Procedural generation applied to a video game level design.

Albert Carrión Díaz

Director: Antonio Chica Calaf
Specialization: Computing

FIB UPC - Barcelona School of Informatics

Spring 2015

Acknowledgements

My first words of gratitude are for Antonio Chica, the director and supervisor of this project. He had the answers every time that I needed guidance and he basically made possible this project helping me transform my original idea into this Bachelor's Thesis.

Secondly, I would like to thank everyone who supported me to finish my degree, from family and friends to even teachers and classmates.

Finally, I have to thank the ten testers who helped me with the validation process and were kind enough to do more than I asked:

- David Bigas
- Adrià Urgell
- Aitor Castella
- David Peribañez
- Sergio Cecilia
- Borja Herrerias
- Miguel Rodilla
- Joaquim Casals
- Daniel Garcia
- Álvaro Bru

Thank you all!

Abstract (English)

The objective of this project was to create a 2D side-view platformer video game with a huge focus on creating the levels with procedural generation instead of making them by hand. By doing so, this project approaches a creative yet technical discipline like level design in video games from an algorithmic approach.

This has resulted in a system that is able to automatically generate levels that can be used by a game. These levels are as fun as levels created by level designers hands, and we can generate an infinite amount of different levels by changing the various parameters that change the generation process.

The generation of levels takes into account: their feasibility, levels must have some way to be completed; their difficulty, the challenge that the levels represent to the user; and the game rules, the resulting designs must work with the gameplay, physics,...

Besides the level generating system, the project has resulted in a video game that uses this generator and lets you play on those levels.

Abstract (Catalan)

L'objectiu d'aquest projecte era crear un vídeo joc de plataformes 2D de visió lateral donant molta importància a la creació dels nivells amb generació per procediments en lloc de fer-los a mà. Fent això, el projecte aborda una disciplina creativa però tècnica com el disseny de nivells als vídeo jocs des d'un punt de vista algorísmic.

Això ha resultat en un sistema capaç de generar automàticament nivells que poden ser utilitzats per un joc. Aquests nivells son tan divertits com els creats a mà per un dissenyador de nivells i podem generar un nombre infinit de nivells diferents només variant els diversos paràmetres que canvien el procés de generació.

La generació de nivells té en compte: la seva viabilitat, els nivells han de poder ser completats d'alguna manera; la seva dificultat, el repte que els nivells suposen per l'usuari; i les regles del joc, els nivells resultants han de funcionar amb el gameplay, físiques,...

Apart del sistema generador de nivells, el projecte ha resultat en un vídeo joc que utilitza aquest generador i permet jugar a aquests nivells.

Abstract (Spanish)

El objetivo de este proyecto era crear un video juego de plataformas 2D de visión lateral dándole mucha importancia a la creación de niveles con generación por procedimientos en lugar de hacerlos a mano. Haciendo esto, el proyecto aborda una disciplina creativa pero técnica como el diseño de niveles en los video juegos des de un punto de vista algorítmico.

Esto ha resultado en un sistema capaz de generar automáticamente niveles que pueden ser utilizados por un juego. Estos niveles son tan divertidos como los creados a mano por un diseñador de niveles y podemos generar un numero infinito de niveles diferentes solo variando los diversos parámetros que cambian el proceso de generación.

La generación de niveles tiene en cuenta: su viabilidad, los niveles tienen que poder ser completados de alguna manera; su dificultad, el reto que los niveles suponen para el usuario; y las reglas del juego, los niveles resultantes deben funcionar con el gameplay, físicas,...

Aparte del sistema generador de niveles, el proyecto ha resultado en un video juego que utiliza este generador y permite jugar a los niveles generados.

Contents

List of Figures

List of Tables

1. Introduction

This document is going to show all the investigation, planning and work done on this Bachelor's Thesis. To do this, first we will explain the problem we are trying to solve and everything we planned for the solution, from the objectives and way of working to the context and state of the art of the project, in this introduction.
Then, we will show all the project management that includes the temporal planning and the budget and sustainability estimations.
Later, we will disclose the technical details of the work done in different chapters.
Finally, we will explain the validation process and conclusions. Additionally, there will be some annexes more directly related to the development that will help understand the effort behind this project.

1.1 Problem

There is a well-recognized problem in the video games development field concerning the constant increase on the number of assets a modern game needs. Each year, new video games have better graphics, bigger maps, and, in general, more content that needs artists or designers who make them. This big demand of assets problem has been solved in two ways: making international companies like *Ubisoft* [20] which can afford huge developers teams with lots of artists and designers, or, what is called procedural generation, programming algorithms to do this job.

This project will focus on using the latter on the level design of a video game, which would obviously dismiss any need of level designers for developing the game. However, procedural generation is not a technique that can be used as a general solution for all the cases, and needs to choose a specific problem to fully adapt the algorithm to create something as a real designer could do. Given this condition, we decided to create a 2D side-view platformer game that welcomes players to fully explore the levels to give more importance to the level design we will be working on.

We will explain in detail how we planned to solve this problem (including an scope, temporal and budget planning) and how we finally tackled the problem in this document.

1.2 Scope of the project

This section is going to describe how the problem we have introduced in the previous section is going to be tackled. Then, the scope and possibilities of the project are going to be defined more clearly. What is more, the methodology behind all the work is going to be explained, and monitoring and validation tools or methods are going to be disclosed too. After that, problems and risks of the project are going to be described.

1.2.1 Solution and objectives

As we said in the problem introduction, the project will be developing a 2D side-view platformer video game. And in this game we will use knowledge from the degree together with investigation we will do to achieve an innovative automatic level design. The idea to choose this project was inspired by now popular games like *Rogue Legacy* [1], *Spelunky* [11] and *Cloudberry Kingdom* [12]

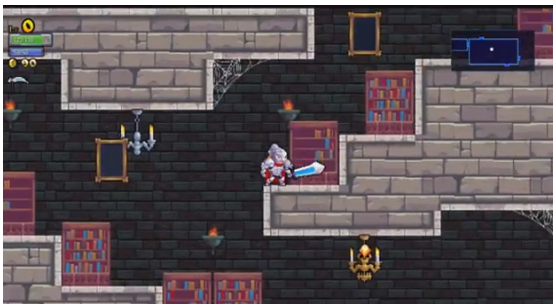


Figure 1.1: *Rogue Legacy*, inspiration and example of 2D side-view platformer.[1]

Therefore, the objectives for this project are creating a system that creates levels for this kind of game, and then creating a video game that works with those procedurally generated levels. The video game will be developed to show that levels , which will be a result of the investigation, are fun. We will define fun as the validation factor of the quality of the project and the levels taking into account: their feasibility, levels must have some way to be completed; their difficulty, the challenge that levels represent to the user; and the game rules, resulting designs must work with the gameplay, physics,...

1.2.2 Scope definition

Defining a scope for a video game can always be a difficult because you can always add more features easily, but taking into account the limited time and big focus that will go into level design, a decision has been made. This decision, that will have to adapt to reality as the project advances, consists on making a system that creates maps made of rooms for the game, another one that designs the interior of rooms and finally building a game that works around those maps and rooms that will make possible to check the quality of those levels for players.

1.2.2.1 The map

Creating the map will be translated into creating a graph where each node will be a room and arcs between them represent adjacency, and those nodes will contain all the

information needed for creating the inside room such as if there is a key inside or if any special power is needed to beat the room interior. The creation of this map will let the game designer (in this case me) see how the resulting map will be with some parameters. For example the depth of paths in the map or the shape of the map should be something possible to control.

1.2.2.2 The rooms

The creation of the interior of rooms will be a more difficult task where, in a similar fashion to the map, a graph will be made. In this case though, each node will be the platforms where the player's character can stand and arcs between them will represent if they are reachable from one to the other. This graph should let the game designer (me again) control with some parameters things like the difficulty to reach the platforms or the abilities needed to do so.

1.2.2.3 The game

The video game will try to be as simple as possible, almost as if it is just to let players try the work made on level design. However, this part will still be a really big portion of the project because it represents obtaining (or creating) a lot of art assets, programming physics and gameplay, ... Even if this represents an even bigger effort, the plan is to make the game have the basics controls for a 2D game of its genre (moving and jumping) and extra mechanics (double jump, dropping,...), enemies, items, etc. that will make the game closer to a real commercial product. This closeness to a market product will help in evaluating the level design.
In addition, the target platform will be Windows PCs as it is the easiest option too.

1.2.3 Methodology and validation

Some agile software development method will be used to adapt the work load this project represents to limited time and the type of project. This is also an usual approach to video game development because it is difficult to totally define the final game before starting the development, and it is easy to have the need to evolve the requirements and solutions of the problem through the development.
Using this type of methodology will help too with the fact that parts of the project will implement algorithms that we are currently investigating or we do not fully know yet, and these algorithms may work differently than what we expect. With the obvious changes to which the problem should adapt.
Being more specific we will use an adaptation of the *scrum methodology*[21] to my working conditions. This will translate into the director of the project being the product owner and me being the whole development team and my own scrum master. Moreover, sprints (or iterations) will be of minimum a week and maximum of three weeks to let me combine the work load of the project with the work load of other university duties like exams. Meetings at the start and end of each sprint will be mainly done virtually and my director and I will meet face-to-face when needed (when deep subjects or difficult problems as the scope definition arise). Therefore, the daily scrum meeting will just be a ten minutes thinking on my previous work and keeping up a little written discussion with my log and it will be done every two days instead of one.

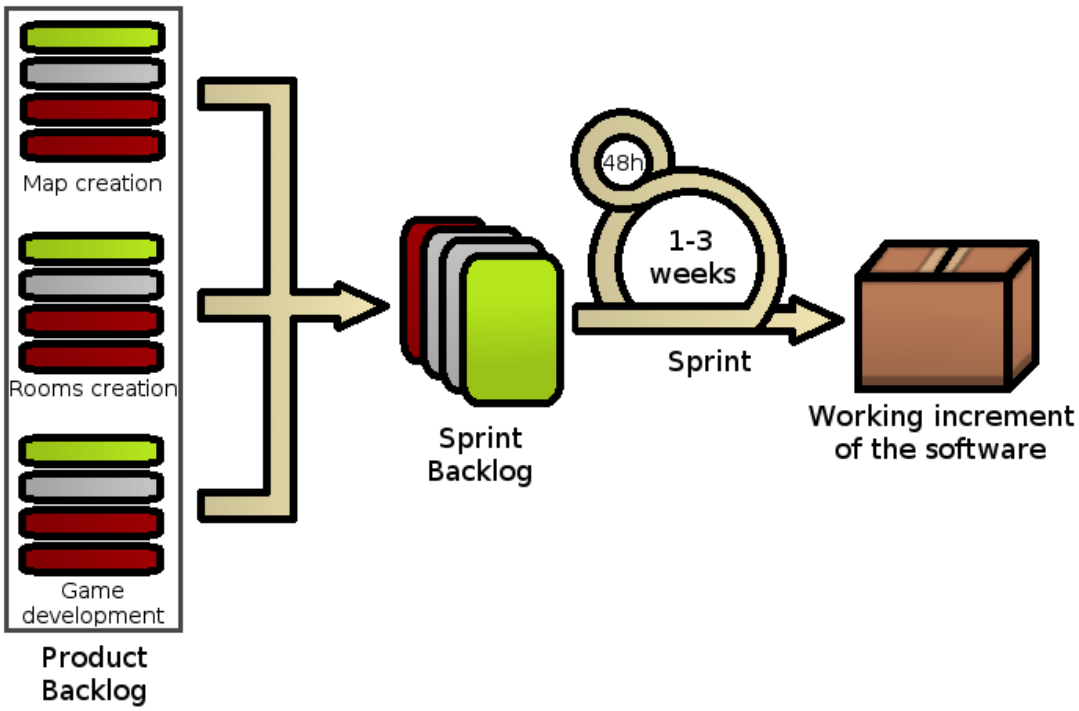


Figure 1.2: Diagram of my adaptation of the scrum methodology. [2]

Furthermore, as explained in the scope section, the project has three big parts that will be developed overlapping and that will work independently at the first sprints. And because evaluating of the quality of the level design can only be validated by humans, as long as the map creation, the rooms creation and the video game are not working together, the validation of each part will be done by us and the director of the project. Once the three parts start working together, a little group of around ten people will be gathered to work as testers of the project and will give simple feedback to confirm that the work done is correct.
The mechanism for receiving feedback will be separated into two parts. First, objective feedback that will be data recorded from testers playing the project, and second, subjective feedback that will be obtained with some little surveys to the testers after they have played with the project.

1.2.4 Development tools

The main tools for the development of the project are going to be *Unity 3D* [3], which is a great game engine for quick development, and *Mono Develop* [4], that is the default IDE for programming with C# (the chosen language from the ones compatible with *Unity 3D*).
Git [5] and *Bitbucket* [6] are going to be the tools used for version control and tracking the development.
Many other tools, such as *Gimp* [7] and *Audacity* [8] for preparing the graphics and audio for the game, will be used but in comparison to the ones already shown they will play a relatively small part in the project development.

Figure 1.3: Tools logos. [3] [4] [5] [6] [7] [8]

1.2.5 Obstacles and risks

First, the main risk in the planning of this project is the really tight schedule that will result of such an ambitious project. The obvious consequence is then that big setbacks that slow down our work could prove to be fatal to the quality of the final result. Specially to the video game wrapping around the level design which will be the last part to be completed and which would be more affected by a cut in its development time, probably resulting in a variation of what the validation tests show. For instance, the level design could be pretty solid but the limitations on the game to test it would occlude the way to seeing it.
Another big risk are errors in the code or the implementation, not detected because the small number of testers and the fact that randomness can play an important part in the procedural generation of the levels, which may lead to unexpected and incorrect outcomes in the final delivery of the project.

In a smaller magnitude we find risks like implementing algorithms or systems too slow for a video game which is a type of application really critical with the efficiency. For example, the generation of the rooms of the map should not interfere with the game being able to produce more than 30 frames per second.
One more risk is not being able to find an ideal solution, as part of the level design evaluation is subjective and it may happen that during the development the implementation does not go into the right direction to make levels liked by players (or in our case testers).

Lastly, other minor risks may be having problems to find enough testers for the validation of the results or not being able to find good graphics or audio for the video game. However, those aren't big problems because for the former the search for testers will be for a very reduced number of people, and for the latter it can be solved by using really simple art created by us with the minimum requirements to let the players focus on the levels.

1.3 Context and state of the art

This section is going to describe the areas of interest the project is about, which actors are going to be affected by the development and the state of the art of the fields the project is going to touch. This will result in a general idea of what are some similar commercial projects made in the past and what we can expect this project to add to the current knowledge.

1.3.1 Context

This project has three big areas of interest that are not totally independent between them. First, we have the level design field, which studies and tries to improve the structure, organization and content of the levels to make challenges for the player. The second area we have is procedural generation and more specifically procedural generation of levels, which attempts to create algorithms and methods that automatically create the levels instead of designing them by hand. Third, we have the video games that use procedural generation, being the ones that use procedural generation of levels to create experiences that do not repeat the main focus because they are the most similar to this project.

Let's see in detail the three areas in the next lines.

Level Design. As the title of the thesis says we are going to apply procedural generation to level design, so before we can do it, we need to have a good understanding on what level design is and how level design works on video games.
Level design is the discipline of games development that takes care of creating the scenarios, environments and stages of the game. This means creating where the action of the game is happening and it usually includes deciding where the challenges (puzzles, enemies, etc) await the player. This discipline is a mix of artistic (as it creates a great part of what the player is going to see) and technical (besides looking good the levels must work with the rest of the game, for instance the physics of the game) processes.

The level design of a game, as explained by professional experts and communities (such ones as the references authors [22], books [23] [24] [25] [26] and websites [27] [28]), is closely related to its parent discipline, game design, and consists of creating and balancing the challenges and experiences of the player with the game's world. This is usually done with few metrics and a lot of playing experience of the levels (we can see this explained by authors of games with an important level design like *Super Meat Boy* [29]). It is also work frequently done by the level designer with some tools (usually software) specially created for the game.

However, relying in the subjectivity of a designer and having to create some tools for him are exactly what we want to avoid in this project as we want to make the level design an automatic process.

Automatic level generation. As a solution or alternative to having a level designer, game developers have created many methods to have a lot of stages generated using random or pseudo-random generators and lists of rules to control this generation. As

there is a enormous number of kind of games and consequently a lot of kinds of levels (for instance, levels for a side-view platformer with double jump and levels for a top-down-view hack and slash are really different), there are many methods that adapt to the different solutions we may be looking for. We decided to study the automatic level generators that work with maps and 2D platformers stages, and to create a mix that brings something new to this area of interest. For this purpose, we must know a lot of examples of other projects that have tried this, and we must take the ideas that can be useful and simplify them for the mix.
This need of ideas is what brings us to the next area of interest.

Video games using the previous areas of interest. As stated before, not only seeing this area of interest shows us the context of the project, but it is also an important part of the research to decide what are some good ideas to develop in our project. Therefore, let's take a look into the games we found most relevant that use procedural generation into the creation of their levels:

Dwarf Fortress [9] is a game where almost everything (including the map, enemies and even legends for example) is created by procedural generation. Everything is on a grid and then represented by text, giving us the idea that making the representation simple and the content very variable is liked by many players.

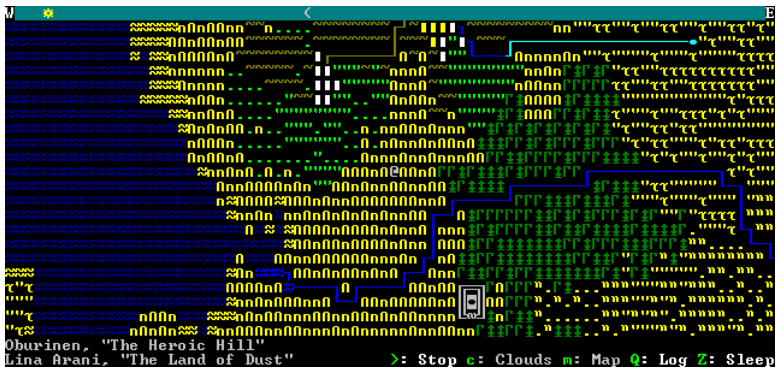


Figure 1.4: *Dwarf Fortress* map part example. [9]

The Binding of Isaac [10] is an RPG (*Role Playing Game*) roguelike game where the hero has to explore dungeons of rooms with challenges. The map and order of the rooms is procedurally generated, but the rooms are only selected from a long list of available ones depending on some conditions like the position in the map. The main idea we got from this game is the importance of having an objective in the map and how to decorate the rooms that are not indispensable.

Figure 1.5: *The Binding of Isaac* map example. [10]

Rogue Legacy [1] is a 2D side-view platformer based on the exploration of a castle. As in *The Binding of Isaac*, the map is made of rooms, but in this case the rooms can be of different sizes and are combinations of pre-made elements. This game gave us the idea of letting the map take more freedom in its construction while it does not break the rules we create, and it is the most similar to the project expected outcome.

Figure 1.6: *Rogue Legacy* map example. [1]

Spelunky [11] is a 2D side-view platformer. This game has no map as there are only different stages which can be considered big rooms. Those rooms are totally procedurally generated with almost none pre-made patterns. We decided to learn how to divide the room generation into smaller problems from this game.

Figure 1.7: *Spelunky* room example. [11]

Cloudberry Kingdom [12] is a 2D side-view platformer. In this game, like in Spelunky, there are only rooms and no map. The interesting part of the rooms in this game is that not only they are procedurally generated, they can also adapt from a lot of conditions such as if the player can jump, run or many other abilities and parameters by having a system that checks if the room is playable. This idea of room adaptation is what we mostly got from this game for our project.

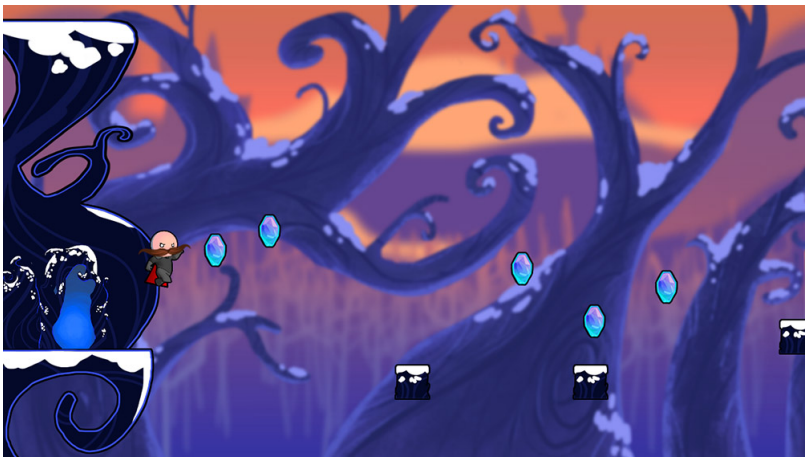


Figure 1.8: *Cloudberry Kingdom* room example. [12]

Many others games could be looked at, but these five are the most influential in the direction of the project and are, most of them, currently well-known (although we used images from old versions instead of the famous current remakes of some of them).

1.3.2 Actors

The development of the project involves the following described actors.

Project developer. I am the only developer of this project. This implies I am going to take care of the planning, the documentation, the design, the coding and the validation process of the project. As the only developer I will also be responsible of any marketing, advertising and distribution of the result of the project after its development.

Project director. Antonio Chica Calaf, Assistant Professor from the Computer Science department of Universitat Politècnica de Catalunya, is the director of this project. He is guiding me, the project developer, whenever I have doubts or difficulties with the project.

Testers. These will be a little number of people who will play the video game resulting from the project. Their playtime will result in some objective data (like finishing time and some other metrics), and they will also answer a survey for some subjective answers about the game. The objective and subjective results will be used to validate the quality of the outcome of the project.

Users. The aim of the project is to create a game that can be played by anyone and that they have fun with it. Therefore, the users will be any player who decides to play

the video game resulting from the project, and they will be the only beneficiaries of the project if they have fun with it.

1.3.3 State of the art

In this section, we are going to explain the state of the art of the main parts of the project development. The parts are the three we have been talking about in the scope and what mainly define the temporal planning of the project, which are the following: map generation, level generation and video game development.
With these three parts we have to take a look into the following tools and methods for getting an idea of the state of the art of our project. Most of the more technical concepts will be referenced, not exclusively, from the *Procedural Content Generation Wiki* [30] for a more in detail easy, yet technical, explanation.

1.3.3.1 Map generators

Map generation and specially dungeon generation (a labyrinth made of different rooms connected) is one of the most used parts of procedural generation in video games, and it is also a frequent topic for investigation [31] [32] [33]. The most common ways nowadays of obtaining maps with procedural generation are:

Simple maze generation algorithms based on grids. These are the most simple algorithms that having a graph with equal nodes representing a grid, can create a maze [34] that explores all the nodes and connects them if there is a path from one to the other. Some of these algorithms are: *the recursive backtracker*, *Eller's algorithm*, *Kruskal's algorithm*, *Prim's algorithm*, *the recursive division algorithm*, *Aldous-Broder algorithm*, *The Hunt-and-Kill algorithm*, *the Growing Tree algorithm*, *the Binary Tree algorithm*, *the Sidewinder algorithm* and many more. This great diversity of solutions plays with the length, shape and number of the corridors of the outcome while balancing the implementation difficulty, the efficiency and the comprehension of the algorithm. All the mentioned algorithms can be found over different references but are really well explained in detail in *Jamis Buck's blog* [13].

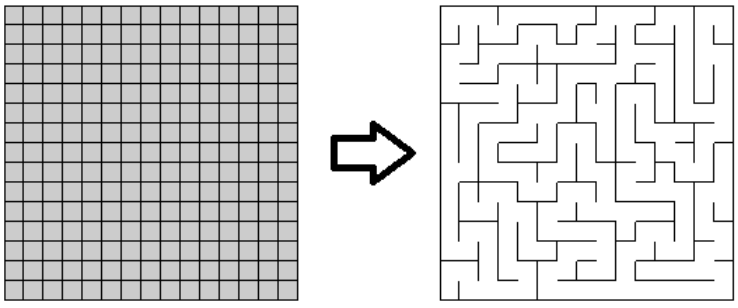


Figure 1.9: Example of one of the algorithms. Initial graph and *the recursive backtracker* outcome. [13]

Cellular automata. In an attempt to create organic looking maps, this method works with the same grid as the previous algorithms but from another point of view. Every node knows some rules and has a state and depending on the rules and the states of its neighbor nodes it changes its own state. There are many ways to apply this method as

we can see in the references [35] but the most common use in dungeon generation can be seen in this video from *Nathan Williams* [36].

Binary Space Partition. This method is similar to *the recursive division algorithm* but it does not work with a grid so it gives some more freedom to the outcome. We can see this in another video of *Nathan Williams* [37].

Using a Delaunay Triangulation. Taking advantage of some random generation of the rooms and then working from there with a *Delaunay Triangulation* to create the corridors and a nice map is the main idea of this method. We can clearly see how it works in yet another video of *Nathan Williams* [38] or the video game *TinyKeep* [39].

Genetic algorithms, Generative Grammars, Occupancy Regulated Extension, Constraint-Based and others. As we can see in the references there are many more options but they are farther from the direction the map generator of the project is expected to take. For example, genetic algorithms lack control over some aspects such as the resulting shape, which made us discard them as an option.

Combinations of different map generators. At the end, each method has its advantages so really good map generators mix different methods and let you select what you need each time you use them. A great example of this kind of generator is the *Pro-D Total* [14] map generator, an already made tool that lets developers generate maps. Moreover, with the long development of *Pro-D Total*, the generator has even included AI (*Artificial Intelligence*) methods such as path-finding in the maps it generates.

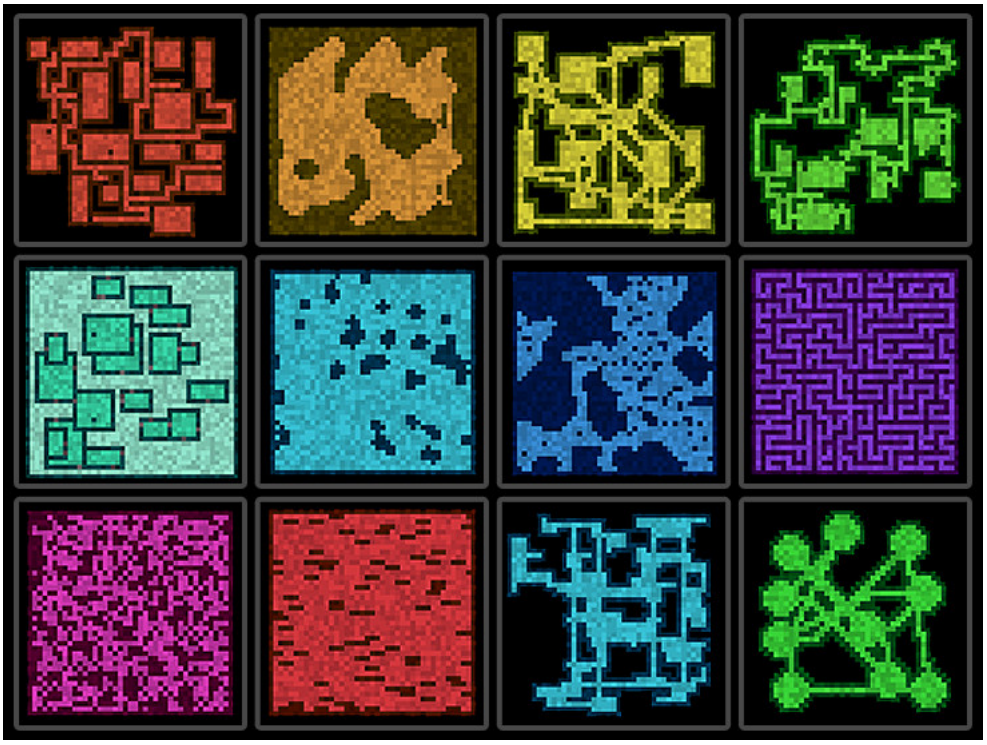


Figure 1.10: *Pro-D* outcome examples.[14]

1.3.3.2 Level generators

Level generation is a much more difficult area, and it is less common to find information of how developers have used it [40] or as an investigation topic [41]. Some common ways (although most of them have been mainly used for generating levels for games like *Super Mario Bros* [42] and not rooms like the ones we expect to develop in the project) to solve this problem are:

Combining pre-made parts. This is the simplest and most used method to generate levels. We just have some pre-made parts we know work with the game and repeat them in a more or less complex way. We can do from the most basic repetition where the patterns are obvious to the most complex random selection that is controlled by some rules. One of the ideas we liked the most in this kind of method is the concept of *Wang tiles* to hide the patterns of using pre-made parts. The easiest to understand basic explanation of this concept is at *Wikipedia* [15] and there are even variants of it [43]. Moreover, there are many other methods, such as *Occupancy-Regulated Extension* [44], to combine pre-made chunks in more complex ways.

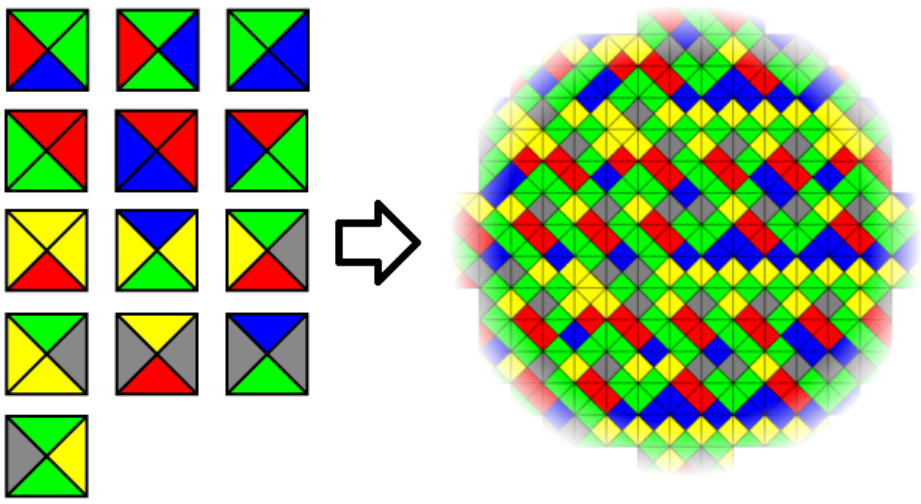


Figure 1.11: *Wang tiles* application example. [15]

Rhythm based. Some levels generators, such as *Launchpad* [45], play with what they call the rhythm of a level to divide the generation problem into smaller pieces. These smaller pieces are usually solved by using pre-made parts (smaller than the ones used by *Occupancy-Regulated Extension* for example, but still pre-made) but could be solved with smaller generators too. The division of the problem is then the important part of this method, and works like the division of a music piece that is clearly marked by some rhythm. But instead of defining the rhythm of a song with beats we define the rhythm of a level with the player actions (for example jumps) following the rules and metrics we have implemented. This a really interesting group of solutions for this level generation problem, but it works best in long levels and not in small rooms like the ones in our project.

AI based. The most complicated method is based on the idea of creating some kind of AI that can play the level you randomly generate and that gives feedback. This feedback may include if the level can be completed, if it is difficult, if it is long, etc. It is the most complex and least efficient, but it is possible the most versatile method we have seen. Furthermore, used with iterations of random generation and other variations, it can almost guarantee no repetition and there are no observable patterns on the outcome. However, there is few detailed documentation on how this can be implemented, as each level generator that uses this must be developed with the final game in mind. Looking at this article [40] from the *Cloudberry Kingdom* lead programmer can give us a quick view of his method.

1.3.3.3 Game engines

The main tool to create a video game is a game engine and there are two options, creating your own engine for the video game you want to make or using a general purpose game engine already made. As the main objective of the project is to focus on the procedural generation in the level design, a general purpose game engine is going to be used for developing the video game and saving time. Let's see three of the most popular game engines that have free versions:

Game Maker: Studio. [16] This engine is specialized in 2D games development and includes many tools to simplify the quick development of games (like *Spelunky*) and prototypes. This simplification may be a problem for developing or for guaranteeing efficiency of some complex ideas.

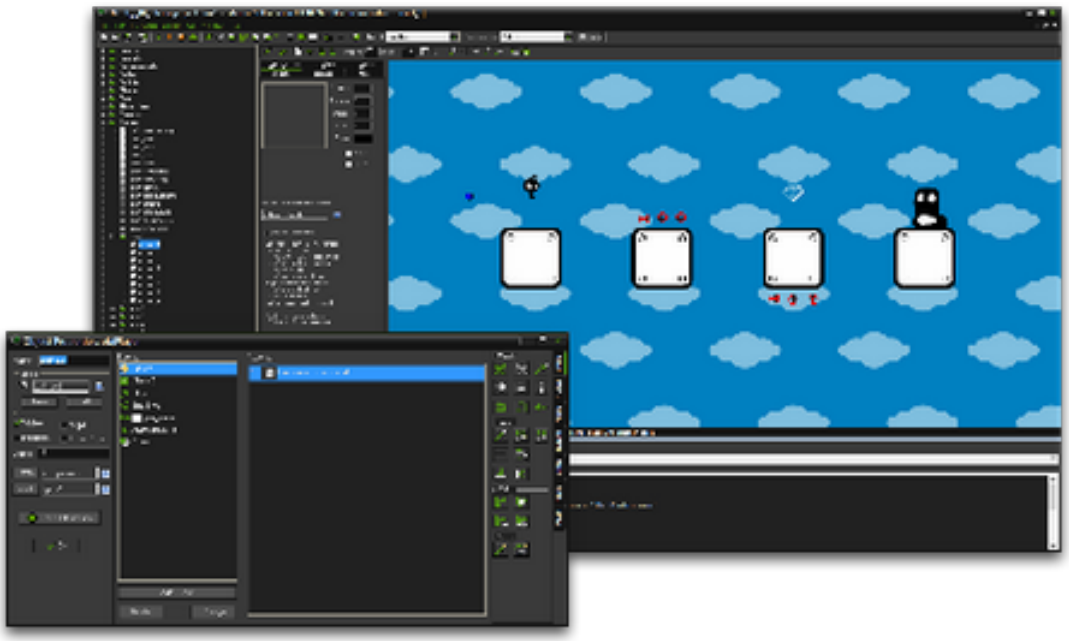


Figure 1.12: *Game Maker: Studio* development windows examples. [16]

Unreal Engine.[17] Made by *Epic Games* [46], it is probably the most powerful free game engine of the market. A lot of tools for high quality games development are included, but it is not the easiest game engine to learn.

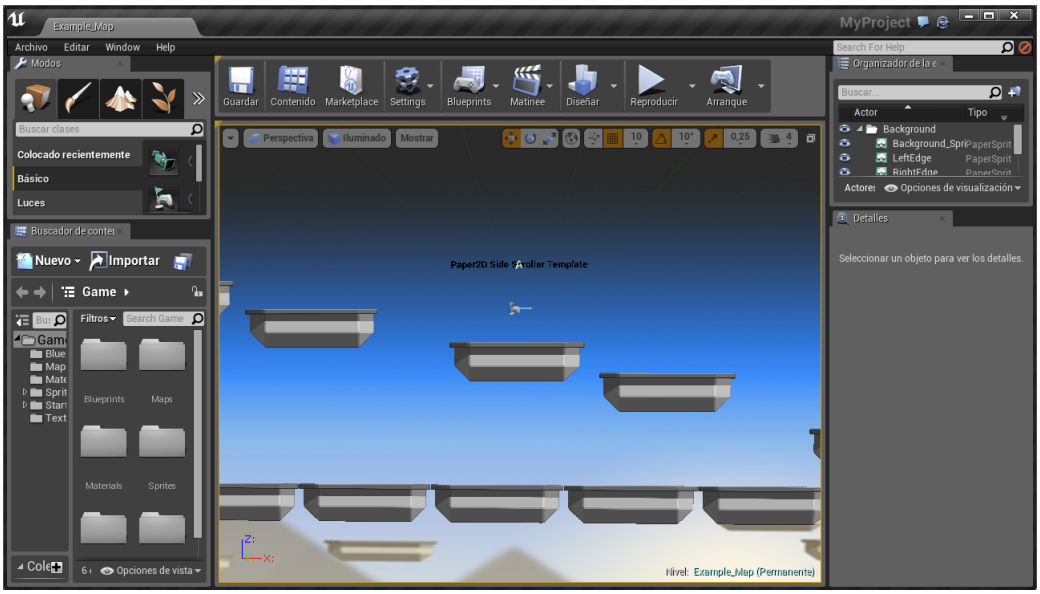


Figure 1.13: *Unreal Engine* project example. [17]

Unity 3D. [3] It is a game engine that is really easy to use, but it still maintains a lot of features and tools that help in game development. *Unity 3D* is good too at accepting lots of formats for the assets without too much problems.

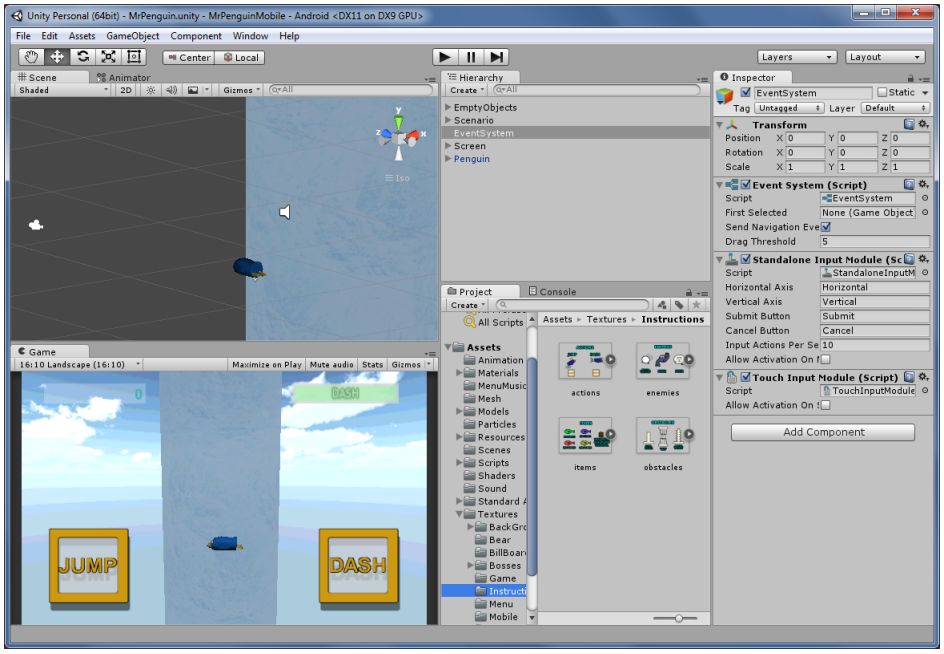


Figure 1.14: *Unity 3D* project example. [3] [2]

1.3.4 Use of previous results

1.3.4.1 Map generators

As we have seen in the previous section, there are a lot of already made methods for generating maps, but we will not use any previous implementation. From all the ideas gathered from the investigated methods, we will develop a map generator specially created for a 2D side-view platformer. The biggest influences will be *the recursive backtracker* and the *cellular automata*, making the resulting graph a kind of a tree that grows following rules.
The decision to make our own generator is mainly due to the fact that most of the previous research was not done having in mind a side-view platformer as the target (ignoring for example the gravity our game will have), and due to the feeling of wanting to create our own innovative system.

1.3.4.2 Level generators

The idea of procedurally creating little challenges in rooms instead of using pre-made patterns seems that has not been investigated too much, so we will implement our own method (which is expected to have an outcome a lot less repetitive than other easier methods we have seen) taking big references in the level generation of *Cloudberry Kingdom*.

1.3.4.3 Game engines

Unity 3D is going to be the game engine used in the video game development as it versatile enough to let us use procedural generation without complications, and it is simple enough to not waste time on learning how it works. Furthermore, we already have experience developing video games with *Unity 3D*.

2. Project Management

2.1 Temporal planning

This section is going to describe the planning of the time and resources spent on the project. The project officially started on the 22nd of January when it was accepted by the FIB administration, and it must be delivered before the 22nd of June.
The planning explained in the next pages is the ideal effort distribution for the 5 month time between the start and end of the project, but we have to take a realist point of view and consider that the work time will probably have to adapt to the development (and this isn't necessarily something bad).

2.1.1 Tasks description

Although the project will be following an implementation of the *Scrum* agile development methodology, and this implies the tasks will be defined at each sprint, a previous definition of all, or at least groups of, tasks is needed for making a temporal planning. The agile development will let the development adapt and search for alternatives in many cases but those are the groups of task that must happen for the project to reach an end:

2.1.1.1 Work previous to the project

This subsection gathers all the tasks that were made before the official start of the project. These tasks are, including but not limited to, the following:

- Investigating the contexts and states of the art of interesting topics.
- Investigating the viability and effort required for the different options.
- Choosing the best option.
- Defining the initial name and description of the problem.
- Applying to the FIB administration process.
- Trying some really small prototypes.

All these tasks were made with the help of the project director and even if they are not part of the project, they were decisions that lead to the development of this project. Furthermore, they would also help in future tasks such as the description of the state of the art in the project planning.

2.1.1.2 Project planning

This subsection is the point of the project where we currently are. All the tasks in this group are essentially everything that is covered in the GEP course and are:

- Scope definition.

- Temporal planning.
- Economic and sustainability management.
- Preliminary presentation.
- Context and bibliography.
- Specialization justification
- Planning presentation and final document.

This is the part of the project with the most control on the time for every task because there is a compulsory planning with deliveries, such as this, every five days.

2.1.1.3 Main development

The tasks in this subsection are the ones that will need the biggest effort and time of all the project. They are also the most sensitive tasks to be changed and affected by alternatives while developing the project. They can be mainly divided in three big parts.

- **The map generator**. The development of this generator can be divided in the investigation and analysis of different known algorithms, and the implementation and adaptation of these algorithms to the desired solution for the video game we design.
- **The room generator**. As in the map generator, the room generator can be divided too in investigation and implementation of an algorithm.
- **The video game**. This part has many different tasks composing it such as: the game design, the implementation of the generators in the game, the physics, the gameplay, the graphics, the audio, the user interface and the the analytics functions.

These three big parts are explained in the scope section of the planning too but it is important to highlight how the two generators are mainly independents one from the other, but the video game needs them both to work. Taking this into account, the development of the generators will be, as the whole project, an iterative development, and the video game will start developing as soon as there are working generators (even if they are really primitive).
This iterative way of working makes leaving some time for refactoring the code after its development a really good idea. The code refactoring may lead to some optimization, but more importantly to improve the code readability and reduce its complexity for a better maintainability and reuse in the future.

2.1.1.4 Validation process

This is an important part of the development as it will evaluate the quality and results of the project. The tasks in this subsection are mainly divided like this (in order of frequency):

- **Own tests**. These will be done over all the project to make sure everything we develop works as expected and to check the quality of the results by our own standards.
- **Director supervision**. This is translated into all the communication and meetings where the director of the project is able to give us feedback seeing the changes on the project.
- **Testers evaluation**. This includes making the surveys, finding the testers, letting them play the game and answer the survey, and analyzing the results obtained.

2.1.1.5 Final tasks

This group of tasks represents all the tasks needed to complete the documentation, to take care of any unfinished work and prepare the final presentation of the project. It is also the time for solving any last problem found while checking if everything is correct.

2.1.2 Time table

As this project is done only by one person (me), we only need the table 2.1 that shows the time needed for each task defined before.

Task	**Time spent (hours)**
Viability investigation and first prototypes	30
Project planning	65
Map generator	80
Room generator	80
Video game	140
Validation process	50
Final tasks	30
Total	475

Table 2.1: Estimations of the time spent in each group of tasks. [2]

2.1.3 Resources

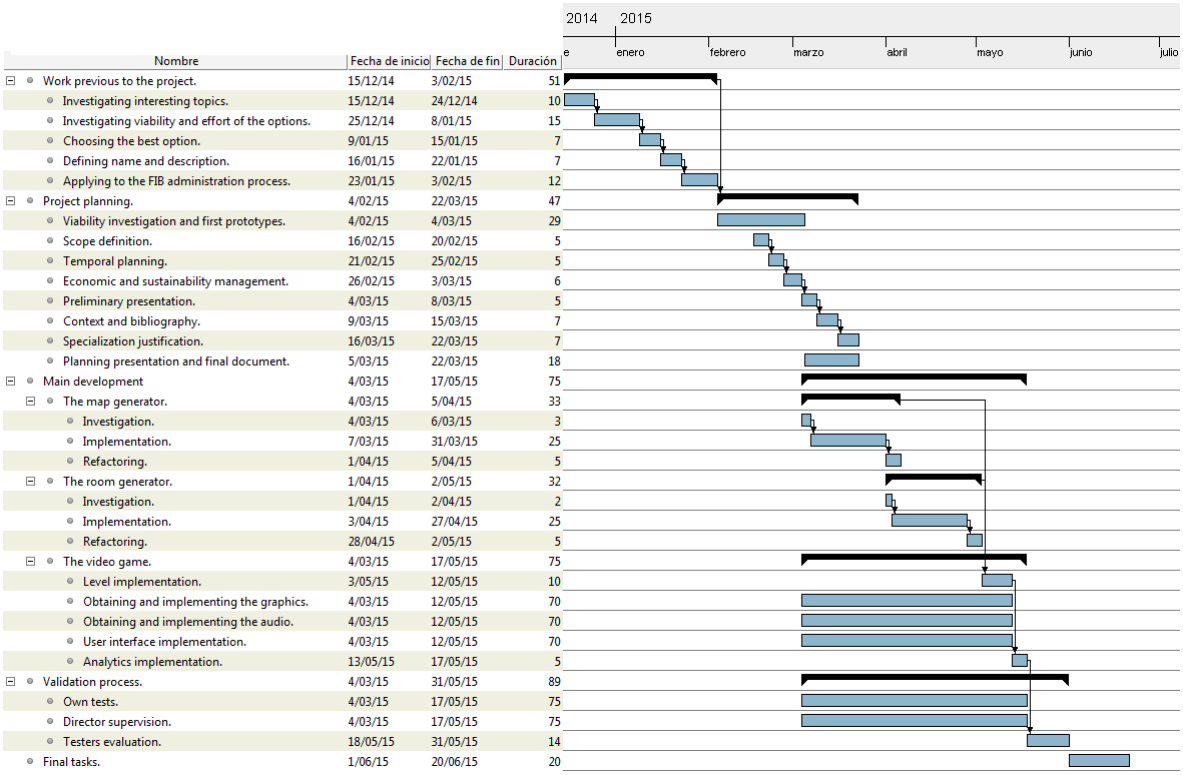
As for the resources needed for the project, only time with my PC and some software will be needed. The PC works on Windows 8.1 and has the following specifications: Intel Core 2 Duo E8400 at 3.00 GHz, 3 GB of RAM, NVIDIA GeForce GTX 650.
All the other resources needed are software that can be obtained for free: *Unity 3D*[3], *MonoDevelop*[4], *Git*[5] (using a *Bitbucket*[6] account), *Gimp*[7], *Audacity*[8] and *LaTeX*[47].

The PC will be used for all the tasks, *Unity 3D* will be used for the video game creation and the validation process, *MonoDevelop* will be used in the main development (explicitly for all the code in the generators and the game), *Git* will be used for controlling versions and having a back up of all the project, *Gimp* and *Audacity* will only be used

in the video game development and *LaTeX* will be used for all the documentation of the project.

2.1.4 Gantt chart

Following the estimations and the ideal time distribution, a Gantt chart showing all the schedule has been made.



Nombre	Fecha de inicio	Fecha de fin	Duración
Work previous to the project.	15/12/14	3/02/15	51
Investigating interesting topics.	15/12/14	24/12/14	10
Investigating viability and effort of the options.	25/12/14	8/01/15	15
Choosing the best option.	9/01/15	15/01/15	7
Defining name and description.	16/01/15	22/01/15	7
Applying to the FIB administration process.	23/01/15	3/02/15	12
Project planning.	4/02/15	22/03/15	47
Viability investigation and first prototypes.	4/02/15	4/03/15	29
Scope definition.	16/02/15	20/02/15	5
Temporal planning.	21/02/15	25/02/15	5
Economic and sustainability management.	26/02/15	3/03/15	6
Preliminary presentation.	4/03/15	8/03/15	5
Context and bibliography.	9/03/15	15/03/15	7
Specialization justification.	16/03/15	22/03/15	7
Planning presentation and final document.	5/03/15	22/03/15	18
Main development	4/03/15	17/05/15	75
The map generator.	4/03/15	5/04/15	33
Investigation.	4/03/15	6/03/15	3
Implementation.	7/03/15	31/03/15	25
Refactoring.	1/04/15	5/04/15	5
The room generator.	1/04/15	2/05/15	32
Investigation.	1/04/15	2/04/15	2
Implementation.	3/04/15	27/04/15	25
Refactoring.	28/04/15	2/05/15	5
The video game.	4/03/15	17/05/15	75
Level implementation.	3/05/15	12/05/15	10
Obtaining and implementing the graphics.	4/03/15	12/05/15	70
Obtaining and implementing the audio.	4/03/15	12/05/15	70
User interface implementation.	4/03/15	12/05/15	70
Analytics implementation.	13/05/15	17/05/15	5
Validation process.	4/03/15	31/05/15	89
Own tests.	4/03/15	17/05/15	75
Director supervision.	4/03/15	17/05/15	75
Testers evaluation.	18/05/15	31/05/15	14
Final tasks.	1/06/15	20/06/15	20

Figure 2.1: Gantt chart of the project. [2]

2.1.5 Action plan

Finally, an explanation of how is the plan going to be followed is needed.

The initial idea is to work following the schedule seen in the Gantt chart, but there are always obstacles and unexpected setbacks in projects so we must be ready to prioritize objectives. First of all, if the limited time ends falling short for the development, parts of the schedule like the refactoring of the code will be cut because they improve the quality of the project but they are not totally necessary for it to work.
The next part to be cut would be the quality of the video game graphics, audio and user interface as they are really time consuming and are not a technical part of the project.
In the worst case scenario, the last part to be cut would be the analytics implementation, which would leave the testers evaluation as something subjective only.
If even then the time was not enough, the planning would have been totally wrong

and instead of decreasing more the project quality, a new plan using the next semester would be done.

As explained in the methodology section of the previous delivery, we are going to try to arrange meetings with the project director every time there is an important topic to discuss (hopefully at least once a month) and we are going to keep communicating virtually.

In total, as it can be seen in the time table, the project should take approximately 475 hours. Taking into account the 22 weeks we have to develop the project, about 20-25 hours per week will be needed to finish it in time. The time spent each week will not be something totally exact as we have to consider other factors like the two subjects I am currently enrolled at the FIB.

2.1.6 Planning modifications

This section is going to describe how the planning of the time and resources has been followed and how it has been changed. There is no need to repeat the tasks definitions because they are still the same as in the original plan, but we will show again the time table with each task and the original planned hours for it, the real number of hours done until the follow-up meeting and an estimation of how finished the task was, and the real number of hours done at the end. After that, we will compare the original Gantt chart to the updated one.

2.1.6.1 Final time table

Here we can see how the different groups of tasks have been going in a quick to understand table:

Task	**Original estimated hours**	**Hours spent (follow-up)**	**Estimated percentage (follow-up)**	**Hours spent (final)**
Viability investigation and first prototypes	30	30	100 %	30
Project planning	65	70	90%	72
Map generator	80	70	95%	72
Room generator	80	85	95%	90
Video game	140	30	15%	140
Validation process	50	20	25%	50
Final tasks	30	0	0%	30
Total	475	305	66%	484

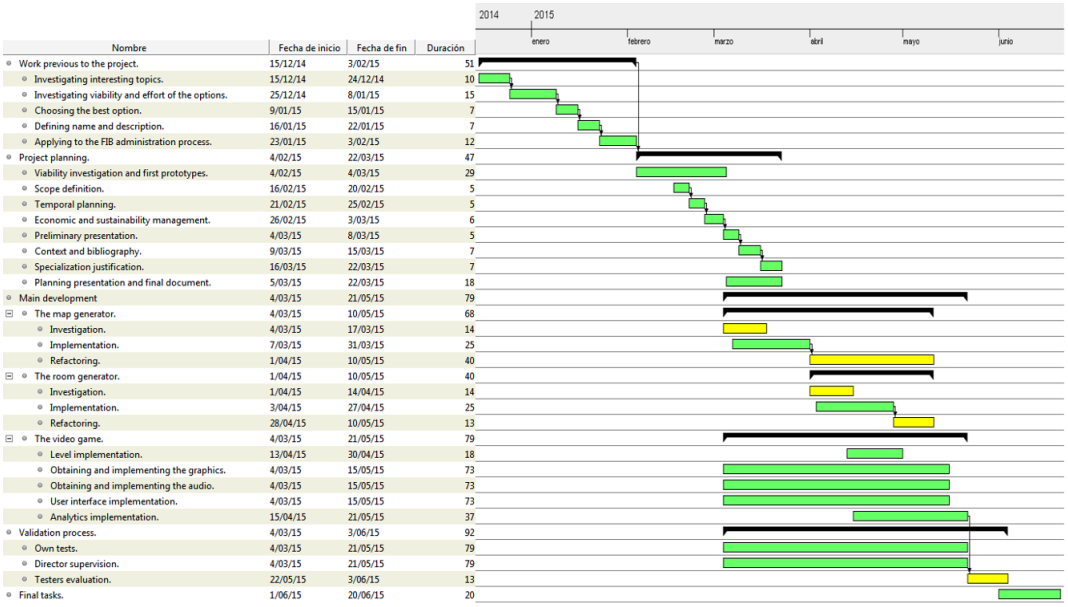
Table 2.2: Original estimations and real spent hours on the project. [2]

Looking at the table we can see that on the follow-up meeting the two first big parts (the generators) were practically finished and there was only one big part left that was the game and the finishing tasks, such as ending the validation process or the

documentation. The project was pretty much following the original plan before June and this allowed to finish without any big problem.

2.1.6.2 Final Gantt chart

Thanks to the agile development methodology and the fact that the original estimations were tight but were pretty flexible, the temporal planning has been followed nicely and as we can see in the updated Gantt chart there was no need for too many changes.



Nombre	Fecha de inicio	Fecha de fin	Duración
Work previous to the project.	15/12/14	3/02/15	51
Investigating interesting topics.	15/12/14	24/12/14	10
Investigating viability and effort of the options.	25/12/14	8/01/15	15
Choosing the best option.	9/01/15	15/01/15	7
Defining name and description.	16/01/15	22/01/15	7
Applying to the FIB administration process.	23/01/15	3/02/15	12
Project planning.	4/02/15	22/03/15	47
Viability investigation and first prototypes.	4/02/15	4/03/15	29
Scope definition.	16/02/15	20/02/15	5
Temporal planning.	21/02/15	25/02/15	5
Economic and sustainability management.	26/02/15	3/03/15	6
Preliminary presentation.	4/03/15	8/03/15	5
Context and bibliography.	9/03/15	15/03/15	7
Specialization justification.	16/03/15	22/03/15	7
Planning presentation and final document.	5/03/15	22/03/15	18
Main development	4/03/15	21/05/15	79
The map generator.	4/03/15	10/05/15	68
Investigation.	4/03/15	17/03/15	14
Implementation.	7/03/15	31/03/15	25
Refactoring.	1/04/15	10/05/15	40
The room generator.	1/04/15	10/05/15	40
Investigation.	1/04/15	14/04/15	14
Implementation.	3/04/15	27/04/15	25
Refactoring.	28/04/15	10/05/15	13
The video game.	4/03/15	21/05/15	79
Level implementation.	13/04/15	30/04/15	18
Obtaining and implementing the graphics.	4/03/15	15/05/15	73
Obtaining and implementing the audio.	4/03/15	15/05/15	73
User interface implementation.	4/03/15	15/05/15	73
Analytics implementation.	15/04/15	21/05/15	37
Validation process.	4/03/15	3/06/15	92
Own tests.	4/03/15	21/05/15	79
Director supervision.	4/03/15	21/05/15	79
Testers evaluation.	22/05/15	3/06/15	13
Final tasks.	1/06/15	20/06/15	20

Figure 2.2: Updated Gantt chart of the project.[2]
Green means finished as planned and yellow finished with more time than expected.

We can observe that the main changes in the temporal planning are the extra time we did on the generators investigations, which was specially useful to see how different the project is from other common approaches to the same problems, and the fact that the refactoring phase is still not finished because I am still not satisfied with the code of the project. Furthermore, there is another unexpected change in the planning that is the soon level implementation in the game. This was a consequence of realizing we could do this task while creating the generators more easily than if we left it for later. Taking this changes into consideration we decide to plan some more days for the other parts of the game just in case.
The last little change in the planning was adding 5 more days to the time assigned to doing the test because some testers could not meet on the scheduled week.

2.1.6.3 Action Plan following

The action plan remained the same all the project as we thought maintaining the work rhythm we planned would lead to finishing the project on time while being able to handle other curricular and extracurricular projects. We were right and even with

different setbacks and obstacles we only did have to sacrifice a bit of the time scheduled for code refactoring.

2.2 Budget and sustainability

This section is going to describe the planning of the resources (including material and human resources) spent on the project, and it is going to evaluate the sustainability of the project too.
This will result in a total budget and some marks for sustainability aspects.

2.2.1 Budget estimation

This subsection will show the original evaluation of the resources usage and costs, and it will obtain a total budget from this evaluation. The resources will be divided into material resources (hardware and software) and human resources, and then they will be joined.

2.2.1.1 Material resources

These are the hardware and software resources that will be needed for the project development.

The amortisation of the material resources will be calculated following the Spanish law and taking into account the useful life of the products used (4 years for hardware and 3 for software at most) and the five month length of the project.

Product	Price	Useful Life	Amortisation
PC	700.00€	48 months	72.92€
Computer peripherals	200.00€	48 months	20.83€
Windows 8.1 [48]	119.00€	36 months	16.53€
Unity 5 Personal Edition	0.00€	36 months	0.00€
MonoDevelop	0.00€	36 months	0.00€
Git	0.00€	36 months	0.00€
BitBucket	0.00€	36 months	0.00€
GIMP	0.00€	36 months	0.00€
Audacity	0.00€	36 months	0.00€
LaTeX	0.00€	36 months	0.00€
Total	1019.00€	-	110.28€

Table 2.3: Material resources budget. [2]

As we can see in table 2.3, we are going to use many software tools with no cost. Moreover, *Git*, *GIMP* and *Audacity* are open source too.

2.2.1.2 Human resources

These are the costs originally estimated from all the work hours that would go into the project development.

We must differentiate between two kind of testers (both part of the validation process): the software tester who evaluates all the software while knowing how it works and how it should work, and the game tester who only will play the resulting game of the project and will provide us feedback. It is important to notice that the game tester will be the only work we will not be doing ourselves, as we explicitly need feedback from other people.

Role	Price per hour	Time	People	Cost
Project Manager	50.00€	115 hours	1	5750.00€
Software Designer	35.00€	85 hours	1	2975.00€
Software Programmer	25.00€	225 hours	1	5625.00€
Software Tester	20.00€	50 hours	1	1000.00€
Game Tester	20.00€	1 hour	10	200.00€
Total	-	-	-	15550.00€

Table 2.4: Human resources budget. [2]

The salaries of table 2.4 have been taken from real salary surveys [49] as if it was the first year of experience of all the roles.

2.2.1.3 Total budget

Taking into account the costs that were shown in the previous sections, the total budget can be calculated without ignoring the taxes that would apply to a project of this kind (in Spain a 21% of IVA for video games which is what this would be considered).

Concept	Cost
Material resources	110.28€
Human resources	15550.00€
Total	15660.28€
IVA	21%
Total with taxes	18948.94€

Table 2.5: Total budget. [2]

2.2.2 Budget control

As it can be seen in the previous sections, most of the cost of the project comes from human resources. This means that any big change in the hours needed for any part of the development is the biggest threat to our budget, and we should be careful about following the planning.

On the other hand, the material resources represent a non-significant part of the cost, as very few hardware and software is needed. So during the development, finding any new requirement (like some non-free software that would improve the project) would

not be a problem and could be included easily.

The last risk for the budget would be not being able to obtain free graphics and audio for the game, which would lead into buying them already done or paying someone to make them (it could be me working as an artist). But this is a really small threat as there are plenty of free and open assets we can use on the Internet.

2.2.3 Budget modifications

In the follow-up meeting we had some costs because human resources and we did some new estimations. We had to add 5 extra hours that went into the project planning which is the group of tasks mainly done by the project manager and we estimated even 10 hours more to finish it, which was a totally correct decision we have been able to achieve. We also estimated that even if there were some unexpected hours in the generators tasks, at the end everything would finish with a correct balance of hours taking into account that some other parts were expected to be finished with less hours, as it has been. Therefore, we just have to add again the changes we did for the follow-up meeting. This makes a total of: 15 hours x 50.00€/hour of the project manager = 750€to add to the budget.

Just adding the new costs creates the final budget estimation (which was already correctly estimated on the follow-up meeting).

Concept	**Cost**
Material resources	110.28€
Human resources	15550.00€
Extra human resources	750.00€
Total	16410.28€
IVA	21%
Total with taxes	19856.44€

Table 2.6: Final total budget. [2]

2.2.4 Sustainability

This section will analyze the sustainability of the project in three different areas: economic, social and environmental.

2.2.4.1 Economic sustainability

After evaluating the resources costs, we have to consider some more things about the project. For instance, if after finishing the planned development there will be updates or any other working. If the project had a real commercial objective there would be updates for sure, but as the final resulting video game will only be for academic purposes and validating the procedural generators investigation there will be no need for further work.

However, the viability of the budget if this had to be a competitive project has to be evaluated anyway. For this evaluation, we have considered that the budget was as cheap as possible because all the working hours will be needed and there is nothing that could be cut. If the video game was the only purpose of the project there could have been alternatives like using already developed map generators to save a lot of money. There are really cheap and good solutions such as *Pro-D Total* [14] which only costs 75 dollars instead of creating our own map generator that costs more than 2000 euros just in human resources, but that would defeat the purpose of the project to investigate and create our own version. Moreover, any other kind of shortcut has been discarded too for the same reasons.

We have to consider too that the working hours of the project are relative to the importance of the tasks and that they are already quite exact, so there is no room for saving money cutting some hours.

Even after taking into account all this, and knowing there would be no external support for the project such as any kind of collaboration with a company for example, we believe the budget is a cheap one for a video game (which is the easiest part of the project to be sold and make the project viable). Proof of this is the popularity and economic success of many games of the genre like the ones inspiring this project. Taking a look into one of the most successful games from the list of inspirations, we find the story of *Rogue Legacy* [50] which had a cheap budget too and became profitable really easily. We couldn't expect for our game to be that successful. But just seeing that selling the game for 15 euros and having 1% of the sales *Rogue Legacy* had in one month during all the commercial lifetime of my game would result in some profits, makes it easier to imagine the economic viability of the project.

Therefore, we award a 7 in economic sustainability as we consider the budget is realistic and cheap which makes it viable with a big effort to make a profit if there was an intention to go commercial.

2.2.4.2 Social sustainability

The result of the project development is just a video game without any kind of ulterior motive than entertaining. Therefore, we will only take into account the state of the video games sector which is really healthy and is always welcoming new attempts at creating something fun and has no real impact in any person's life. Everyone that uses the resulting product will not be using it for need but just entertainment so we award a 5 in social sustainability as the project will not be doing anything good or bad for society.

2.2.4.3 Environmental sustainability

The usage of resources is extremely little and will be limited to all the time the computer used in the development is running.
Knowing that this means 475 hours, we can estimate that, if the computer consumes about 300W while running, the energy spent on the project is around 142.5 kWh.

These 142.5 kWh could be translated into around 48 kg of CO2, which is quite a bit but nothing compared to what we produce in any other daily activity and that is inevitable. Thus, we award a 10 in environmental sustainability. We considered this mark because there is absolutely no direct usage of resources for making the product (as it is digital), there is no dismantling waste, there is no other pollution than the energy consumption explained, and the map generator and room generator may be recycled for future projects.

2.2.4.4 Sustainability evaluation

As we have seen in the previous sections, those are the marks we originally awarded the different sustainability areas:

Sustainability	**Economic**	**Social**	**Environmental**
Planning	7	5	10

Table 2.7: Original marks of the different parts of the project's sustainability. [2]

2.2.4.5 Sustainability reevaluation

However, after finishing this project, we must reevaluate these marks with the complete sustainability matrix. This means revisiting the three sustainability areas and comparing with a mark (from -10 to 10) the expected results with the actual ones, and giving a mark (from -20 to 0) to the risks to the different sustainability types.
We must give a bad mark to the economic results because we finally had to add some more money to the original budget, but we just had adapt the budget once and with not much so the mark will be only a little under 0. We must also give another little bad mark to the economic risks as now there are even more games in the over-saturated market and some of them like *Cavern Kings* [51] and *Pixel Dungeon* [52] would be direct competition to our game.
On the other hand, we give a perfect score in results and risks to social and environmental sustainabilities because nothing has changed from the original evaluation.

Sustainability	**Economic**	**Social**	**Environmental**
Planning	7	5	10
Results	-2	10	10
Risks	-5	0	0
Total	0	15	20

Table 2.8: Finl marks of the different parts of the project's sustainability. [2]

This gives a total mark of 35 (in a scale that goes from -90 to 60) to the sustainability of this project.

3. Map generator

3.1 Design

As we have previously defined in the scope section of this document, the map we want to generate is in fact a simple graph where each node will be a room and arcs between them represent adjacency.
Moreover, each node (or room) has all the information needed to create its inside (which will be the job of the room generator). This information, that we call parameters, will include details such as if the room has a key, a special treasure or a power-up. And the other way around too, the parameters also include if the room needs a key to open a door or if there are special challenges that need a certain power-up to be feasible.
The first design decisions that were made were that the game would have a final objective at each level/map, so another parameter was added that said if the room had the objective or not, and that the graph would not have cycles (therefore, it would be a tree from the initial room).

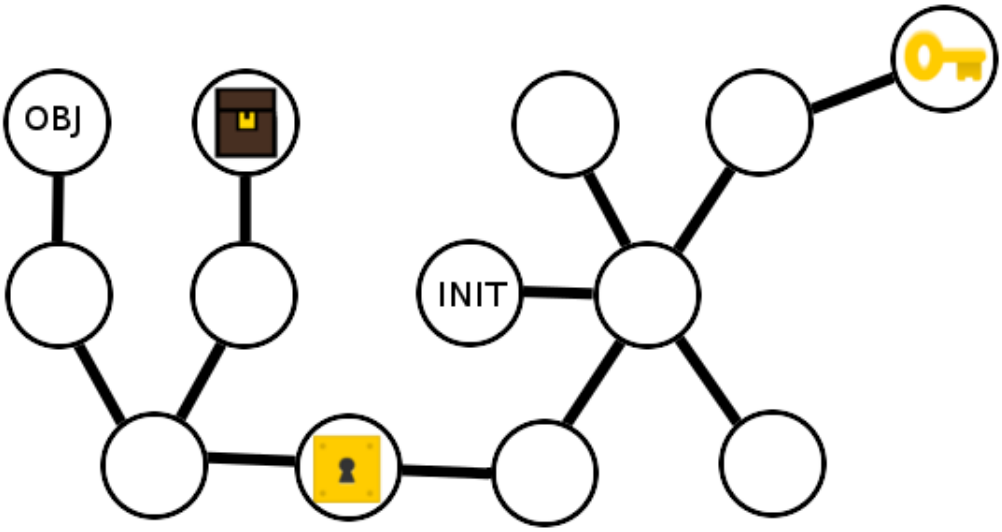


Figure 3.1: Original map idea example with a key, a lock, a treasure and the objective. [2]

This original idea was a little transformed for its implementation as we decided to create the map having each room as a rectangular tile (with no overlapping) that would make the process of knowing where a room is much easier.
This had many other advantages such as being impossible for two rooms to collide on space or obtaining faster the nearest rooms to the actual one.

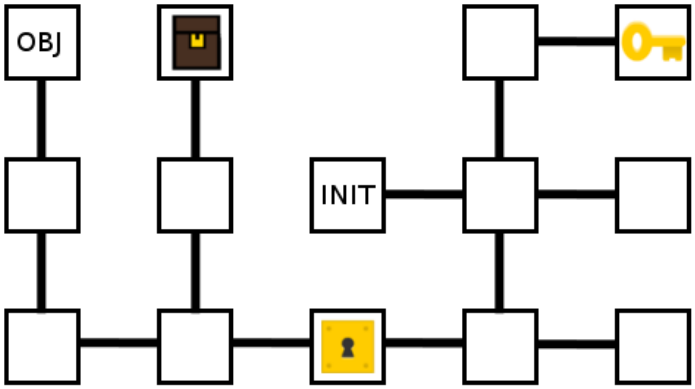


Figure 3.2: First change in the original map idea: Using rectangular tiles. [2]

While prototyping and investigating (as we see in figure 8.4), we quickly noticed that having every room be of the same size would cut the diversity of the rooms (which is something really important for the project) too much and that it would also make difficult to create more than one exit on each wall of the rectangular room.

This made us adapt the idea once again by using the tiles and the tileset (tileset as in grid of tiles) in a total different way. Instead of being rooms, each tile would be an indivisible part of the room that could have walls, exits or nothing at each of the 4 sides, and each tile will have a reference to its room to quickly know if it is being used or not and by what room it is being used if it is not empty. This new version of the idea that combines the graph with a tileset has all the advantages of the graph (for instance, navigation through the map and searches are really easy to do), all the advantages of using tiles (no room collisions and quick situational context of a room for example), and yet we get rid of many limitations from the previous design (rooms can having different sizes is almost trivial and we can control the exits of the rooms easier).
However, we decided to put a limitation by ourselves on this new design, all rooms must have rectangular shapes. This limit was decided to make easier the map implementation, but specially to not make the room generators job too difficult before starting it.

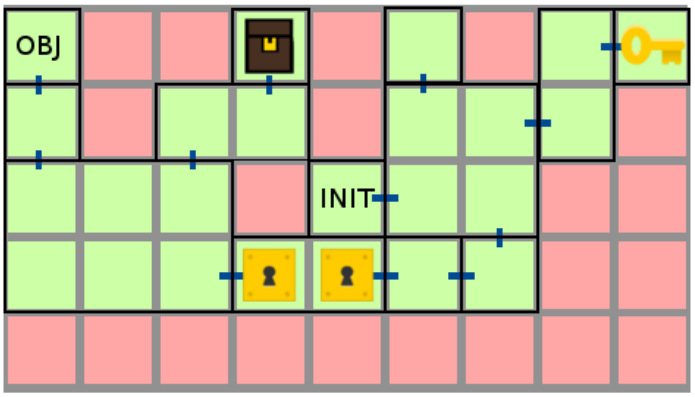


Figure 3.3: Final design of the map example. [2]

As we can see in the figure 3.3, we used this new design as a chance to simplify how putting exits in a room would be by only putting exits in the middle of the tiles sides. This way exits from different rooms will always connect, and we are not limiting too much our map generator because we can decide to have big or small tiles to play with the number of possible exits our rooms have.

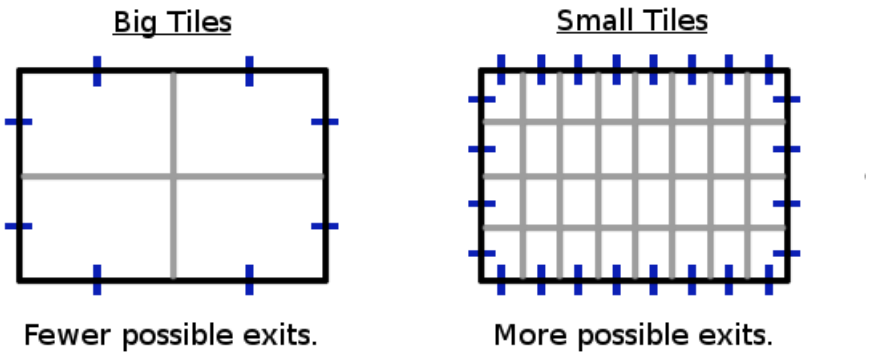


Figure 3.4: Trade-off between tiles size and the number of possible exits of a room. [2]

Finally, there are some other parameters we must highlight. These are the distances in the graph,mainly being used the distance from each room to the initial room and the distance from each room to the correct path. We have named these parameters depth and side depth and they will be important for the implementation explanations of the next sections.

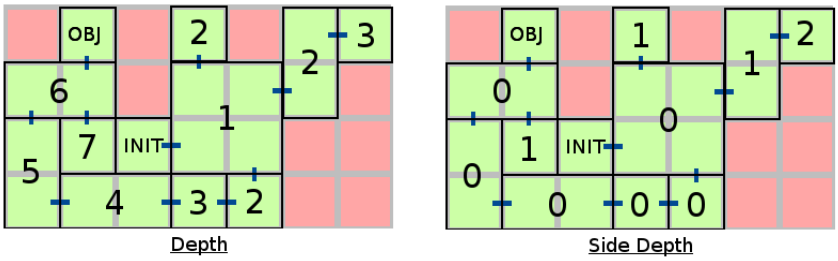


Figure 3.5: Example of depth and side depth parameters. [2]

3.2 General implementation

Once we had the idea of what should be the result of the map generator and we were sure we could implement it thanks to the investigation and prototyping, we started the real implementation of the map generator. This meant as we have explained in the introduction that we would create an algorithm that would create a map from a single room with all the information of the map. This map as we have explained in the previous section will translate into a tree graph where every room will have the connections to its previous (or parent) room and to its next (or children) rooms.

Inspired by the *recursive backtracker* and the *cellular automata*, the map generator works like a plant seed that has in its DNA all the information about how it is going to grow when it becomes a tree. This idea became our algorithm, where we create an initial room that has the following parameters: the maximum depth of the map, the maximum side depth of the map, the maximum size of the rooms, the number of locks in the map, the power-ups we can obtain in the map, the power-ups we will need to complete a map, the treasure probabilities, the shape of the map and the seed for the pseudo-random generator.

Keeping the tree metaphor, the algorithm works by making the tree (map) grow its different branches (paths of rooms) until some parameter tells them to stop. Basically, the process just goes to the most important room (the nearest to the objective) and using the parameters this room has and the information of its context from the tileset (mainly if there is free adjacent space or not), decides if there should be some next rooms to this one, how many next rooms and what adjacent tiles they will need. When creating these next rooms, it also decides by random (following some probabilities we can change) how the parameters are inherited. For example, we may have a room that in its parameters has that this branch needs to have a key so when it creates the next rooms it has to decide if the key will be part of the room own content or if one of the next rooms inherits the responsibility to put this key on the map. This makes that while growing, the graph leaves trail of most of the decisions that happened when the rooms create their children.

Therefore, all this results in a map we can navigate following the connections or quickly access to a certain position through the tiles of the tileset. The nodes/rooms of the map also have where everything is and has a trail of the path of how to get there that could be used in many ways (such as in an ingame map or compass).
Moreover, we must explain that almost every decision is done by having different probabilities (that change with the parameters, for example if a room's depth is over the max depth the probability of this path to keep growing will be zero) for each option, and then we get a number from a pseudo-random number generator and combine this number with the probabilities to choose. This allows us to make all the process predictable if we have the seed of the pseudo-random generator and saving a map is just a matter of saving the number that we use as a seed.

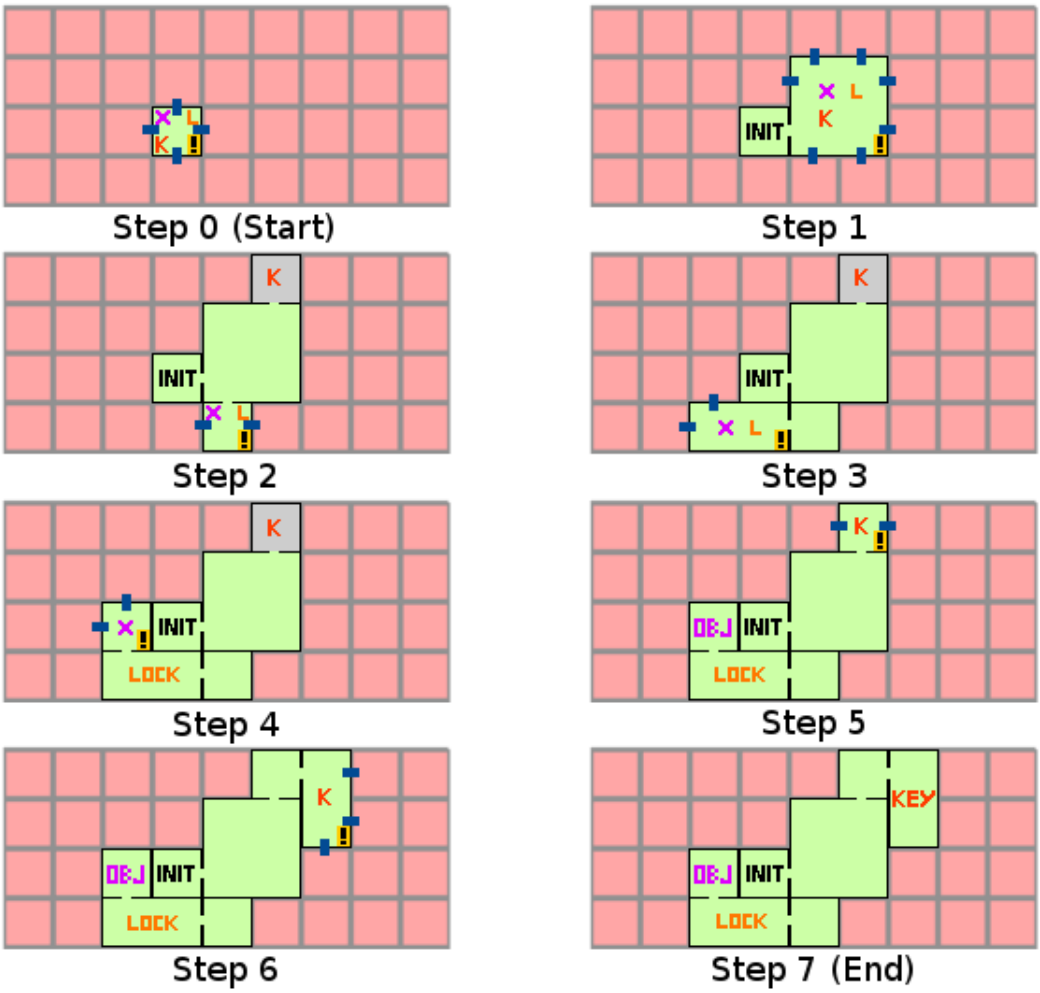


Figure 3.6: Example of the map growing process. [2]
We can observe how the most important room is marked with a !
in every step and that this room has its possible exits indicated.

4. Room generator

4.1 Design

With the idea of procedurally creating little challenges in rooms instead of using pre-made patterns we designed a graph plan that would help to create the content of the rooms and to check that it works for our game. We chose a directed graph, where the graph's nodes are the different platforms and obstacles that we can find in a room, and the edges are the movement connections between these platforms.
The platforms and obstacles only have their type and their position and space, but connections are a bit more complex as they have information about the platform from where you start, the platform where you end, the initial position for the movement, the final position where the movement leaves you and what type of movement this connection is (walking, jumping or dropping down). This kind of graph lets us easily

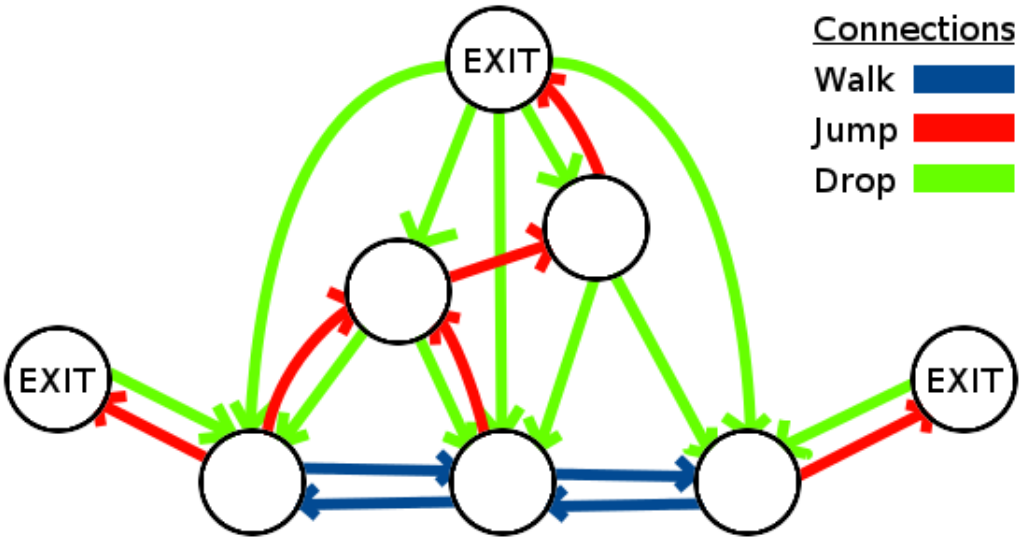


Figure 4.1: Connectivity of the room contents is expressed as a graph. [2]

check if the level is feasible by checking if there is a path between all the exits in the room. However, we should also check if the whole graph is connected to be sure that there are no places where the player could get stuck or that there are no useless platforms with no way to them.
Once the idea of using these graphs for the room contents was decided, we only had to choose how to create them. As we did not want to use pre-made patterns, we finally went with the idea of creating a system that checks a room for those connections we have described. In fact, we decided to make an iterative system where every time we added a node or platform, we would check all the previous nodes with the new one to see if there was any new connection. Which leads us to the next section.

4.2 Connections checking

We implemented a system that given two platforms and the context of those (rest of the content of the room where they will be), the system updates the connections between the two platforms with at least one of the possible movements it finds possible to do. For example, if we have two adjacent platforms it will detect that we can walk or jump from one to the other in both directions.

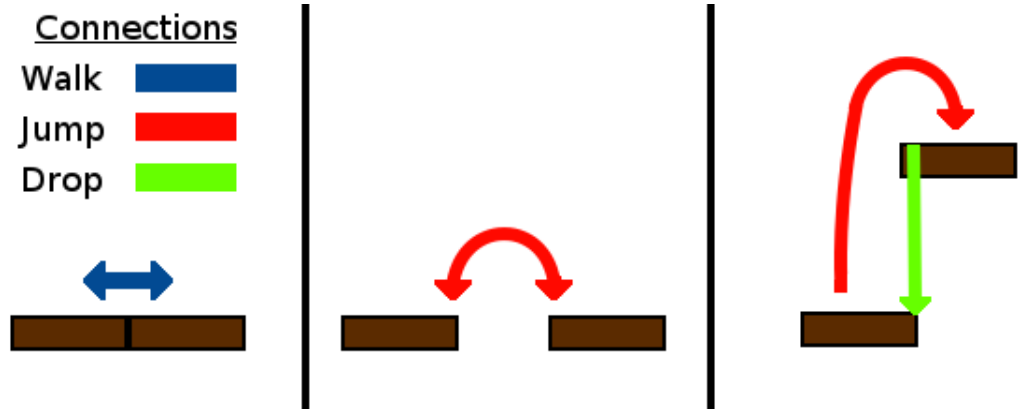


Figure 4.2: Examples of the connections checking system on different platforms. [2]

To do so, the system checks the different distances between the platforms and compares them to the player parameters like its jump height and length. In addition to expected parameters like the jump distances there are a couple of parameters that represent margins of error which are useful for making the jumps easier or more difficult. Once we know the movement we are trying would be possible with the parameters that the game has, we create an interpolation of the path this movement would do with the character hit-box and check if there are any collisions with any obstacle.

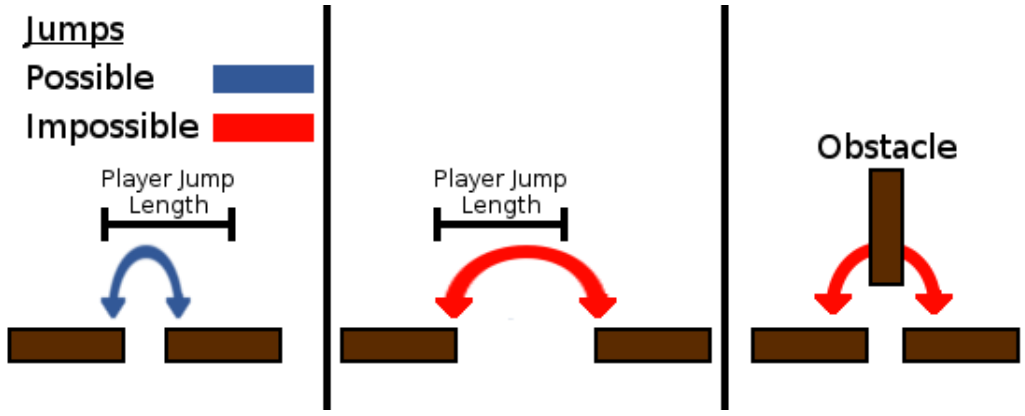


Figure 4.3: Examples of the connections checking system trying if jumps are possible between two platforms. [2]

In the current implementation we have implemented five different types of movements for the connections: walking, jumping, falling, dropping (which is jumping down from

a platform) and double jumping.
With this system and our idea of working with the graph connectivity we are ready to start creating rooms procedurally.

4.3 Random content generation

We have a system that lets us add floors, ceilings and platforms in general to the room content and after each addition check for different information like the feasibility of the room challenge or if every platform is connected. With this it was only a matter of selecting strategies to build our rooms. We could choose to use little patterns, random buildings or have some kind of heuristic function that tells us which should be the next piece. We chose the random buildings, as it is the method that would give us more diversity and that should be impossible to predict like the patterns and most probably the heuristic function (there is a chance that we could make a randomize heuristic function that would be interesting too but that would go to far from our original scope).
Our strategy with the random building is just to start by adding to the content some pre-made pieces we know that work without problem (floor and ceiling or even only the exits) and then use one of the many functions we have created to build rooms. Some of those functions are:

- A function that adds an exact number of random platforms in the room.
- A function that adds random platforms in a limited space of the room.
- A function that fills the room with random platforms until it is a feasible room.
- A function that fills the room with random platforms until all the platforms are connected.
- The same but with a minimum distance between platforms.
- The same again but with a minimum distance between platforms that reduces over time.
- A function that hides the rectangular shape of the rooms.
- A function that adds interesting shapes like a doughnut in the room.
- Etcetera.

Basically we always add content to the room until the level is at least feasible no matter what functions we use.
After finishing the part of the process where we add floors and platforms and we have the shape of the room, we add the objects (like keys), items (like money) and monsters to the room. We get a random number from the room parameters and we just put them randomly on different platforms. In a similar fashion, if the room parameters say that a power-up is needed in the room, we just take a random number of platforms and change their type to the type that needs that power-up (for example we activate the fire type on a platform).

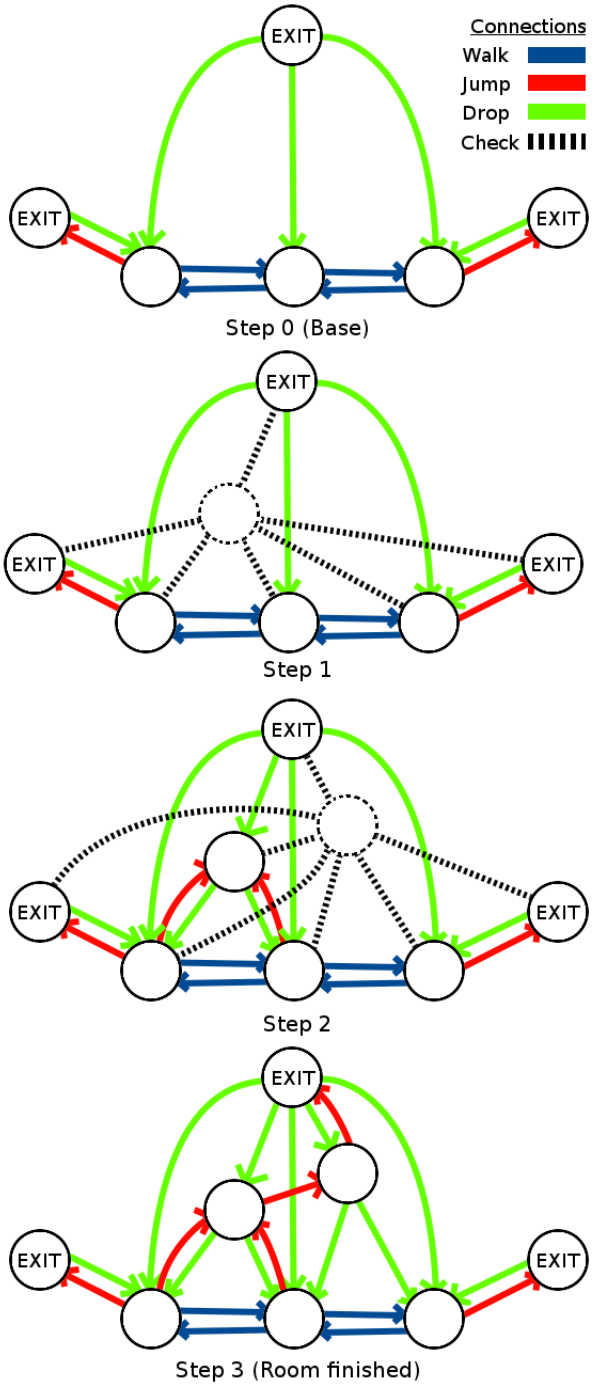


Figure 4.4: Example of how we use the connection checking system to build a room. [2]

5. Video game

This section will explain many parts that were designed and implemented in the video game of the project. For getting a complete idea of how the game was designed without looking at the implementation, the game design document has been attached as an annex in section 8.3.

5.1 Gameplay

We created a character for the player to control that is always the focus of the game. The character can be controlled to run, jump, duck, drop from platforms and double jump, and the camera will always follow him as he explores the levels which are created by the map and room generators.
The character can also kill the enemies on the levels by jumping on them and collect objects and items by touching them.

The character we implemented mainly works by checking the input from the player, and checking what is happening by trying to find any collision of its hit-box[1] with the content of the room. For example if it collides from below with the floor it will stand, but if it collides with an item the character will collect the item.

It is important to highlight that we implemented the jump of the character to be easily adapted to a desired jump height (and in consequence jump length) so the character would jump exactly what we told the generator it would do.

5.2 Generators

Once implemented, both generators were easily used in the video game as they were developed with that goal in mind. Most of the work done was making that everything the room generator created had a hit-box, so everything could be detected and work with the player's character.
Moreover, we prepared different sets of parameters for the different levels in the game, which try to add diversity and have different environments. The six defined sets of parameters are:

- Initial level.
  The depth and max depth are low so we create a short level.
  There are no keys and locks.
  There is the first power-up (double jump).
  The challenges do not need any power-up.
  The rooms have rectangular shapes and there is no attempt to hide it.
  The rooms have a maximum size of 2 tiles of length and 2 tiles of height.

[1] a rectangle defining its space and its collision area

- Factory level.
  The depth and max depth are higher than the initial level and increase with each completed level.
  There is a red key and a red lock.
  There is the second power-up (fire resistance).
  Some challenges need the second power-up.
  The rooms hide their real rectangular shape with different kind of walls and interesting shapes inside the rooms.
  The rooms have a maximum size of 3 tiles of length and 3 tiles of height.
- Pyramid level.
  The depth and max depth are higher than the initial level and increase with each completed level.
  There is a yellow key and a yellow lock.
  There is the third power-up (spikes resistance).
  Some challenges need the third power-up.
  The rooms are tunnel shaped.
  The rooms are 1 tile long and 1 tile tall to make even smaller tunnels.
- Sky city level.
  The depth and max depth are higher than the initial level and increase with each completed level.
  There is a blue key and a blue lock.
  There is the fourth power-up (feather-weight).
  Some challenges need the fourth power-up.
  The rooms have no walls and the map can be explored in unexpected ways.
  The rooms have a maximum size of 3 tiles of length and 3 tiles of height.
- Underwater level.
  The depth and max depth are higher than the initial level and increase with each completed level.
  There is a green key and a green lock.
  There is the fifth power-up (anti-sticky jump).
  Some challenges need the fifth power-up.
  The rooms hide their real rectangular shape with different kind of walls and interesting shapes inside the rooms. They also are a little adapted to the fact that the player has infinite jumps in the underwater level.
  The rooms have a maximum size of 3 tiles of length and 3 tiles of height.
- Final level.
  The depth and max depth are higher than any previous level.
  There is a black key and a black lock.
  There is no power-up.
  Some challenges may need any previous power-up (except the double jump).
  The rooms hide their real rectangular shape with different kind of walls and interesting shapes inside the rooms.
  The rooms have a maximum size of 3 tiles of length and 3 tiles of height.
  There are more treasures.

As we can see in the sets, there is no level that requires the double jump power-up because after testing it, levels that required this power-up were not as interesting.

5.3 Graphics

We obtained graphics from mainly three sources: *Kenney* assets [18], *Eeve Somepx's* fonts [19] and our own content [2].
Kenney assets were great as they were of public domain and were really diverse (there was even an special set of graphics for 2d side-view platformers like ours). Therefore, we could use them and modify them without problem and we could add some diversity to the levels without paying anything.

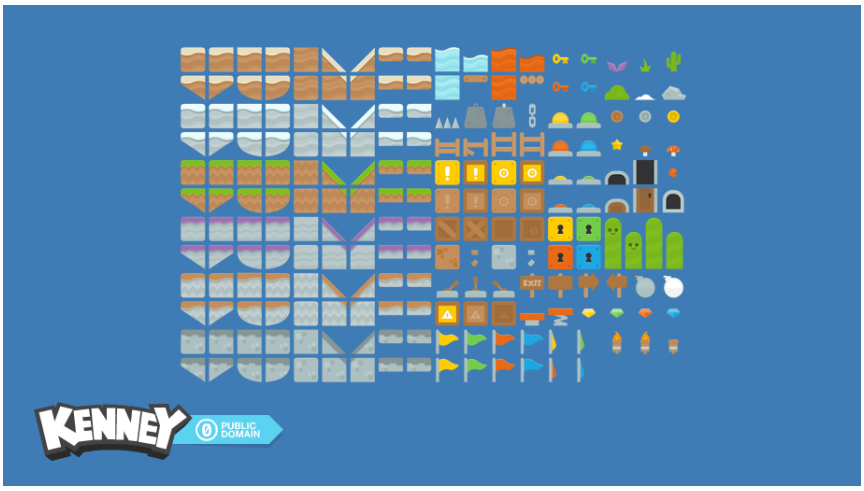


Figure 5.1: Kenney assets diversity. [18]

As we had many different graphics, we made the graphics change for each level to add diversity and try to differentiate the environments of each level.
Eeve Somepx[19] font *Eleven*, which has a *CC-BY license*[53], was really useful for creating the interface with an alternative to the usual boring fonts and gave the game a more professional look.

Figure 5.2: Somepx font preview. [19]

Finally, our own content is really little but was needed as some graphics (like the power-up platforms) were too specific to find online and for free.

5.4 Interface

We can divide the interface implemented in mainly three parts: the main menu, the level selection screen and the HUD[2].
For the main menu, we just implemented some important buttons and panels:

- Play button. It leads to the level selection.
- Options button. It opens the panel that controls the music and sound volumes, and that resets the game state.
- Instructions button. It opens the panel with the instructions text.
- Credits button. It opens the panel with the credits of the graphics, the font used (important due to the *CC-BY license*) and the project.
- Exit button. It leads to the leaving decision.

The level selection screen is a simple menu where there is a button for each level unlocked where the powers and objectives obtained are also shown.

The HUD shows the health of the game character, the money collected and notification texts when the player collects an object or an item. Moreover, the interface in the game screen also includes the map. The map can be shown anytime by pressing a button an it shows different information depending on the upgrades the player has collected:

- Without upgrades: The map shows every room visited.
- 1 upgrade: The map shows every room visited or not.
- 2 upgrades: The map shows where the map upgrades are.
- 3 upgrades: The map shows where the power-ups are.
- 4 upgrades: The map shows where the keys are.
- 5 upgrades: The map shows where the stars are.

5.5 Audio

We decided to use some procedurally generated music and sounds for the game. This way we would not find any licenses problems. Luckily, nowadays there are simple online solutions for this task, so using *Abundant-music*[54] for the music and *Bfxr*[55] for the sounds, we tried to create the most fitting audio for our game. We can specially hear this attempt at using fitting audio in the different songs for each level, where for instance the music for the pyramid level tries to simulate Arabic music.

[2]Heads-up display, which is mainly the interface that goes with the gameplay part

6. Validation

6.1 Test explanation

As we have mentioned before, we did not think our own opinion would be enough to validate the results of the projects, so a test with people outside the project was designed. The test chosen was playing the resulting game because validating the game also involves validating the two generators that are inside of it. Added to playing for a while (which was finally decided to be at least 10 minutes so it could be enough to get a nice idea of the game, but not too much to be a burden for the testers), some feedback would be collected. As explained in previous chapters, this feedback has two parts: subjective answers collected with a simple survey, and objective data collected by the game without the player noticing. Besides this feedback, the test was established to be done by the tester in front of us so we could obtain our own conclusions about their way of playing.

For the subjective feedback we decided to ask, with a score from 1 to 5, about if testers thought the game was fun (the ultimate goal of a video game and what should validate if our levels are good), there was diversity between levels, the fact that levels hide the real rectangle shape of the rooms and the innovation of the levels compared to other games they have played. This four questions were mainly asked to validate different aspects of the resulting video game of our project, but we also decided to include three free questions about what did the testers liked most, what did they liked least and what errors did they find to know how to improve or fix the game. You can check the exact questions and how they were explained in the video game user test on this document annexes.

For the objective data we finally decided to only collect the time spent in each level. Although there were many other things we could measure too, we decided to collect the level times as these would give us some objective feedback of how difficult the levels were and if they were as long as we tried to make them. The game was designed to be finished in about 25 minutes (and less than 30 minutes for sure), giving an average of less than 5 minutes per level (taking into account the first levels are easier and would be finished in less time, but the difficulty increases with each level finished and the later level would be finished in more time).

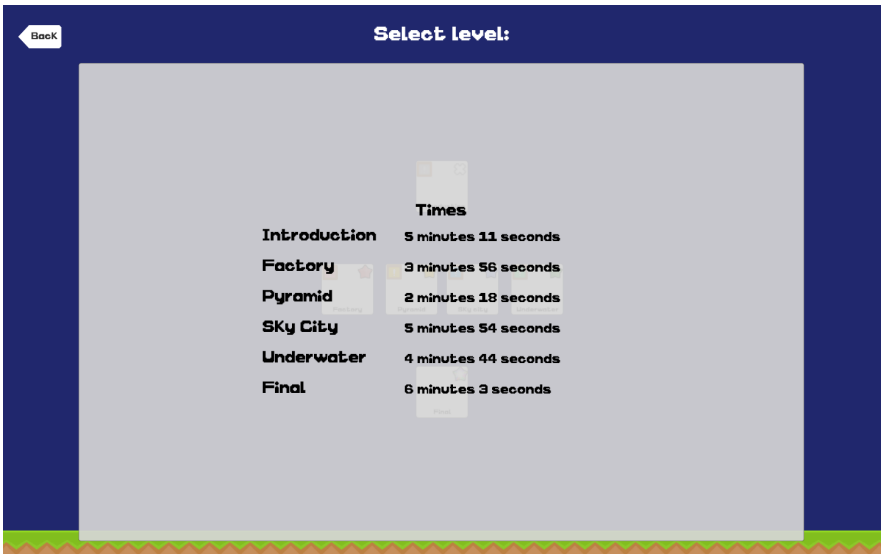


Figure 6.1: Game showing the times of a tester who finished the game.

6.2 Results

Was the game fun? Average answer: 3.8

1	**2**	**3**	**4**	**5**
0	0	3	6	1

Table 6.1: Answers to the first question of the test.[2]

Did you find diversity between the levels? Average answer: 3.1

1	**2**	**3**	**4**	**5**
0	3	4	2	1

Table 6.2: Answers to the second question of the test.[2]

Did you find that this* was well concealed? Average answer: 4.1

1	**2**	**3**	**4**	**5**
0	1	2	2	5

Table 6.3: Answers to the third question of the test.[2]

Did you find the levels to be different from levels of other games you have previously played? Average answer: 2.7

Things people liked most: (ordered from most mentioned to least)

1	**2**	**3**	**4**	**5**
0	5	3	2	0

Table 6.4: Answers to the fourth question of the test.[2]

The jump and jumping around.
The feeling of collecting lots of things.
Map upgrades.
The character.
The infinite jumps power up.
The simplicity of the controls.

Thing people liked least: (ordered from most mentioned to least)
Last level being too long.
The map.
Dropping from platforms.
Slow movement.
Not needing enough your powers.
Needing your powers too much.
Lack of backgrounds and better graphics.
Exiting level without confirmation.
Lack of better story.
No health replenishment items.

Errors found:[1] (ordered from most mentioned to least)
Collision errors.
Monsters not dying.
Not being able to leave sky city.
Not being able to read the notifications.
Not dying when you fall in sky city.
Never landing if you keep the jump pressed.
Animation errors.
Being trapped in a room with the star.
Being able to move after dying.

Times collected:
Introduction level: Average time: 3 minutes and 52 seconds.

< 2 min	**< 4 min**	**< 6 min**	**< 8 min**	**> 8 min**
1	5	3	1	0

Table 6.5: Times collected from the introduction level of the test.[2]

Factory level: Average time: 6 minutes and 21 seconds.

[1]Most of these errors where later discovered to be caused by using the Unity default "Fastest" graphics configuration which for example called the collisions checks less than the minimum needed

< 2 min	**< 4 min**	**< 6 min**	**< 8 min**	**> 8 min**
0	2	4	1	3

Table 6.6: Times collected from the factory level of the test.[2]

Pyramid level: Average time: 4 minutes and 58 seconds.

< 2 min	**< 4 min**	**< 6 min**	**< 8 min**	**> 8 min**
0	4	3	2	1

Table 6.7: Times collected from the pyramid level of the test.[2]

Sky city level: Average time: 5 minutes and 12 seconds.

< 2 min	**< 4 min**	**< 6 min**	**< 8 min**	**> 8 min**
0	3	4	2	1

Table 6.8: Times collected from the sky city level of the test.[2]

Underwater level: Average time: 6 minutes and 35 seconds.

< 2 min	**< 4 min**	**< 6 min**	**< 8 min**	**> 8 min**
0	2	5	1	2

Table 6.9: Times collected from the underwater level of the test.[2]

Final level: Average time: 8 minutes and 39 seconds.

< 2 min	**< 4 min**	**< 6 min**	**< 8 min**	**> 8 min**
0	0	1	4	5

Table 6.10: Times collected from the final level of the test.[2]

Total: Average time: 35 minutes 37 seconds

< 20 min	**< 24 min**	**< 28 min**	**< 32 min**	**> 32 min**
0	1	0	4	5

Table 6.11: Total time collected of the test.[2]

trying to get a better performance.

6.3 Conclusions

With the average answers we have seen for the first four questions of the test, we consider the results to be a success. The main fact that leads to this conclusion is that the testers considered the game to be fun even with the lack of important elements and only playing with the levels alone almost. We are also happy about how most of the testers did not notice the rectangle shape of the rooms until they learnt about it on the survey and most thought it was a nice way to hide the real shape of the rooms.

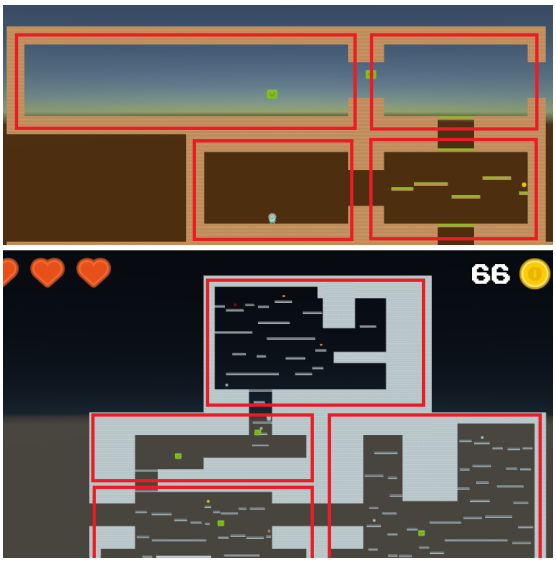


Figure 6.2: Example of how the first level (up) does not hide the rectangle shape and other levels (down) do.

However, we are not surprised about the second question not getting such a good score because even if there was some attempts at bringing diversity with different powers and shapes for each level, the idea behind all of them is the same and the art limitations do not help. On the other hand, we are a little disappointed about the last question as we thought we were bringing some innovation (at least to get a pass on the average score) to the levels compared to other games, but testers thought differently.

We are happy about the feedback on the questions with free answer as the testers showed they really liked the concept of exploring interesting levels and they also liked how the character and controls worked, although most of them were not happy with some small details. The tests were also really helpful for detecting a lot of errors and for seeing possible improvements in many places, which were used to improve and fix the game as we can see in the final version of the game. We must also note that some of the testers feedback was contradictory with the feedback of other testers (for example some felt there were not enough platforms needing powers and other thought there were too many, or there was some discussion about the levels length too), which showed many aspects of the game were just subjective to each player and would be impossible to please everyone.

Finally, when analysing the times collected something obvious was detected quickly: players were much slower than what we expected. We were used to the times we did

when trying for ourselves the game and did not take into account that players such as the testers would have to learn and did not know how the game worked from the inside like us. This was specially painfully visible in the last level were many testers got lost and found too long and difficult as they said themselves on the list of things that they did not like. But another conclusion was reached and that was that the testers played without thinking about the minimum 10 minutes and played the whole game. Even if the testers were acquaintances, we can not ignore most of them stated they thought they had been playing for less time than what they saw at the end when I collected their times, which concurs with the positive fun score we got in the survey. Therefore, just the length and difficulty of the final level were adjusted and the ideal time for the game went from 25 minutes to 30, which is closer to the testers average time too.

7. Conclusions

In this chapter we will talk about the different conclusions we have after all the project development.

7.1 Work done

So first of all, we want to talk about the obstacles and risks we were afraid of at the planning stage and how real they were. As we have explained in more than one of the previous sections, the tight schedule and setbacks were an obvious risk and obstacle that we managed to overcome thanks to the methodology we chose. Then, the fear of using slow algorithms or methods for the demanding videogames software was a true concern at first when due to some bad ideas and some bugs the map creation was slow, but at the end the chosen methods have proven to work nicely. Furthermore, tests have shown how the generators can create levels with more than 400 rooms (which is a number far greater than the actual game will need) in less than 10 seconds for example. The last important risks considered were possible errors in the code, which can still be a threat even after all the fixings done thanks to the validation, and the possibility of the solution we have found to be biased to our personal tastes in games, that until the validation process with testers we were not be able to refute.

The next conclusion we would like to discuss is how the investigation of other works has been much more important than we thought at first. The context and state-of-the-art of the game helped to understand much better what we were trying to solve, to see different approaches other people have tried and to get great ideas for creating our own method. We think that our approach may be quite unique and even if other ways may have worked better against this problem, we have investigated something new and we will be able to show what we have learnt from it.

Consequently to these conclusions and the good results of the validation process, we would like to say that this project has been a success at achieving what we planned. So we have a project that approaches some not too investigated topics, tries new ways to solve the selected problem nicely (if we accept the results of our little validation process as good) and creates the base for some really interesting possibilities. We also want to admit that the project is limited by the selected scope and the development time, but we are truly happy about the result we achieved under these conditions.

After explaining our conclusions about the work done, we would like to show in the next section the interesting possibilities of improvement and future work for this project.

7.2 Possible improvements

First, we would like to explain the obvious improvements that would make the game better but we consider that would not be too interesting. We mean improvements like:

- Better graphics, including better textures, better animations and more visual feedback.
- More audio diversity, specially in sounds where we only have 4 different ones for all the action in the game.
- More enemies, with different behaviors and even final bosses for the levels.
- More items, like potions and shields.
- More map and other upgrades.
- Utility for the gold, being able to buy other upgrades for example.
- Better story, not just an excuse to explore the levels.
- More simple mechanics, like being able to sprint.
- Making a mobile version of the game.

On the other hand, there are some improvements and new paths of work for the project to get better levels and interesting results. Many of those are already prepared in the code to be worked on but were not started because of the scope and the time as we said. Those improvements would be:

- Adding cycles to the map graph and its generation so there could be multiple paths that still followed the rules like having the key before the lock.
- Not using rectangular rooms. For instance a room could have 3 tiles and an L shape.
- More kind of movements, like using stairs or climbing walls.
- More metrics for knowing how a room is being created. The current project mainly checks if the exits are connected or if every platform is connected, but we could also check the number of different paths between exits or the number of connections in those paths to play with how the rooms are created.
- A companion or enemy that follows the player using the movements graph of the room contents. Any kind of automatic playing (this could also be a tutorial or a power-up for the player) using all the information in the connections to move around on the platforms would be interesting.

8. Annexes

8.1 Video game user test

Test number:
Tester name:

Context: (please read before playing)

This is the user test for the result of my Bachelor Thesis on "Procedural generation applied to a video game level design". This test will have two parts: the video game test (that consists in playing for at least 10 minutes) and the survey about the test.
Before playing the game, let us explain a bit of context of it. As the Thesis title shows, this game is the result of an investigation on creating levels automatically. Therefore, it tries to be fun but it may lack in some aspects (such as the graphics for example) because it is not a professional video game. Moreover, this is not the final version of the game and still has some placeholder elements like the music and sounds.
Now, you can proceed to playing the game for at least 10 minutes, so you can answer the following survey and help validate the results of the project.

Survey: (please do not read before playing)

Answer the first four questions with a score from 1 (not at all) to 5 (very much).

Was the game fun to play?
Did you find diversity between the levels?
All rooms are really rectangles and the game tries to hide this fact (the first level does not try to hide it so do not think about it when you answer). Did you find that this was well concealed?
Did you find the levels to be different from levels of other games you have previously played?

If there was anything you enjoyed, what was the thing you liked the most?

If there was anything that bothered you, what was the thing you liked the least?

Any errors you have found?

8.2 Development images log

Here you can see in images how the project followed the plan and how everything was implemented.

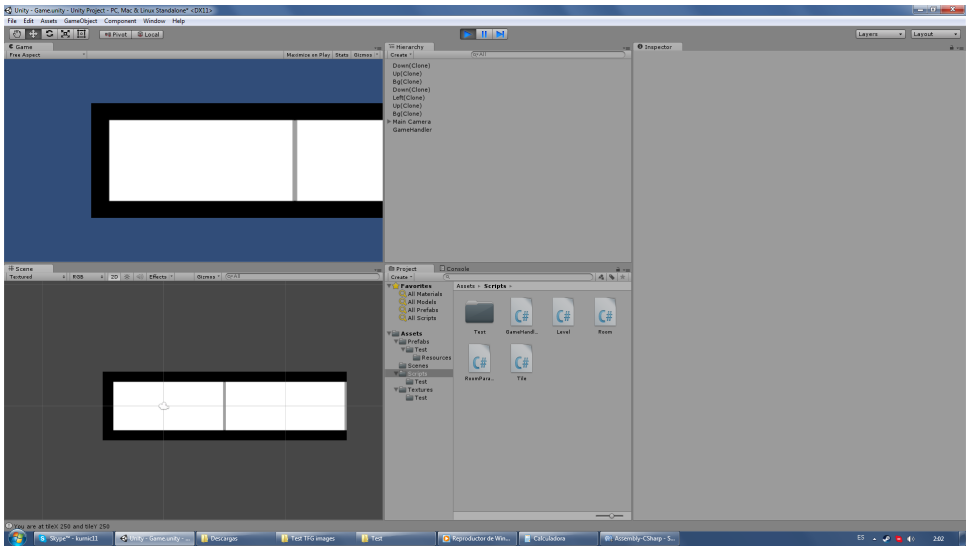


Figure 8.1: Development image 1.
Initial 2 rooms created with wall detections. (February)

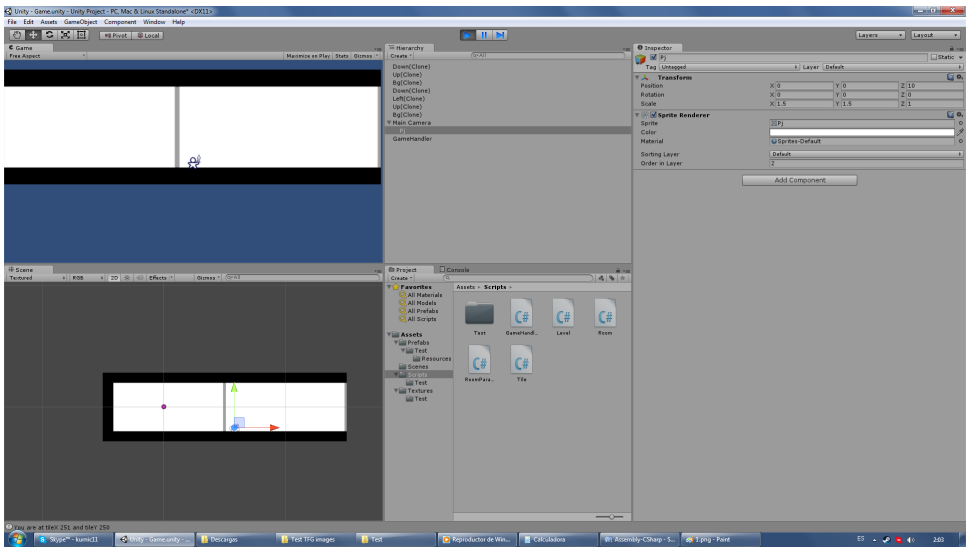


Figure 8.2: Development image 2.
Moving hero created for exploring the rooms graph. (February)

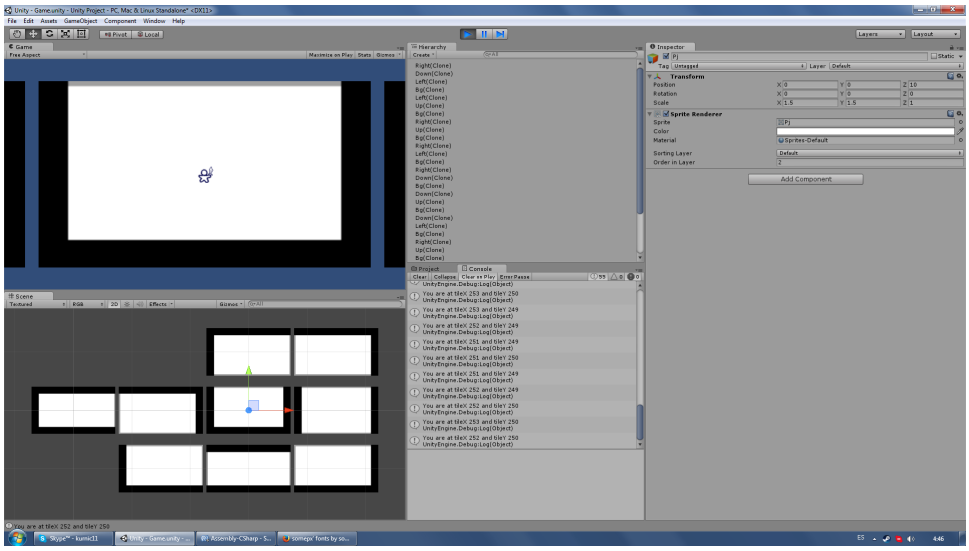


Figure 8.3: Development image 3.
First labyrinth with rooms of size 1 and 1 entry and 1 exit. (February)

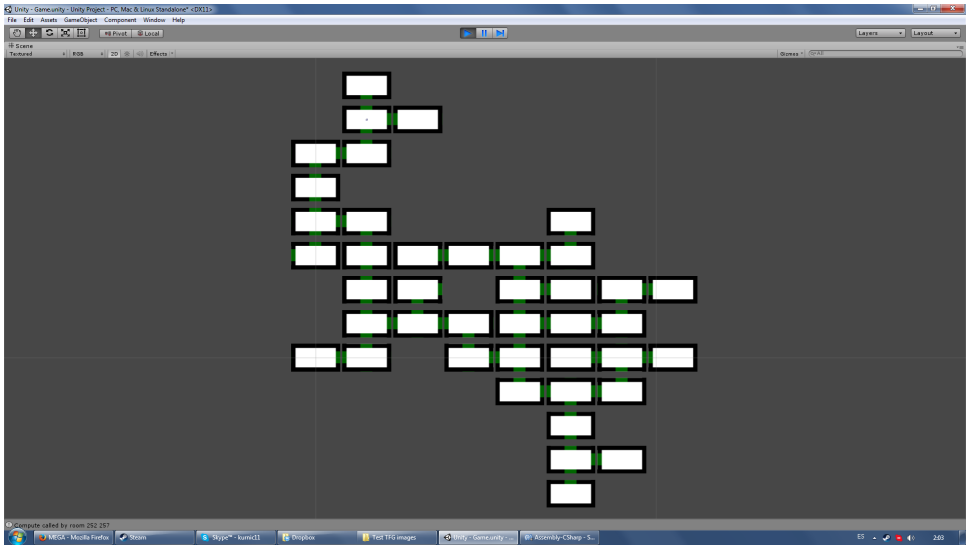


Figure 8.4: Development image 4.
Detection of entries on walls and first labyrinth with more than 1 exit. (February)

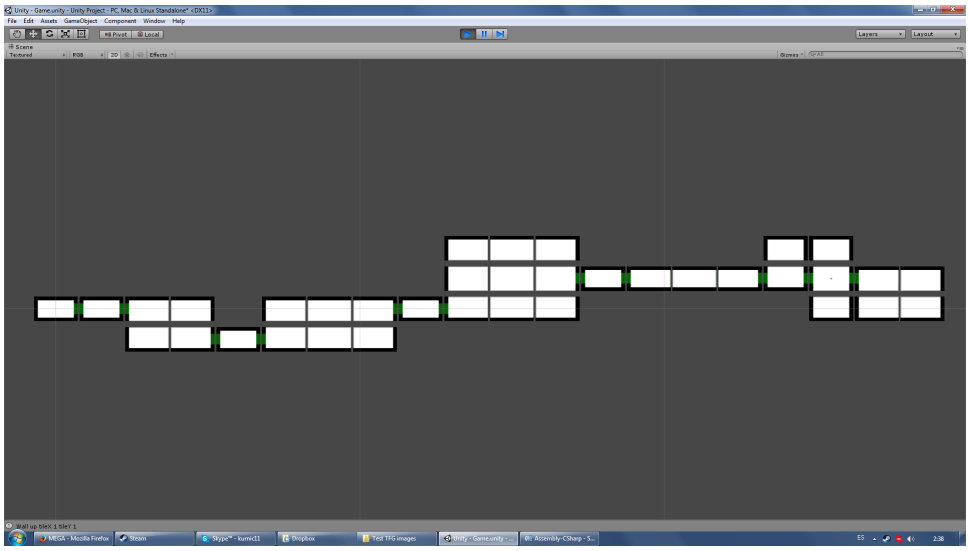


Figure 8.5: Development image 5.
Rooms with different sizes implemented but just in one direction. (February)

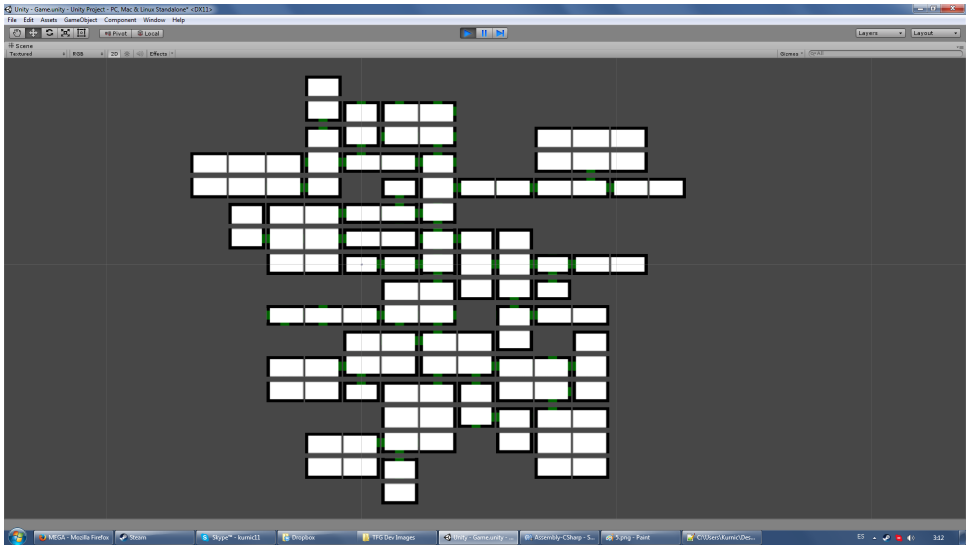


Figure 8.6: Development image 6.
Labyrinth limited to depth 5. Rooms with different sizes and paths in all directions, random number of entries and exits between 1 and the limit of the room. Some obvious bugs can be observed like. (March)

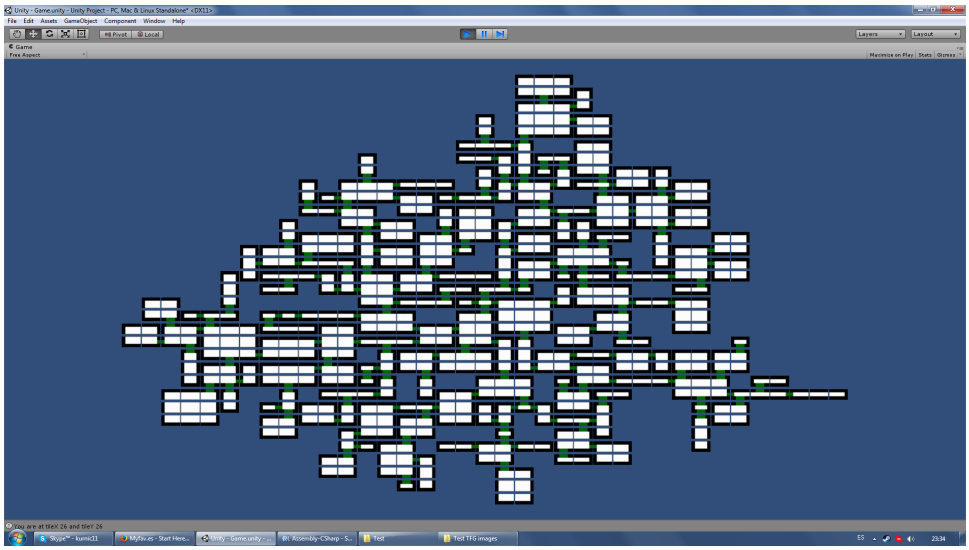


Figure 8.7: Development image 7.
Labyrinth limited to depth 8. Without bugs and bigger walls. (March)

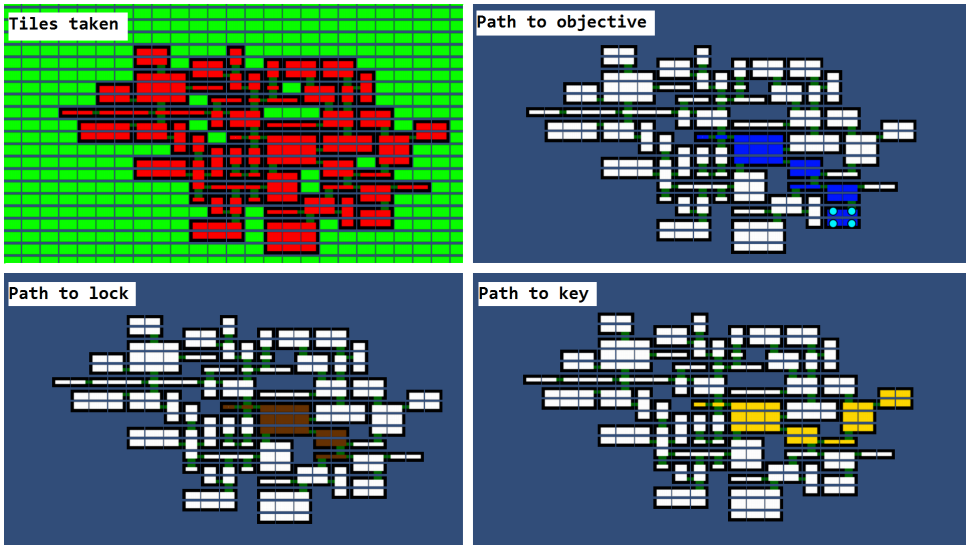


Figure 8.8: Development image 8.
Limited depth to 5. First steps on inheritance of room parameters. (March)

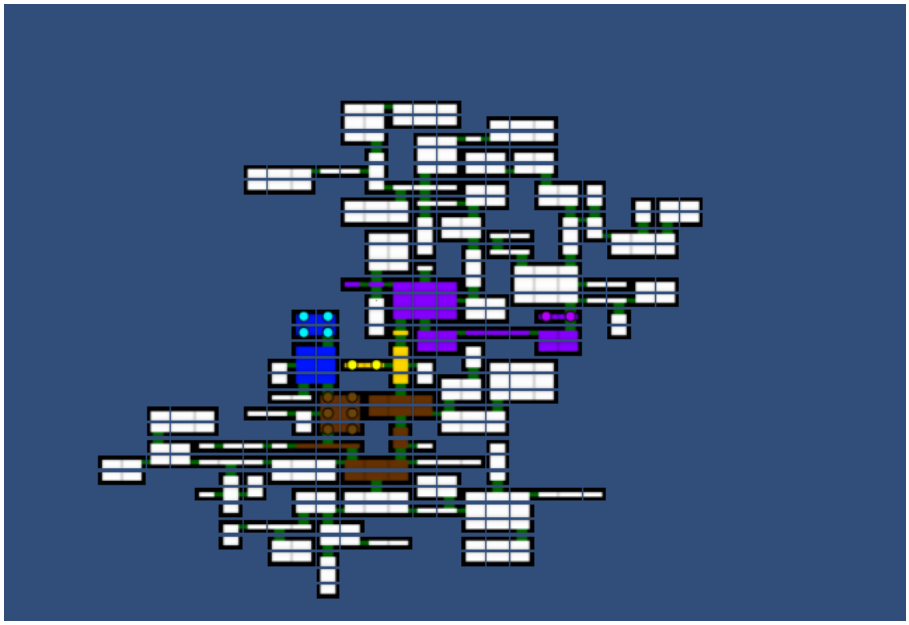


Figure 8.9: Development image 9.
Limited depth to 10. Starting to cut branches when they aren't going to the objective. Inheritance has some more logic now. Added getting and needing a power to the inheritance. Purple is for getting a power, yellow for a key, brown for a door, blue for the objective. (March)

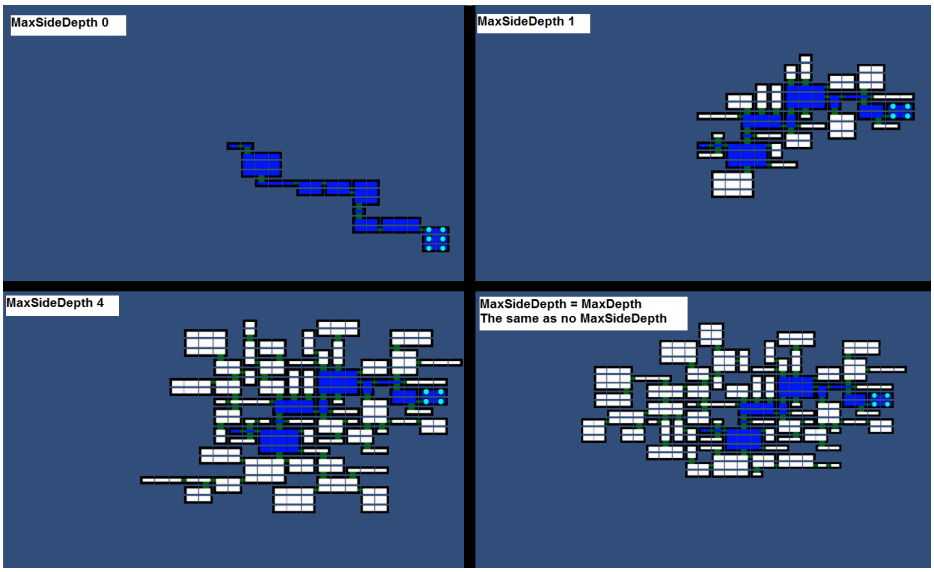


Figure 8.10: Development image 10.
Limited depth to 10. Added MaxSideDepth, which is the depth from the main path.(March)

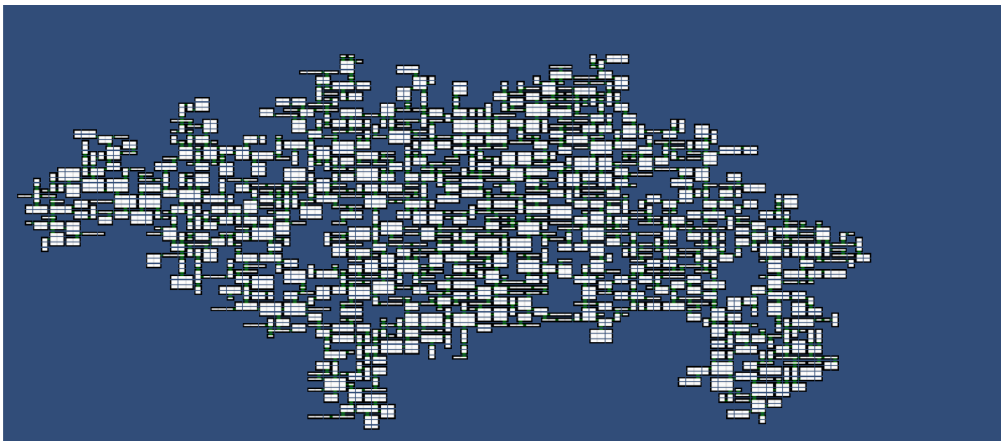


Figure 8.11: Development image 11.
Limited depth and sidedepth to 50. Less than 2 seconds generating and showing the map. (March)

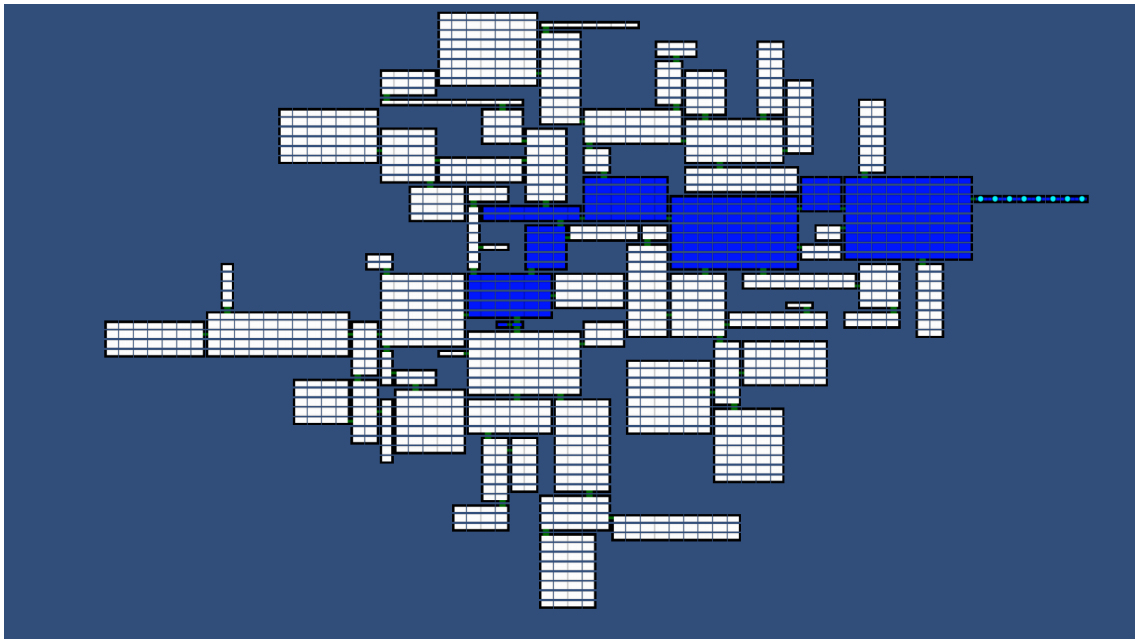


Figure 8.12: Development image 12.
Testing bigger rooms (10x10 tiles maximum). (March)

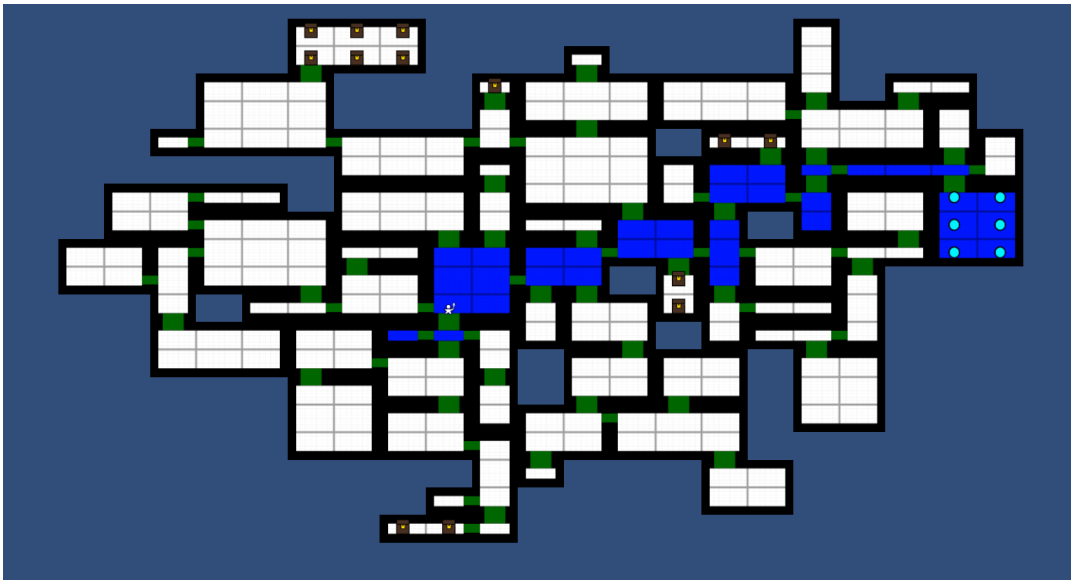


Figure 8.13: Development image 13.
Added treasures to the map. Improved the test images of the map to fit better. Tiles new ratio of 25:15 (March)

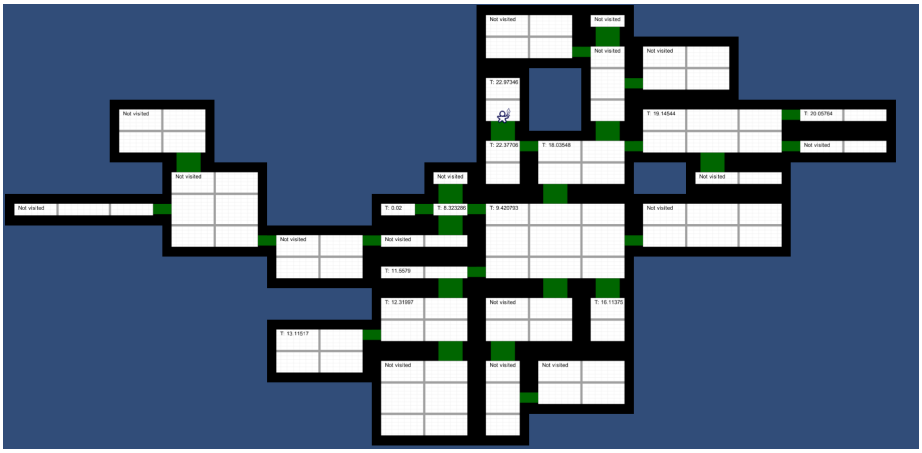


Figure 8.14: Development image 14.
Visiting a room for the first time saves the time when you entered. (March)

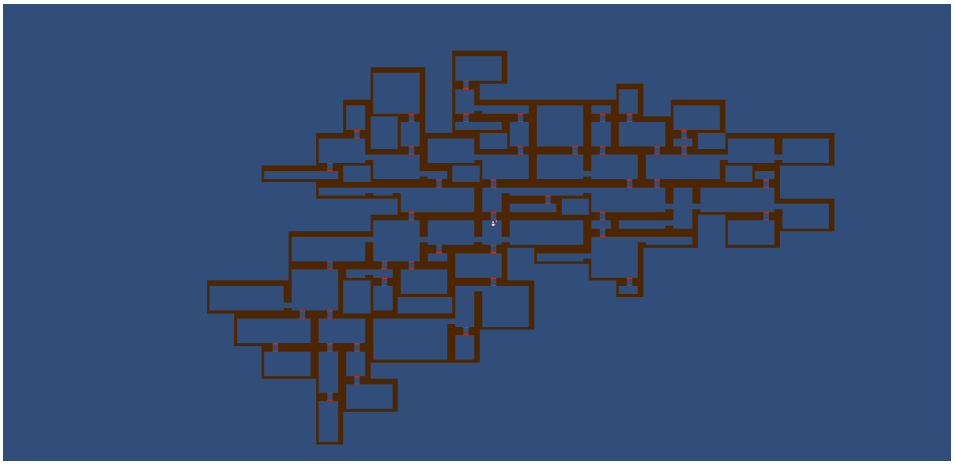


Figure 8.15: Development image 15.
First prototype of the room content generator. It generates and draws the hitboxes of the basic walls. In brown we can see walls and in red the platforms. (March)

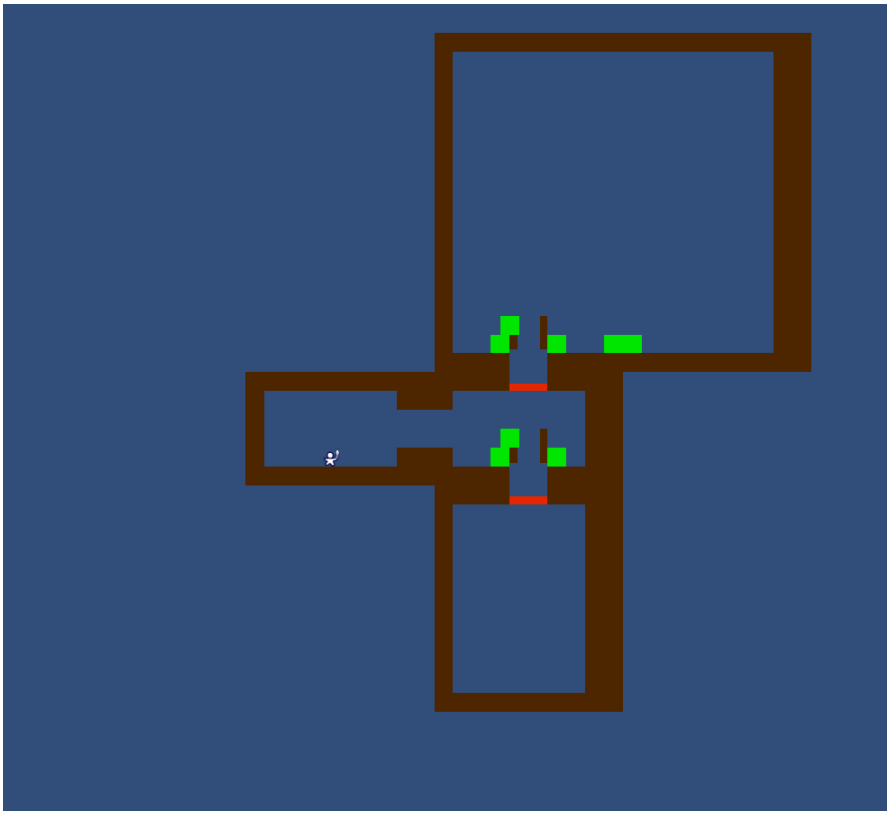


Figure 8.16: Development image 16.
Room content generator advancing. After generating the basic as before it checks how to go from one wall/platform to another. This is a test where we can see the green hitboxes seeing the character can jump from a side to the other of the holes because there are some obstacles I put myself before. (April)

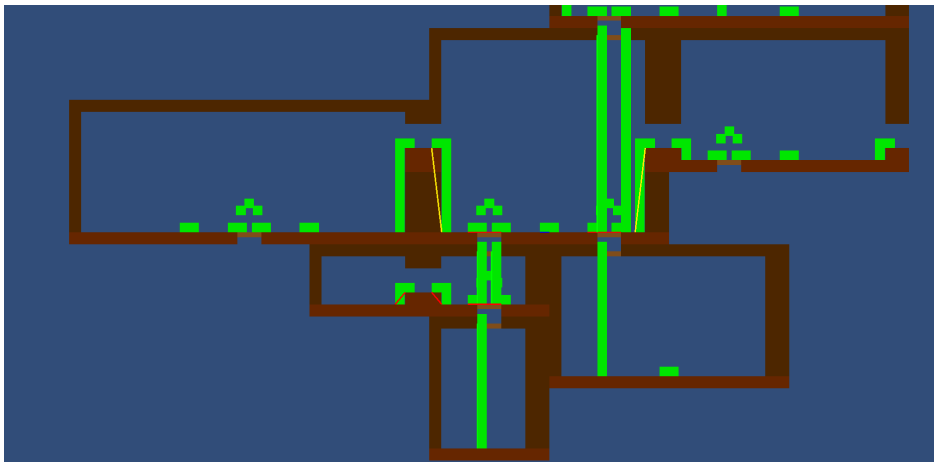


Figure 8.17: Development image 17.
New connections detected. We can still see the green hitboxes for the checking and we can see some connections drawn. Red lines are Jump connections, blue are Walk connections, green are Drop and yellow are Fall. (April)

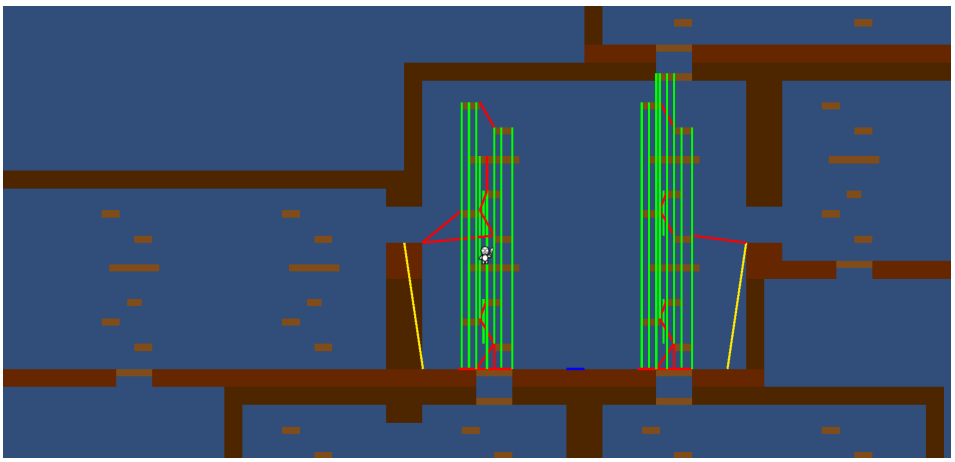


Figure 8.18: Development image 18.
Example room with all the connections detected. Red lines are Jump connections, blue are Walk connections, green are Drop and yellow are Fall. (April)

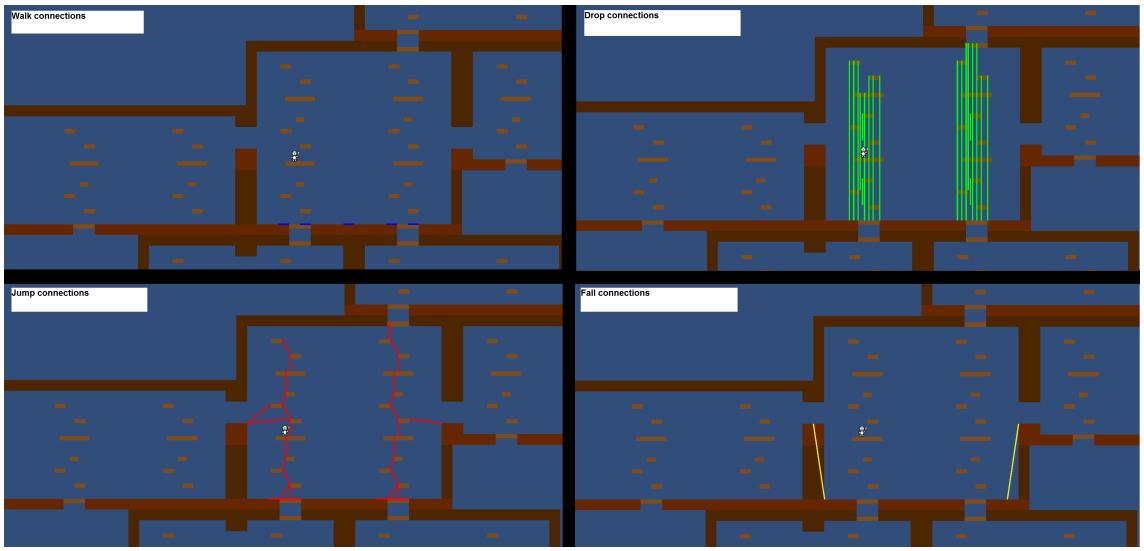


Figure 8.19: Development image 19.
Same connections detected but drawn by turns for a better understanding. (April)

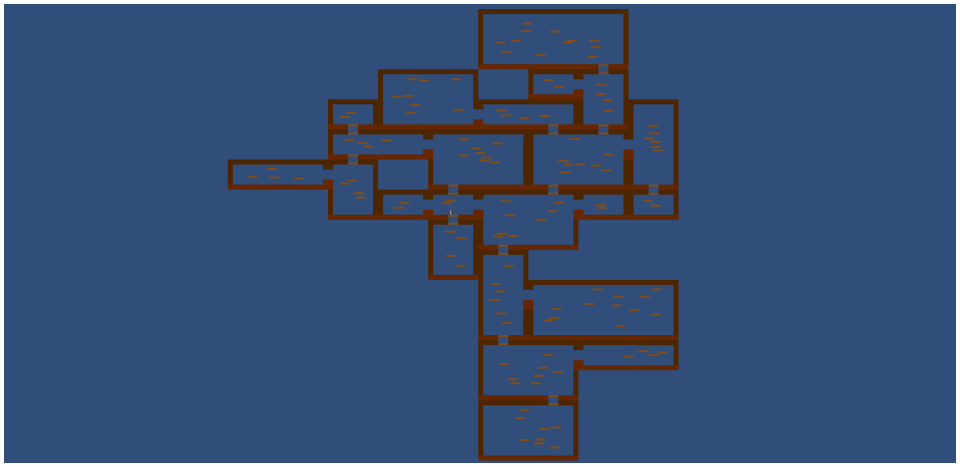


Figure 8.20: Development image 20.
First content generator. Really simple: base walls + totally random inside with an arbitrary number of platforms that we only check to not collide between themselves. Exit accesibility (or room feasibility) is not checked yet. (April)

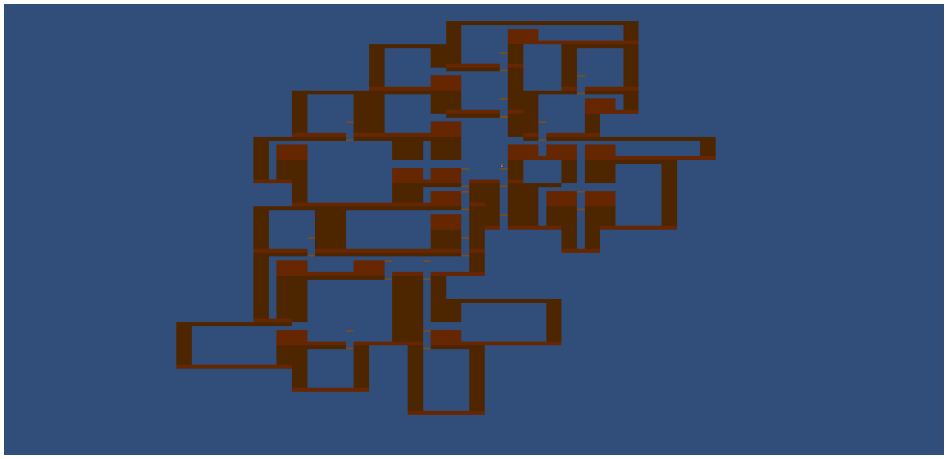


Figure 8.21: Development image 21.
Inside generator off. Playing with irregular constants for the base proves that the base patterns can be hidden a little. In this case for example, pits were created! (April)

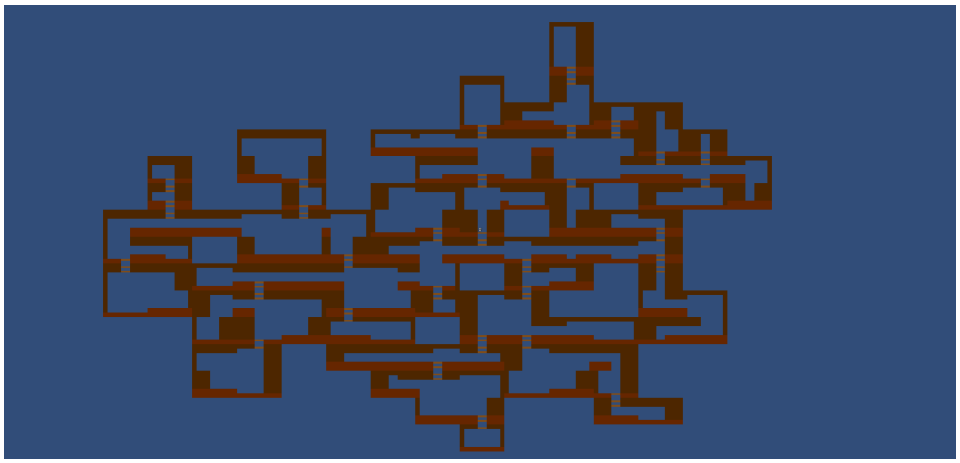


Figure 8.22: Development image 22.
Inside generator still off. Adding some randomness to the basic wall generation creates some cool unexpected shapes. (April)

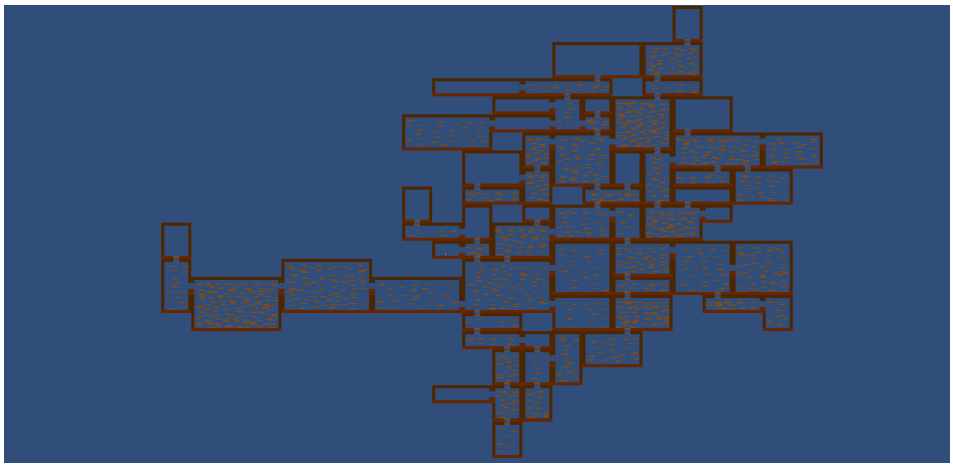


Figure 8.23: Development image 23.
Random content generator that takes exit accesibility (or room feasibility) into account, although rooms are not interesting they can be completed. Test on simple walls. (April)

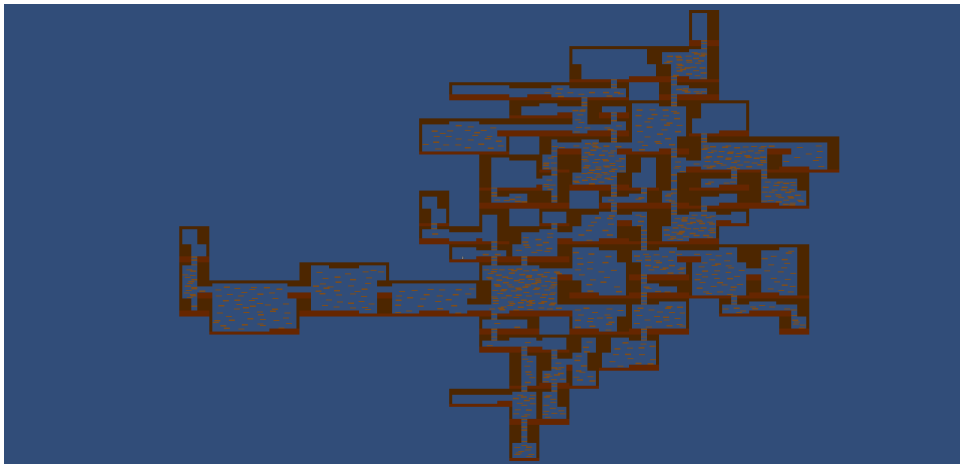


Figure 8.24: Development image 24.
Same test as in 23 with randomness to the basic wall generation. (April)

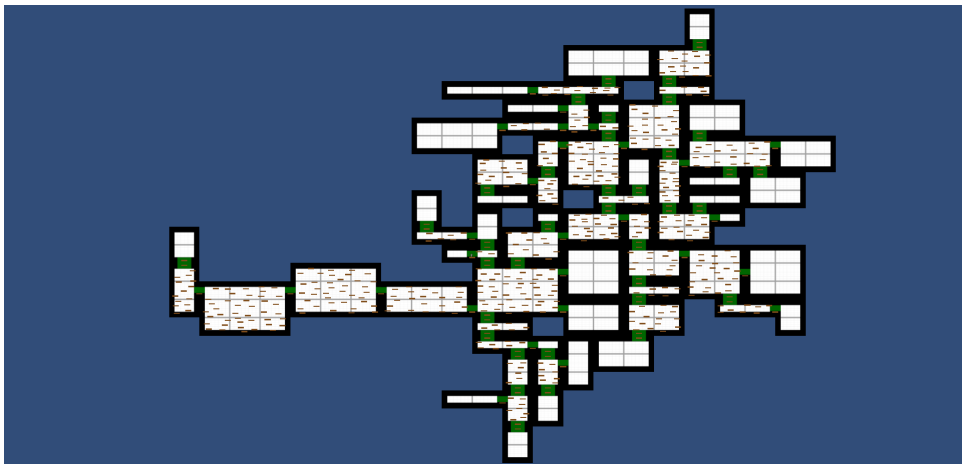


Figure 8.25: Development image 25.
Playing with the idea of a level with no walls and the generator works (maybe even better than with walls). (April)

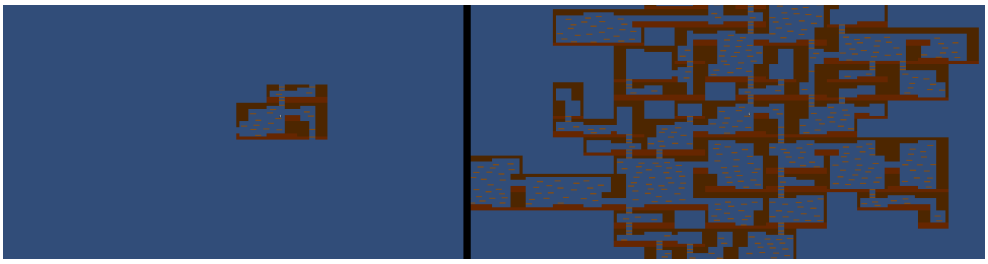


Figure 8.26: Development image 26.
Now only the visible rooms for the camera are rendered and active. This image shows from another point of view how to the left (new) we can only see some rooms while in the right (old) all the rooms were active and rendered. We can still see all the rooms with a debug button. (April)

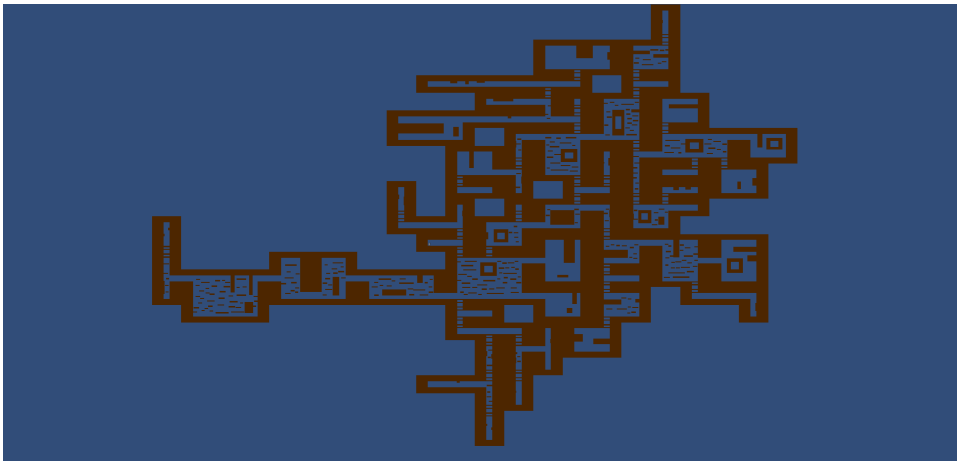


Figure 8.27: Development image 27.
Random content generator already using basic parts such as the donut to create more interesting rooms. Some zones are difficult to see that they are still rectangles, which is great! (to aid this effect the different wall colors was deactivated) (April)

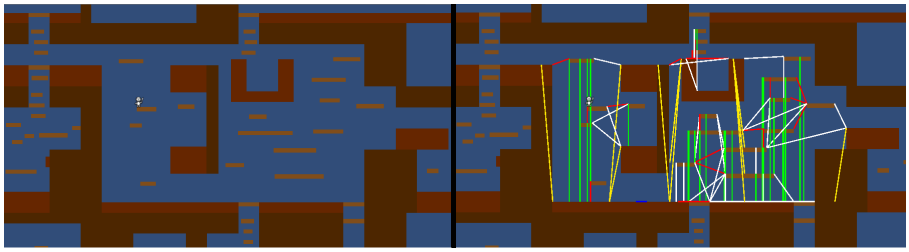


Figure 8.28: Development image 28.
Finally completed the double jump connection detection. Even if it works well it doesn't seem to help making interesting levels. It will be easy to activate but for now it will be disabled as levels without it seem more fun. And even if it can't be seen. The different platforms that depend on power-ups have been implemented too in the content generation. (April)

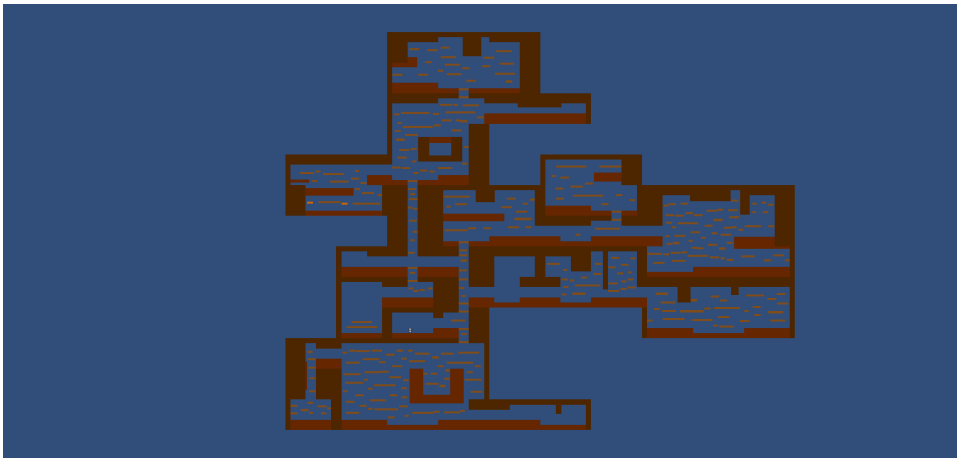


Figure 8.29: Development image 29.
Different types of levels have been implemented. The next images will be the first tests of those different types. This is the factory level for example (it has lots of weird shaped rooms). (April)

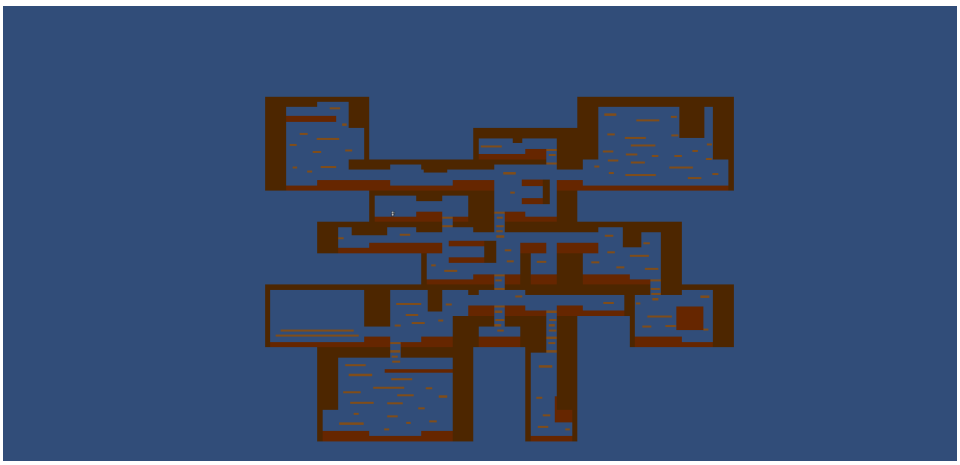


Figure 8.30: Development image 30.
This is an underwater level (it has a lot of space). (April)

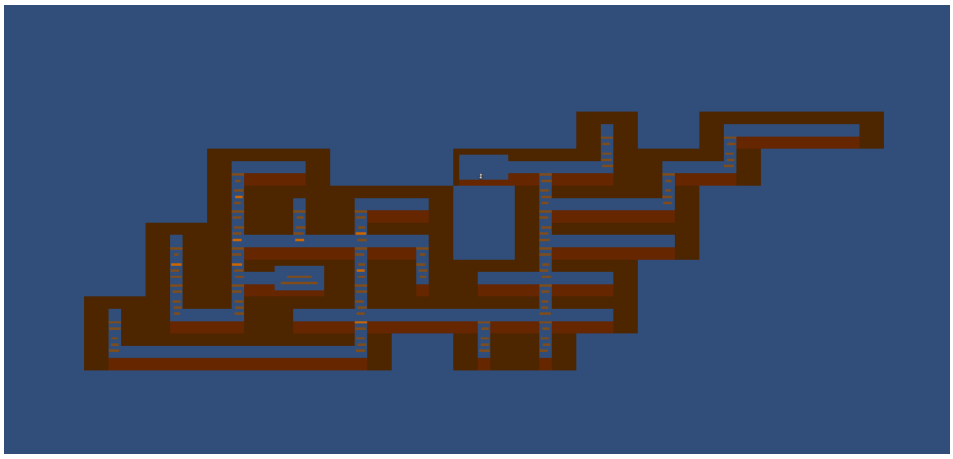


Figure 8.31: Development image 31.
This is a labyrinth level (it is completely made of tunnels). (April)

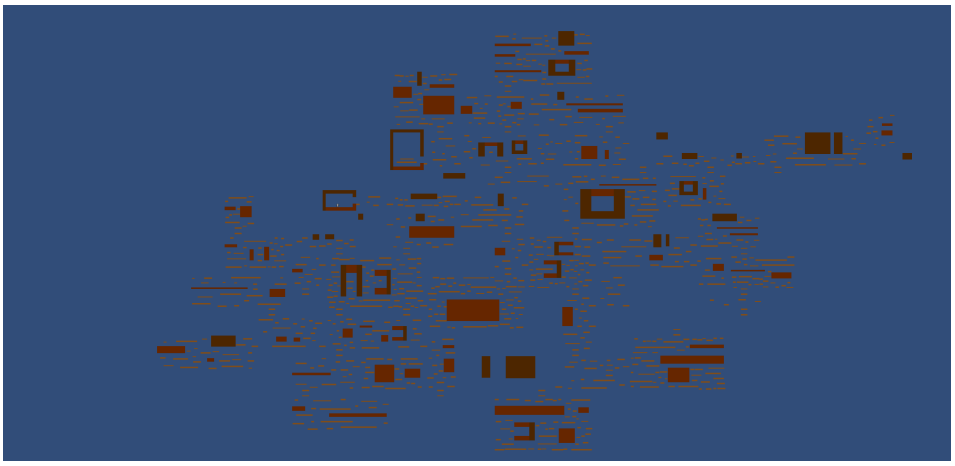


Figure 8.32: Development image 32.
This is a aerial level (it does not have walls around rooms). (April)

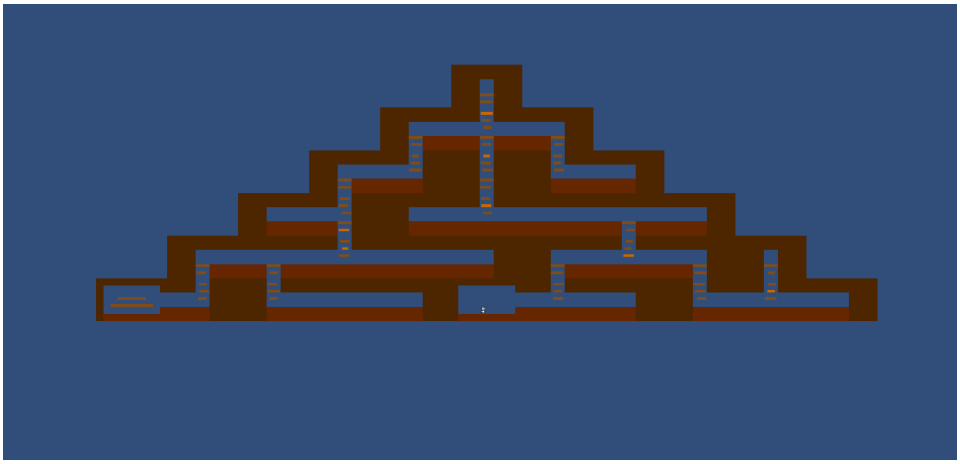


Figure 8.33: Development image 33.
Added the level shapes for different levels. This is an examples of the pyramid shape forced into labyrinth levels. (May)

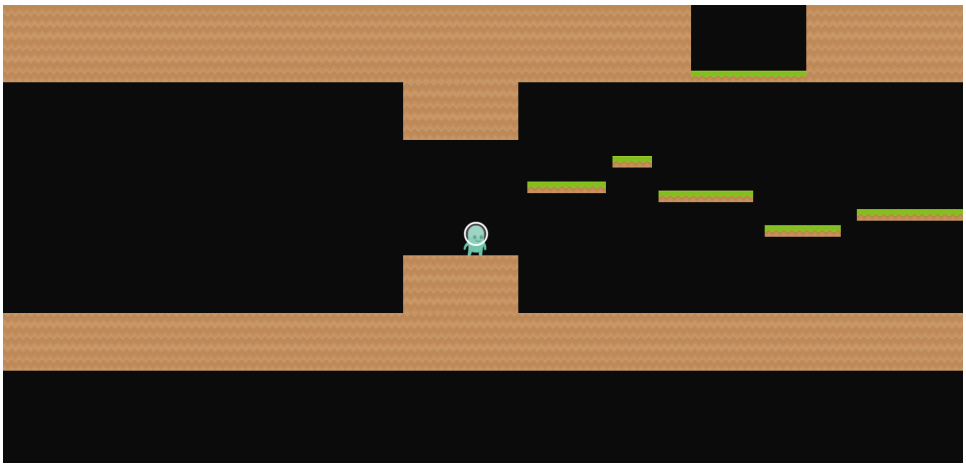


Figure 8.34: Development image 34.
Graphics implementation. Walls have textures and the hero is an animated sprite. (May)

Figure 8.35: Development image 35.
First items implemented. 7 different types of money items added. (May)

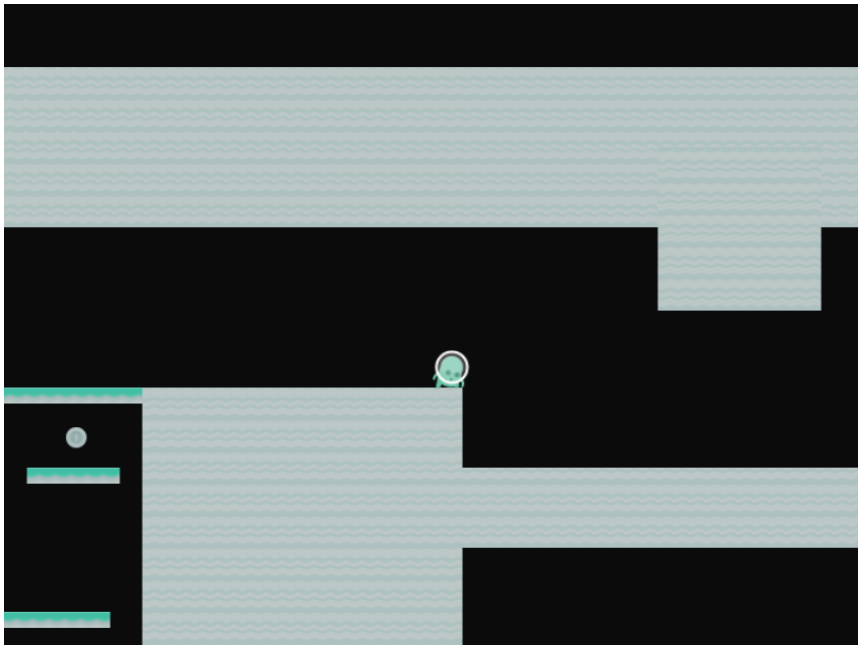


Figure 8.36: Development image 36.
Different types of levels have different types of graphics. (May)

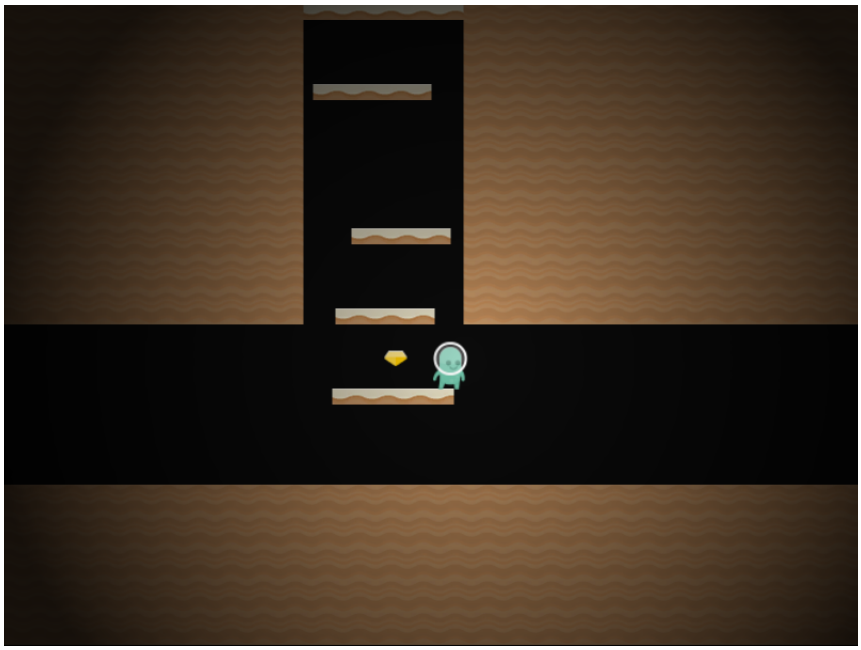


Figure 8.37: Development image 37.
Labyrinth levels have a light around the character for a better ambient. (May)

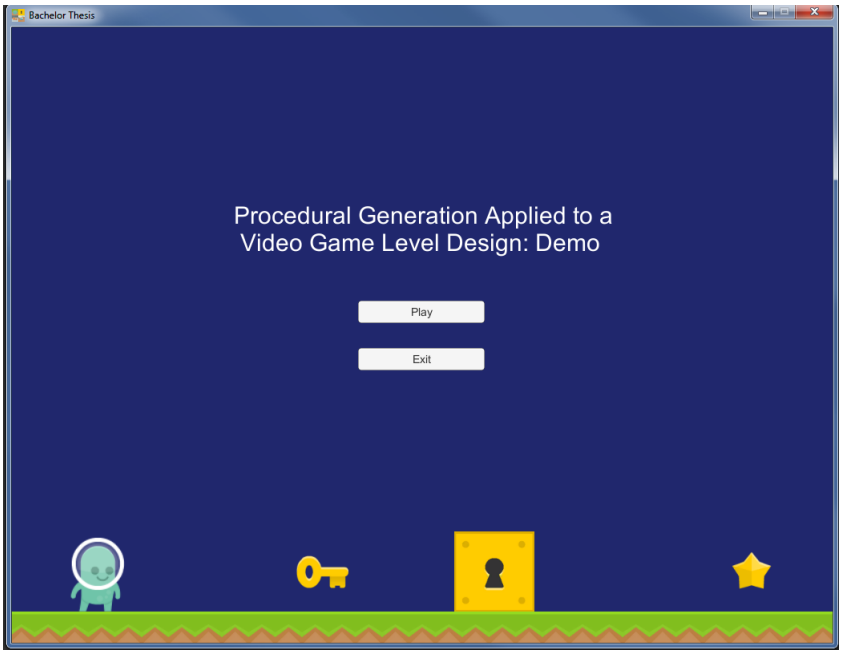


Figure 8.38: Development image 38.
First menu graphics implemented. (May)

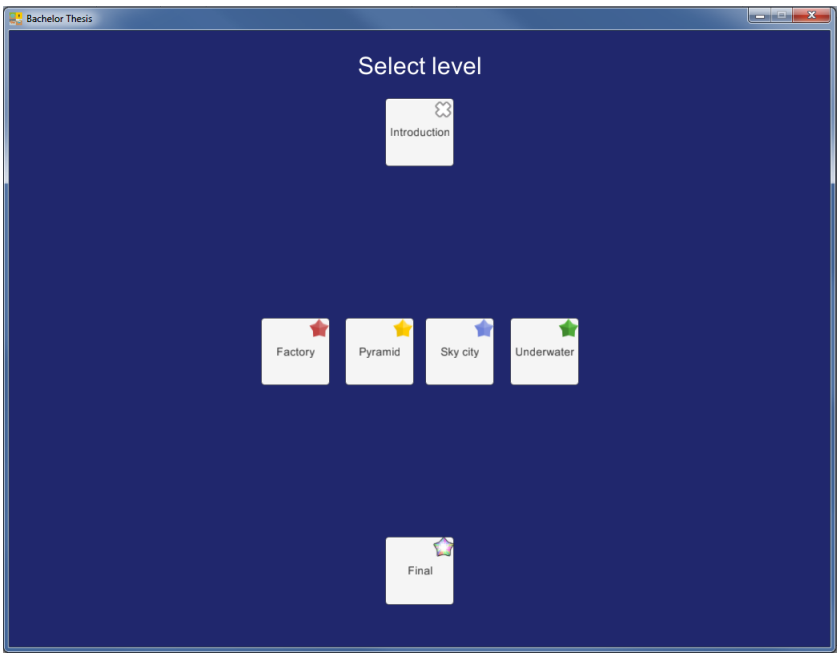


Figure 8.39: Development image 39.
And a level selection screen has been created too. (May)

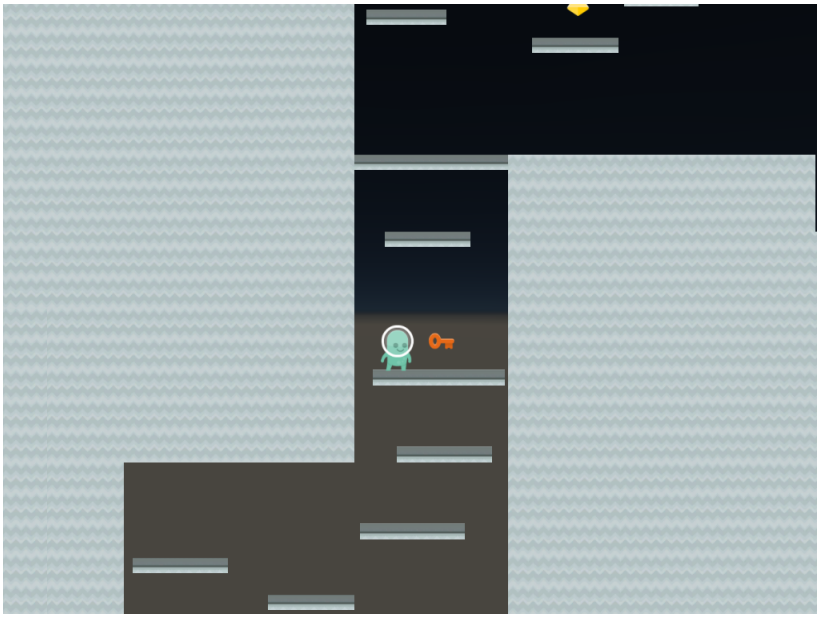


Figure 8.40: Development image 40.
Key items implemented.(May)

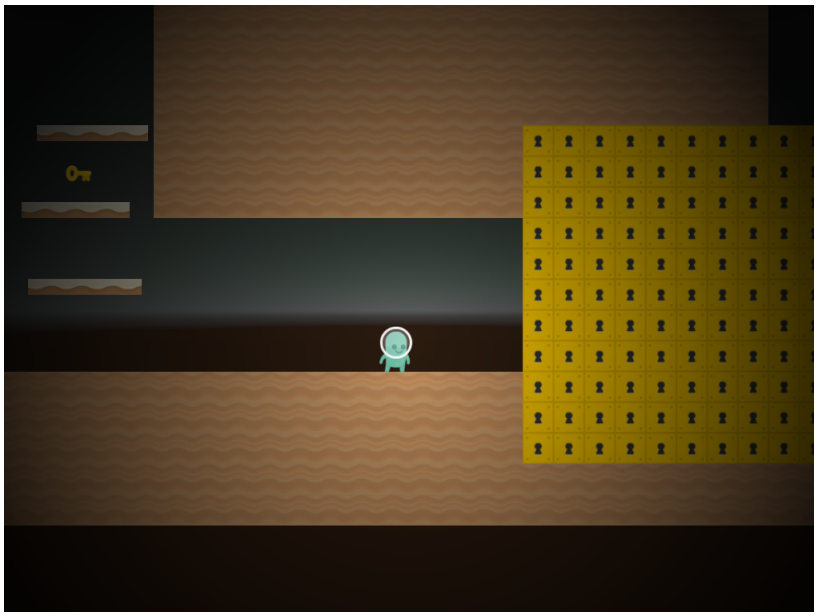


Figure 8.41: Development image 41.
Locks that unlock when you collect the level key implemented. And powers started its implementation here even if you can not see it. (May)

Figure 8.42: Development image 42.
Added a notification text that tells you what items you collect. +50 on money has been just collected here as example. (May)

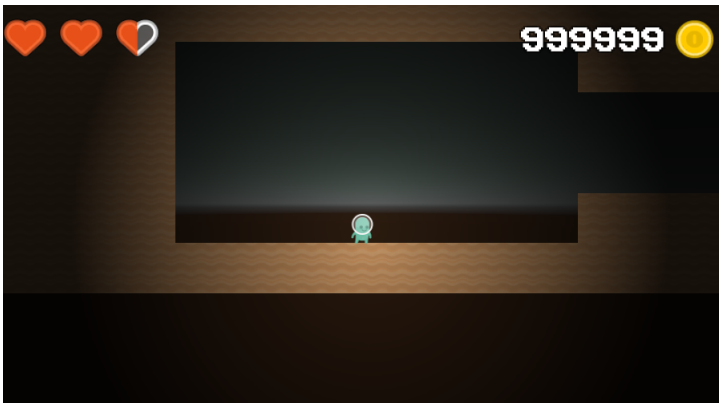


Figure 8.43: Development image 43.
HUD added showing your six lives (half a heart per life) and the money you have collected so far. (May)

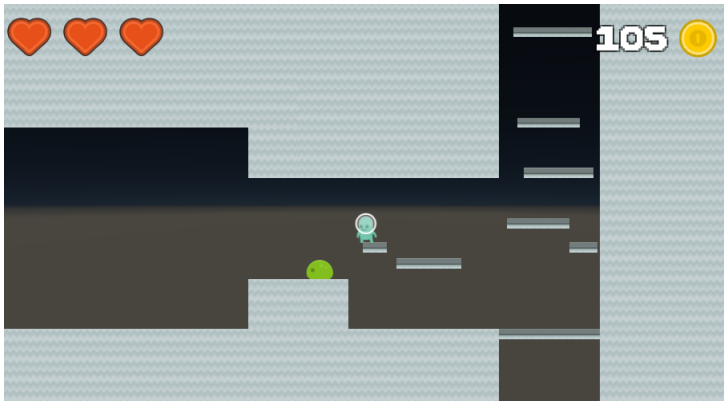


Figure 8.44: Development image 44.
First enemy added. Just a slime that walks a little bit. You can kill him by jumping on him. (May)

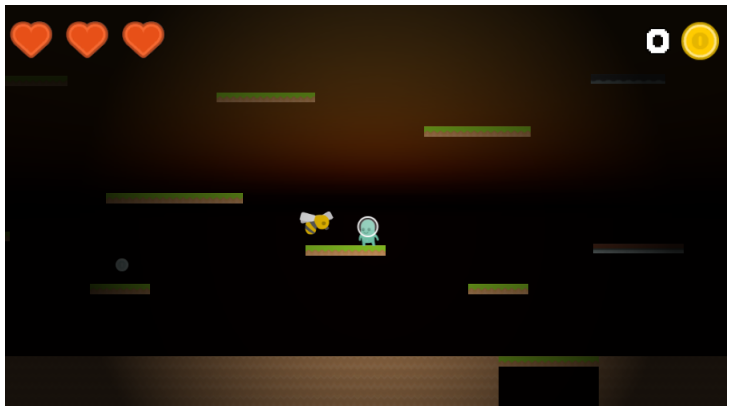


Figure 8.45: Development image 45.
Flying enemies added. They are bees on most of the levels but they transform into fishes in the underwater level. (May)

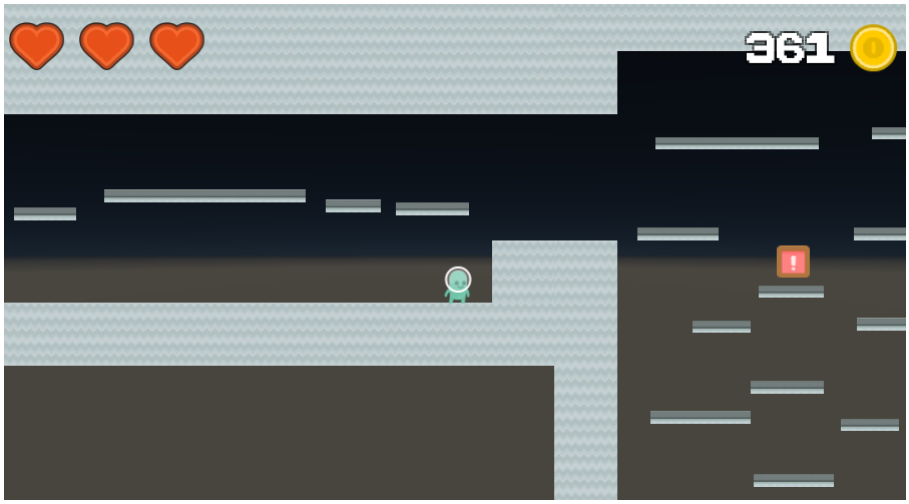


Figure 8.46: Development image 46.
Powers totally implemented. In this image you can see a red box that gets you the fire resistance power when you collect it. (May)

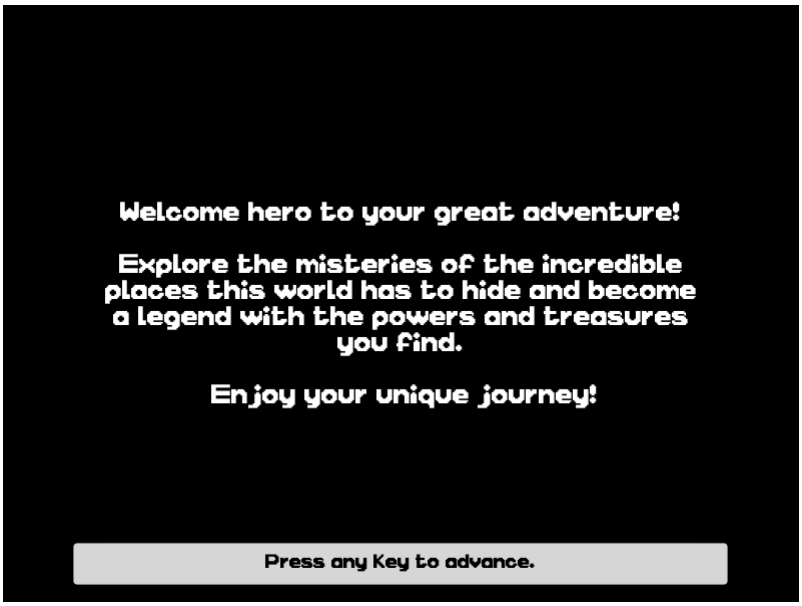


Figure 8.47: Development image 47.
Story transitions implemented. (May)

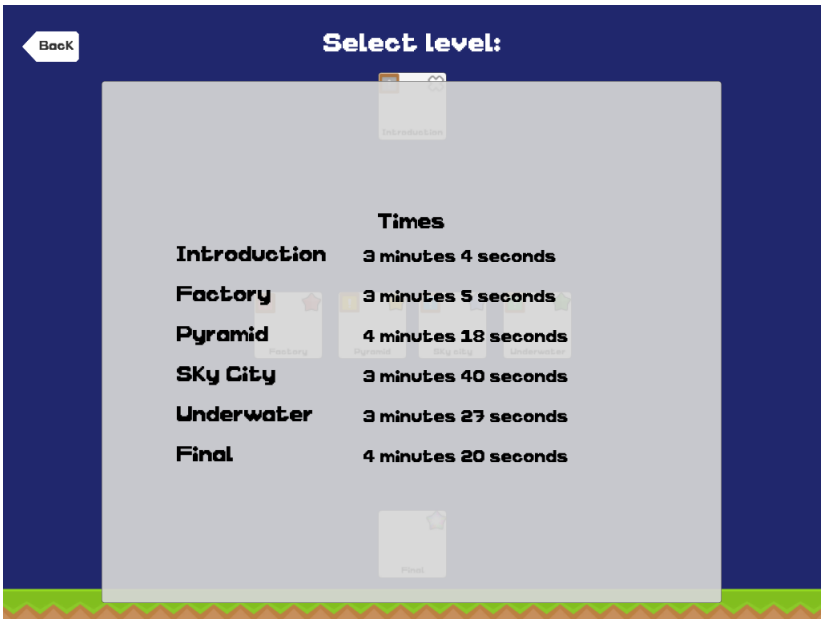


Figure 8.48: Development image 48.
Testers version finished. It collects the time played in each level automatically. (May)

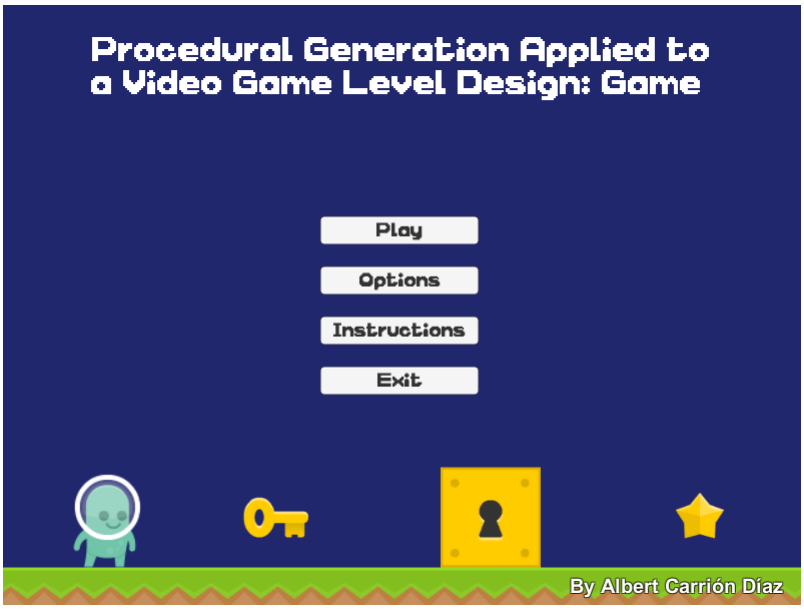


Figure 8.49: Development image 49.
Game finished. (June)

8.3 Game Design Document

A game design document is a living design document made specially for a video game. It is created during the development of the video game and changes over time, but it always organizes everything that has already been thought and puts together the effort and ideas of all the development team so it is written down and clear. It is really important to use one to maintain the cohesion and consistency around the game.

In the following pages, we will show the game design document that was made to plan the video game part of the project. This document was made when the map and level generators were almost finished and has been changed while developing, as new ideas became part of the project.

Procedural generation applied to a video game level design.

Game Design Document

Albert Carrión Díaz

A. Introduction

This game is the result of a Bachelor Thesis trying to apply procedural generation to video game level design.

A.1 Synopsis

The player starts at a room where he is pointed to go find a treasure. When he finds the treasure, the real adventure begins on some procedural generated levels.

A.2 Genre

The game will be 2D side-view platformer. With really simple controls and mechanics, it will strongly focus on the levels exploration and obtaining the requirements (like new powers or keys) to access new zones and advance.

A.3 Inspiration

This game has some big influences from the games "Rogue Legacy" and "Cloudberry Kingdom". "The binding of Isaac", "Spelunky" and "Unepic" have been other places where the inspiration for some ideas came from.

A.4 Purpose and target audience

The real purpose is to create a fun game that will take advantage of all the work on the procedurally generated levels. The target audience will be mainly fans of dungeon exploration and simple recollection platformers with experience with video games, but I will try to make the difficulty of the game depend on the challenge the player is searching.

A.5 Game flow

You start in an introduction level and fight till the first objective and get to know the story. Then you play in a random order the different levels of the different zones collecting at the end of each one a star od a different color to unlock the final zone.

A.6 Style

The style of the game will be non-serious or even cartoon graphics and audio, nothing realistic or complex because we are just trying to do something fun. With this objective the player will have to search different unexplored zones full of enemies and treasures.

A.7 Objective

Finishing each level that will be like a small labyrinth with doors and such. All to collect the star at end of each level that lets you find the legendary white star and become a legend yourself.

B. Gameplay and mechanics

B.1 Gameplay

B.1.1 Adventure progression

You explore a forest for a great treasure and when you touch the treasure you see it is fake and start your real adventure. You must find the different stars to discover where the real treasure is. Each objective is in a final room of the levels, getting one finishes the level and getting them all takes you to the final level. The levels are 4 and you go to them in whatever order you choose:
-A factory.
-A sky city.
-An underwater level.
-The interior of a pyramid (labyrinth).
After the four different levels you go to a final dark level.

B.1.2 Difficult progression

Every level you play is larger and has more to explore with more enemies.

B.1.3 Adventure objective

You want to become a legend by finding the incredible white star of some really old stories.

B.1.4 Game flow

You start only being able to play the introduction level. When you get the fake white star of this level you unlock the next for levels. When you get the final star of each of the 4 levels you unlock the final level. When you get the real white star of the final level you have finished the game.

B.2 Controls

You can walk to both sides and jump as usual in platformers with a button for each of these actions. There is another button for crouching and then dropping from platforms.

B.3 Mechanics

B.3.1 Movement

You can walk and crouch but not at the same time. You can jump up and move while you are in the air. You can jump down from platforms while crouching. There will be a power-up that allows you to jump again on the air.

B.3.2 Combat

You can only hit enemies by jumping on them.

B.3.3 Items

There are coins and other little treasures to collect for money (for now only points). There are map upgrades (which make the exploration a lot easier) and the elemental power-ups that let you use platforms that had obstacles before (for example fire platforms hurt you unless you have the fire power).

B.3.4 Powers

Double jump and the elemental power-ups that let the hero use more platforms:
-Fire power-up to step on fire without damage.
-Wind power-up to step on clouds.
-Water power-up to jump on sticky platforms.
-Earth power-up to step on spikes without damage.

B.4 Enemies

Simple walking monsters that move horizontally, some others that move vertically and then the flying monsters that move in any direction.

B.5 Physics

Only "simple gravity simulation".

B.6 Map

You can check the map at any moment and see a layout of what you have explored by pressing pause. This map shows with each level of upgrade:
0. Where have you been.
1. All the rooms.
2. Where are the other map upgrades.
3. Where are the power upgrades.
4. Where are the keys of the level locks.
5. Where are the stars.

B.7 Rooms

Rooms is how the map is divided but we try to hide this fact by using different shapes and non-recognizable patterns.

C. Interface

C.1 HUD

You just see your lives, the money you collected and the game itself.

C.2 Screens flow

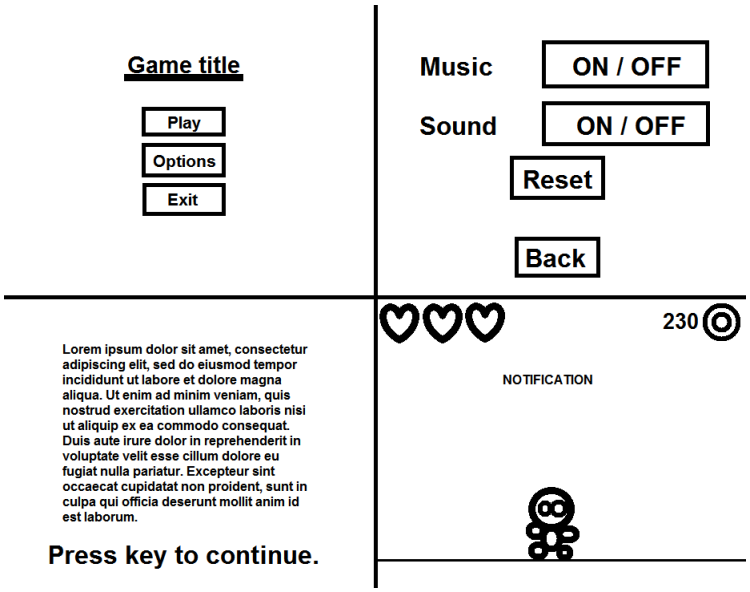


Figure C.1: Designs of the 4 screens of the game.[**?**]
Green means finished as planned and yellow finished with more time than expected.

C.3 Screens description

Main menu: Lets you select what to do (Play, options, instructions, credits or exit).
Options: Lets you change the configuration of the game, such as the music and sound or reseting your game progress.
Game screen: Where you play the actual game.
Story screen: Shows the story as you discover it or events like a game over screen when you die.

D. Story

D.1 Background story

You are a regular adventurer that wants to become famous.

D.2 Current world

You discover that the white star hides more mysteries than what you thought and that this could be the adventure of your life.

D.3 Objectives

Get the white star

D.4 Characters

Hero
White star

E. Technical decisions

E.1 Target hardware

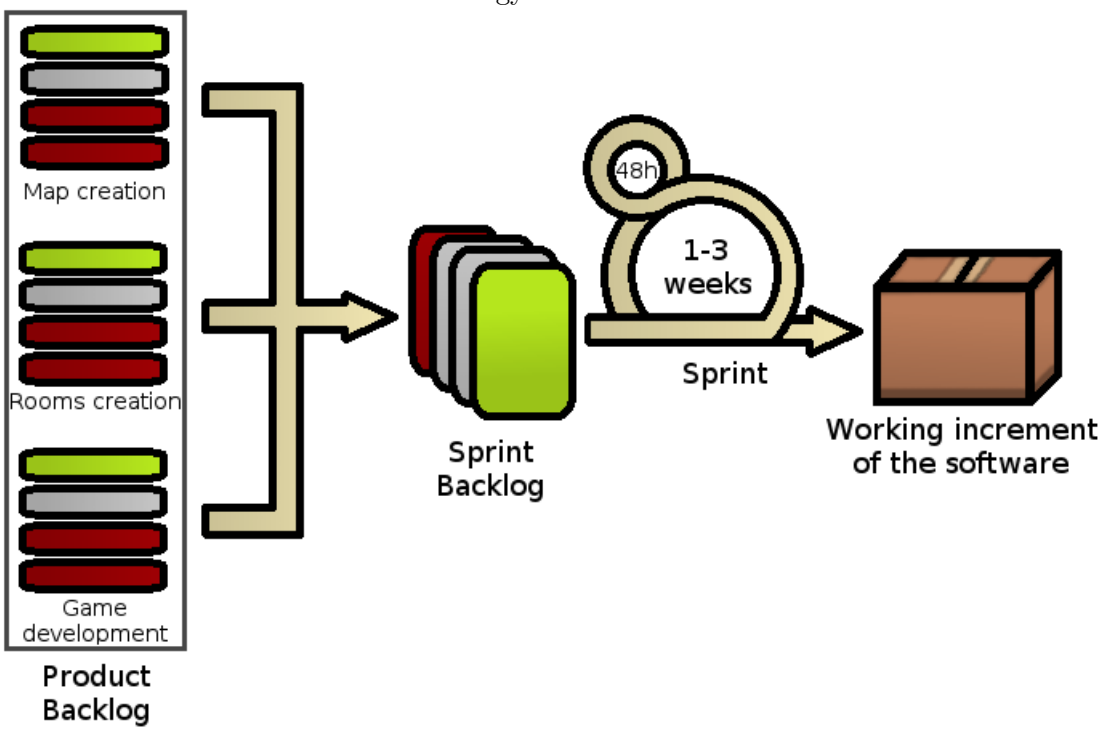
PC.

E.2 Development hardware and software

+5 year old PC with Windows 8.1.

E.3 Development methodology

Own variation of the scrum methodology.



F. Graphics decisions

F.1 Style

Cartoon and not realistic graphics. Really simple.

F.2 Graphics needed

Hero spritesheet.
Walking enemy spritesheet.
Verical enemy spritesheet.
Flying enemy spritesheet.
Coins and money.
Platforms and walls graphics.
Upgrades graphics.

G. Audio decisions

G.1 Style

There is no style for all the game. Whatever goes with each level will be used. For example, Arabic music could go well in the pyramid level.

G.2 Sounds needed

Walking sound.
Jumping sound.
Attack sound.
Getting hurt sound.
Getting item.

G.3 Music needed

Menu.
Introduction level.
4 levels.
Final level.

9. References

[1] Cellar Door Games. Rogue legacy. http://roguelegacy.com, 2014, retrieved on 02/03/2015.

[2] Albert Carrión Díaz. Own generated content, 2015.

[3] Unity Technologies. Unity 3d. http://unity3d.com/unity, 2015, retrieved on 02/03/2015.

[4] MonoDevelop Project. Monodevelop. http://www.monodevelop.com, 2015, retrieved on 02/03/2015.

[5] Linus Torvalds. Git. http://git-scm.com, 2015, retrieved on 02/03/2015.

[6] Atlassian. Bitbucket. http://git-scm.com, 2015, retrieved on 02/03/2015.

[7] The Gimp Team. Gimp. http://www.gimp.org/, 2014, retrieved on 02/03/2015.

[8] Audacity. Audacity. http://audacity.sourceforge.net, 2015, retrieved on 02/03/2015.

[9] Bay 12 games. Dwarf fortress. http://www.bay12games.com/dwarves, 2015, retrieved on 13/03/2015.

[10] Edmund McMillen and Florian Himsl. The binding of isaac. http://bindingofisaac.com, 2015, retrieved on 02/03/2015.

[11] Mossmouth. Spelunky. http://www.spelunkyworld.com, 2013, retrieved on 02/03/2015.

[12] Pwnee Studios. Cloudberry kingdom website. http://www.pwnee.com, 2013, retrieved on 02/03/2015.

[13] Jamis Buck. Maze generation: Algorithm recap. http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap#article_body, 2011, retrieved on 13/03/2015.

[14] Gray Lake Studios. Pro-d total - procedural dungeon generator total. https://www.assetstore.unity3d.com/en/#!/content/8553, 2015, retrieved on 02/03/2015.

[15] Wikipedia. Wang tile - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Wang_tile, 2015, retrieved on 13/03/2015.

[16] YoYoGame. Gamemaker: Studio. https://www.yoyogames.com/studio, 2015, retrieved on 02/03/2015.

[17] Epic Games Inc. Unreal engine. https://www.unrealengine.com, 2015, retrieved on 02/03/2015.

[18] Kenney. Kenney assets. http://kenney.nl/assets, 2015, retrieved on 10/06/2015.

[19] Eeve Somepx. Eeve somepx fonts. http://somepx.itch.io/pixel-fonts, 2015, retrieved on 10/06/2015.

[20] Ubisoft Entertainment. Ubisoft company website. https://www.ubisoft.com/en-US/company/overview.aspx, 2012, retrieved on 02/03/2015.

[21] Wikipedia. Scrum. http://en.wikipedia.org/wiki/Scrum_%28software_development%29, 2015, retrieved on 02/03/2015.

[22] Raph Koster. Raph koster's website. http://www.raphkoster.com, 2015, retrieved on 13/03/2015.

[23] Raph Koster. *Theory of fun for game design 1st ed.* (ISBN-13: 978-1932111972) 2004.

[24] Raph Koster. Theory of fun for game design's website. http://www.theoryoffun.com, 2015, retrieved on 13/03/2015.

[25] Alex Galuzin. *Ultimate Level Design Guide.* (eBook) 2011.

[26] Steve Swink. *Game Feel: A Game Designer's Guide to Virtual Sensation 1st ed.* (ISBN-13: 978-0123743282) 2008.

[27] Alex Galuzin. World of level design. http://www.worldofleveldesign.com, 2015, retrieved on 13/03/2015.

[28] Mateusz Piaskiewicz. Level design.org knowledge base. http://level-design.org/wiki/index.php?title=Main_Page, 2015, retrieved on 13/03/2015.

[29] Team Meat. Super meat boy! blog. http://supermeatboy.com, 2015, retrieved on 13/03/2015.

[30] Andrew Doull. Procedural content generation wiki. http://pcg.wikidot.com/, 2014, retrieved on 13/03/2015.

[31] Nathan Williams. An investigation into dungeon generation. http://www.nathanmwilliams.com/files/AnInvestigationIntoDungeonGeneration.pdf, 2014, retrieved on 13/03/2015.

[32] David Adams. Automatic generation of dungeons for computer games. http://www.dcs.shef.ac.uk/intranet/teaching/public/projects/archive/ug2002/pdf/u9da.pdf, 2002, retrieved on 13/03/2015.

[33] Roland van der Linden. Designing procedurally generated levels. http://repository.tudelft.nl/view/ir/uuid%3A1e24b455-728d-42c0-82c5-e55f64956161, 2013, retrieved on 13/03/2015.

[34] Procedural Content Generation Wiki. Mazes. http://pcg.wikidot.com/pcg-algorithm:maze, 2012, retrieved on 13/03/2015.

[35] Procedural Content Generation Wiki. Cellular automata. http://pcg.wikidot.com/pcg-algorithm:maze, 2014, retrieved on 13/03/2015.

[36] Nathan Williams. Youtube - cellular automata dungeon generation. https://youtu.be/sFP8TuAt_Jc, 2014, retrieved on 13/03/2015.

[37] Nathan Williams. Youtube - binary space partition dungeon generation. https://youtu.be/AUJx3xYM4n4?list=UU4i6i-HBjQzBC_pC7l8GB1A, 2014, retrieved on 13/03/2015.

[38] Nathan Williams. Youtube - delaunay triangulation dungeon generation. https://youtu.be/CaI6edoGbFY, 2014, retrieved on 13/03/2015.

[39] Phigame. Tinykeep. http://tinykeep.com, 2014, retrieved on 13/03/2015.

[40] Jordan Fisher. How to make insane, procedural platformer levels. http://www.gamasutra.com/view/feature/170049/how_to_make_insane_procedural_.php, 2012, retrieved on 13/03/2015.

[41] Julian Togelius. A comparative evaluation of procedural level generators in the mario ai framework. http://julian.togelius.com/Horn2014Comparative.pdf, 2014, retrieved on 13/03/2015.

[42] Wikipedia. Super mario bros. - wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Super_Mario_Bros., 2015, retrieved on 13/03/2015.

[43] Sean Barrett. Herringbone wang tiles. http://nothings.org/gamedev/herringbone, 2012, retrieved on 13/03/2015.

[44] Peter Mawhorter and Michael Mateas. Procedural level generation using occupancy-regulated extension. https://games.soe.ucsc.edu/procedural-level-generation-using-occupancy-regulated-extension, 2010, retrieved on 13/03/2015.

[45] Gillian Smith. Launchpad: A rhythm-based level generator for 2-d platformers. http://sokath.com/main/files/1/smith-launchpad-tciaig.pdf, 2011, retrieved on 13/03/2015.

[46] Epic Games Inc. Epic games website. http://epicgames.com, 2015, retrieved on 13/03/2015.

[47] LaTeX Project. Latex. http://www.latex-project.org, 2014, retrieved on 02/03/2015.

[48] Microsoft. Windows 8.1 price in spain. http://www.microsoftstore.com/store/mseea/es_ES/cat/Windows-8/categoryID.66227200, 2015, retrieved on 02/03/2015.

[49] PayScale Inc. Payscale - salary comparison, salary survey, search wages. http://www.payscale.com, 2015, retrieved on 13/03/2015.

[50] Hamza Aziz (Destructoid). Rogue legacy became profitable within its first hour on sale. http://www.destructoid.com/rogue-legacy-became-profitable-within-its-first-hour-on-sale-272074.phtml, 2014, retrieved on 27/02/2015.

[51] Vine. Cavern kings. http://www.cavernkings.com/, 2015, retrieved on 10/06/2015.

[52] Retronic Games. Pixel dungeon on steam. http://store.steampowered.com/app/365900/, 2015, retrieved on 1/06/2015.

[53] Creative Commons Corporation. Creative commons by 4.0 license. https://creativecommons.org/licenses/by/4.0, 2015, retrieved on 10/06/2015.

[54] Per Nyblom. Abundant-music. http://www.abundant-music.com, 2015, retrieved on 10/06/2015.

[55] Increpare. Bfxr. http://www.bfxr.net, 2015, retrieved on 10/06/2015.