Available from: https://www.bloomberg.com/news/articles/2019-01-23/peak-video-game-top-analyst-sees-industry-slumping-in-2019/. Visited on: 30 Apr. 2021.

STEINBERG, Scott. **The definitive Guide: Videogame Marketing and PR (1st ed.)** [*S. l.*]: iUniverse, 2007. ISBN 9780595433711.

TAFT. **Make a game like the Legend of Zelda using Unity and C#**. 2019. Available from: https://www.youtube.com/watch?v=F5sMq8PrWuM&list=PL4vbr3u7UKWp0iM1WIfRjCDTI03u43Zfu. Visited on: 6 May 2021.

TOGELIUS, Julian *et al.* What is Procedural Content Generation? Mario on the borderline, Jan. 2011. DOI: 10.1145/2000919.2000922.

UNITY TECHNOLOGIES. **Unity home page**. 2021. Available from: https://unity.com/. Visited on: 6 May 2021.

VIANA, Breno M. F.; SANTOS, Selan R. dos. A Survey of Procedural Dungeon Generation. *In*. 2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). [*S. l.: s. n.*], 2019. p. 29–38. DOI: 10.1109/SBGames.2019.00015.

WITKOWSKI, Wallace. **Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic**. 2020. Available from:

`https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990/`. Visited on: 30 Apr. 2021.

WOLF, Mark J. P. **The Video Game Explosion: A History from PONG to PlayStation and Beyond**. [*S. l.*]: Greenwood, 2007. ISBN 9780313338687.

WOLFRAM, Stephen. Statistical mechanics of cellular automata. **Rev. Mod. Phys.**, American Physical Society, v. 55, p. 601–644, 3 July 1983. DOI: `10.1103/RevModPhys.55.601`. Available from: `https://link.aps.org/doi/10.1103/RevModPhys.55.601`.

WRIGHT, Will. **2007 TED video of Spore**. 2007. Available from: `https://web.archive.org/web/20070817073543/http://www.joystiq.com/2007/07/22/todays-most-delayed-and-ambitious-video-ted-spore-demo/`. Visited on: 1 May 2021.

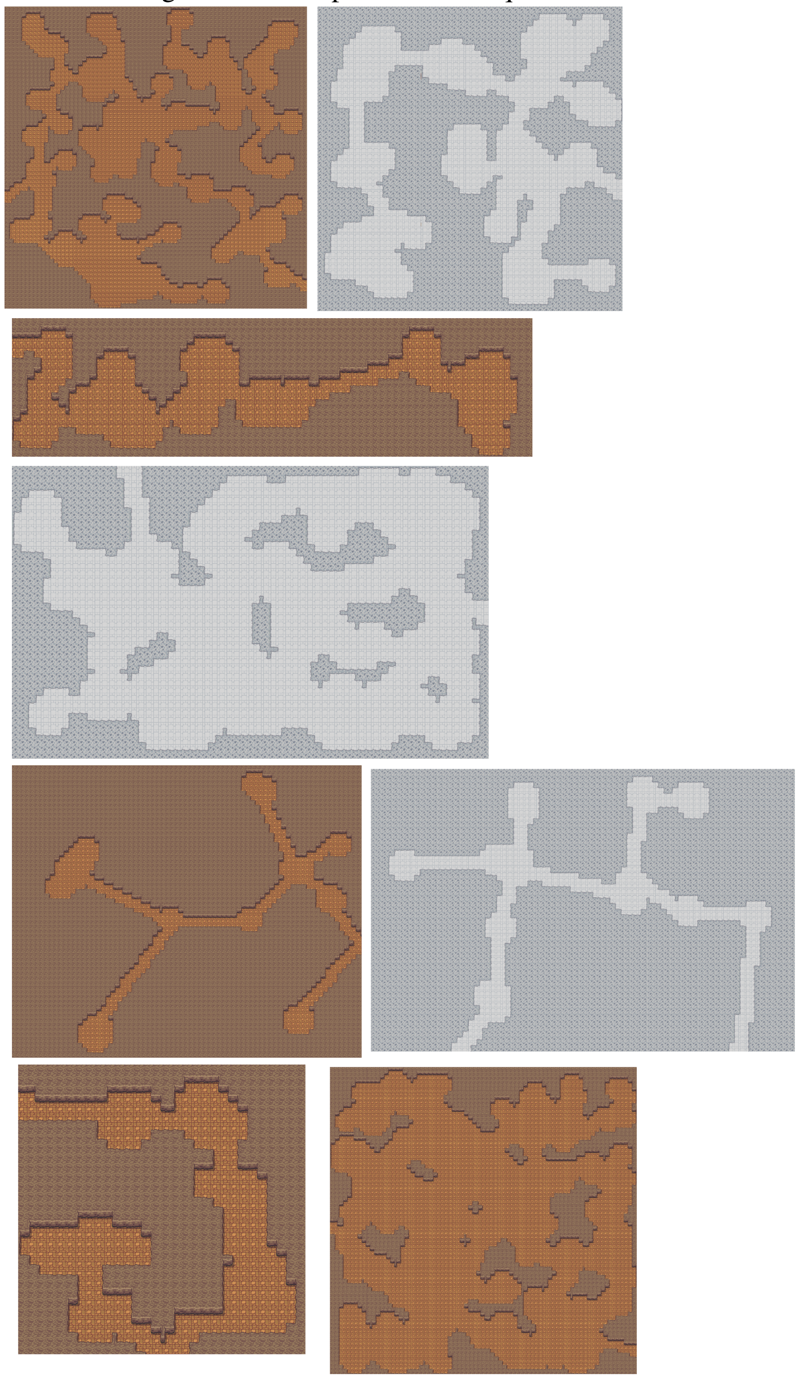
ZAPATA, Santiago. **A Classic Roguelike**. 2014. Available from: `https://blog.roguetemple.com/roguelike-definition/`. Visited on: 6 May 2021.

ZUBEK, Robert. **Elements of Game Design**. [*S. l.*]: The MIT Press, 2020. ISBN 9780262043915.

l;

APPENDIX A — MAP FIGURES USED IN BOTH VERSIONS OF THE QUESTIONNAIRE

Figure A.1 – Maps used in the questionnaires



Source: Image provided by author

APPENDIX B — PRETEST QUESTIONNAIRE

Table B.1 – Part of the pretest questionnaire regarding the maps.

No.	Question	Justification
1	There was something interesting in the cave maps that caught my attention.	Question related to the Attention component of the ARCS model.
2	The first time I saw the cave maps, I had the impression that they would be easy to explore.	Question related to the Confidence component of the ARCS model.
3	The structure of the caves was harder to comprehend than I would have liked them to be.	Question related to the Confidence component of the ARCS model.
4	While looking at the maps, I feel that it would be easy for me to get lost.	Question related to the criteria: maps should have an entrance, an exit, and a path between them.
5	The structure of the map looked natural, it didn't look like it was generated by an algorithm.	Question related to the criteria: generated maps should have a natural look.
6	While I observed the maps, I felt like I was seeing a representation of real caves.	Question related to the criteria: generated maps should have a natural look.
7	In the generated maps, what reminds me of a real cave are the used colors, and not the structure of paths.	This question serves to evaluate how important the structure is in the creation of a natural look.
8	In the generated maps, what reminds me of a real cave is the structure of paths, and not the used colors.	This question serves to evaluate how important the used colors are in the creation of a natural look.
9	The generated map's design made it difficult for me to keep my attention.	Question related to the Attention component of the ARCS model.
10	The generated cave maps were capable of capturing my attention.	Question related to the Attention component of the ARCS model.
11	While looking at the maps, I feel like exploring them.	Question related to the criteria: maps should encourage players to explore.
12	The design of the generated cave maps is attractive.	This question serves to evaluate the overall attractiveness of the maps.
13	The design of the maps is very simple and not attractive.	This question serves to evaluate the overall attractiveness of the design.
14	The generated structures look very similar, with little variety.	Question related to the criteria: maps should look different from one another
15	The variety of the maps helps to keep my attention.	Question related to the criteria: maps should look different from one another.

Source: Table provided by author.

Table B.2 – Part of the pretest questionnaire regarding the questionnaire itself.

No.	Question
1	Do you think that the questions from this questionnaire were easy to understand? If not, why?
2	Are there repeated questions in this questionnaire? If yes, which?
3	Would you include other questions on this questionnaire? If yes, which?
4	Would you change the text of one or more questions in this questionnaire? If yes, which?
5	Do you think a question comparing the generated maps to real cave representation is necessary for this questionnaire?

Source: Table provided by author.

APPENDIX C — SECOND VERSION OF THE QUESTIONNAIRE

Table C.1 – Second version of the questionnaire.

No.	Question	Justification
1	There was something interesting in the cave maps that caught my attention.	Question related to the Attention component of the ARCS model.
2	The structure of the caves was harder to comprehend than I would have liked them to be.	Question related to the Confidence component of the ARCS model.
3	I feel like it would be easy for me to get lost in more than one of the observed maps.	Question related to the criteria: maps should have an entrance, an exit, and a path between them.
4	In more than one of the generated maps, the structure of the map looked natural, it didn't look like it was generated by an algorithm.	Question related to the criteria: generated maps should have a natural look.
5	While I observed the maps, I felt like I was seeing a representation of real caves.	Question related to the criteria: generated maps should have a natural look.
6	In the generated maps, what reminds me of a real cave are the used colors and textures.	This question serves to evaluate how important the used colors and textures are in the creation of a natural look.
7	In the generated maps, what reminds me of a real cave is the structure of paths.	This question serves to evaluate how important the structure of paths are in the creation of a natural look.
8	The design of more than one map made it difficult for me to keep my attention.	Question related to the Attention component of the ARCS model.
9	While looking at the maps, I feel like exploring them.	Question related to the criteria: maps should encourage players to explore.
10	The design of the generated cave maps is attractive.	Question related to the Satisfaction component of the ARCS model.
11	The generated structures look very similar, with little variety.	Question related to the criteria: maps should look different from one another
12	I would play a game that utilizes the generated maps.	Question related to the Relevance component of the ARCS model.

Source: Table provided by author.