Procedural Game in Unity

Student Name: Sorin Iliuta

Reg. No: 1401971

Supervisor: Vishuu Mohan

Second Assessor: Dongbing Gu

Degree: Computer Games

# Acknowledgements

Special thanks to Prof. Richard Bartle, who helped me with a few game design aspects of the game. He is the one that suggested allowing people to track where they’ve been. This led to the implementation of the spray can system.

I also used Kenny’s models in this project, which I purchased for myself a long time ago and his work proved useful in allowing me to concentrate on the important part of the game. All of his work can be found at <https://kenney.nl/>

My friends also deserve an honorable mention since, while most work was done by myself, occasionally they gave me advice or offered me solutions to certain problems that I had. While not implementing his idea, Perry’s advice to how to detect if something collides without using physics is what ended up leading to an entire rework in the generation process. This resulted in a much cleaner, faster and precise system.

# Summary

The project is about creating a whole game in Unity, concentrating on the procedural part of it. The game does not have a proper name, in the file being simply called Dissertation, but in the Design Document I named this “Trapped in Darkness”.

I worked mostly on the procedural aspect of the game and it successfully generates a mansion-like structure with an exit and an enemy inside. Lockers can be found in the hallways as a way of hiding from the monster.

Every time a new game is started, the layout is completely different, but the player can still go from point A to point B.

Contents

[Acknowledgements 2](#_Toc512588311)

[Summary](#_Toc512588312)

[List of Symbols 5](#_Toc512588313)

[Similar Examples 6](#_Toc512588314)

[What does this accomplish and is it useful? 6](#_Toc512588315)

[Game Design 7](#_Toc512588316)

[Why Procedural Generation and What is it? 7](#_Toc512588317)

[What is Procedurally Generated in the Project and How Does it Work? 8](#_Toc512588318)

[Project Planning 11](#_Toc512588319)

[Conclusions 13](#_Toc512588320)

[References 14](#_Toc512588321)

# List of Symbols

RPG – Rople-playing Games

PG – Procedural Generation

# Similar Examples

While I may not be the first one to possible attempt this, I am the first one to do things the way they are, as far as I could tell. Searching through the web for any form of procedural house generation yields almost no results. The only one that looks decent comes from a stackexchange question [1], but the problem with that is the result is too square and doesn’t offer much of a solution.

The 2nd best example I could find, that I took inspiration from, is a Gamasutra article by A Adonaac about Procedural Dungeon Generation [2]. The article is interesting and I drew inspiration from it. You start with a bunch of rooms stacked on top of each other. You then spread them apart by simply activating the physics and let them separate themselves until they all settle down. Select some “main rooms”, meaning any room above a certain threshold, but not remove the rest as they will be useful later. You take the middle of each room and feed them in a Delaunay triangulation, which is just a method of triangulating the position so no point is inside the circumcircle of any of the formed triangles.

The author does not specify why he uses that method, maybe it’s because it is a commonly used triangulation algorithm, maybe there is more to it than I understood. I did not research more into it, because I never used this method in my project.

After the triangulation we create a minimum spanning tree, this way all rooms are reachable. We then connect the rooms with some lines. The way this is done is up to you, the author does offer a solution to this. Once we have our lines we see which of the rooms we had collide with these lines. If a room doesn’t do that, we remove it. Once this is done we are one step away from a dungeon. The last part is filling any missing gaps and that’s it.

This whole thing works nicely, but again, it’s for a dungeon, I wanted a mansion. The first article does offer a nice solution for a house, but it’s too basic, mansions have weird shapes and are pretty large, so I had to take a different approach.

# What does this accomplish and is it useful?

This project is trying to create a mansion-like structure in a video game. Dungeons have been created numerous times, because they are easy to procedurally generate: spawn rooms, spread them apart and then link the rooms via hallways. However, I have yet to see somebody create a whole mansion in this way. As such, I set myself the task to do it.

Is it important to anybody? I wouldn’t say it is really important, unless you create video games. This would allow developers to create nicer games with a lot of replayability value, despite having maps that are man-made structures. You could use this to generate houses and combine it with a town generator to procedurally generate a whole city.

Society already looks down to games, a lot less than before, but there is still a bit of stigma with playing video games. It doesn’t even benefit from my project in a direct way. That being said, gamification is a lot more popular nowadays, so this could be used in architecture. You could imagine architects using my game to load in different types of rooms and hallways and running a few instances to look at some possible arrangements and maybe settle with a design. This would help them a lot, since they can visualize different houses a lot more quickly than they could build replicas of, either in real life or on a computer.

Could this lead to, eventually, replacing architects entirely? No. We are still better at creativity and even if normal people would use this to design their houses, the validity of the structure would need to be approved by an architect who would still need to supervise constructors and more. So, in the end, it would be just a tool that can expedite a long and laborious project.

# Game Design

The project, while concentrating on the procedural aspect of it, it does have a game design document attached to it. Originally planned to be even more random and it might still be for the future, switched directions as I got more into it. Originally planned to be an RPG, it quickly turned into a Survival Horror.

To this day, I still believe that players enjoy uniqueness, discovering new things, be the hero and sometimes get a good scare. Trapped in Darkness was originally designed to trigger a fight or flight response from the player when faced with danger. Each game would have a set amount of monsters, each unique, selected from a list of possible ones and each with its own random set of statistic such as: peaceful/neutral/aggressive, vulnerable/invincible, support/tank. Each monster had a few skills they could use, once more, the game pre-selecting a few of them at random at the start of the game.

I wanted to give the player the sense that they are always discovering something new, which is why the map is procedurally generated. I also originally intended for them to approach monsters to discover what they can do, but looking back on the design that could turn bad really fast, so I eventually scrapped it.

# Why Procedural Generation and What is it?

So what exactly is PG and why do I care so much about it? Procedural Generation is a method where content for a game is produced by the computer, following a set of rules defined by the creator. For example, if you would want the game to have different types of swords, you could split the sword in multiple parts and then mix and match those at random to get different types. You could even try and create different 3D models based on roughly what a sword is. The way it is implemented does not matter. It can be as complex as you need it to be.

But why PG? Well, to put it simply, replayability. You might have never heard of this word before and if you would look in a dictionary, you will quickly realize it does not exist. That is because this is a more recent term used by game developers to describe a game in terms of replaying a game without getting bored. A good example would be any RPG, such as Skyrim. It has a huge replayability value, because each time you start a new game you could act completely different, do missions in a different order or even do missions that you’ve never even had in previous plays of the game.

The lack of repetitiveness of the game keeps players engaged for many more hours than normal. Most games offer about 30 hours of content, but with PG this can be increased in the hundreds or even thousands. It adds value to the game. This is all because players can still experience new things, despite knowing so much about the game, yet they will never know everything. They can know how the story will progress, but not how a fight will go, what enemies they will encounter, what the map looks like, etc.

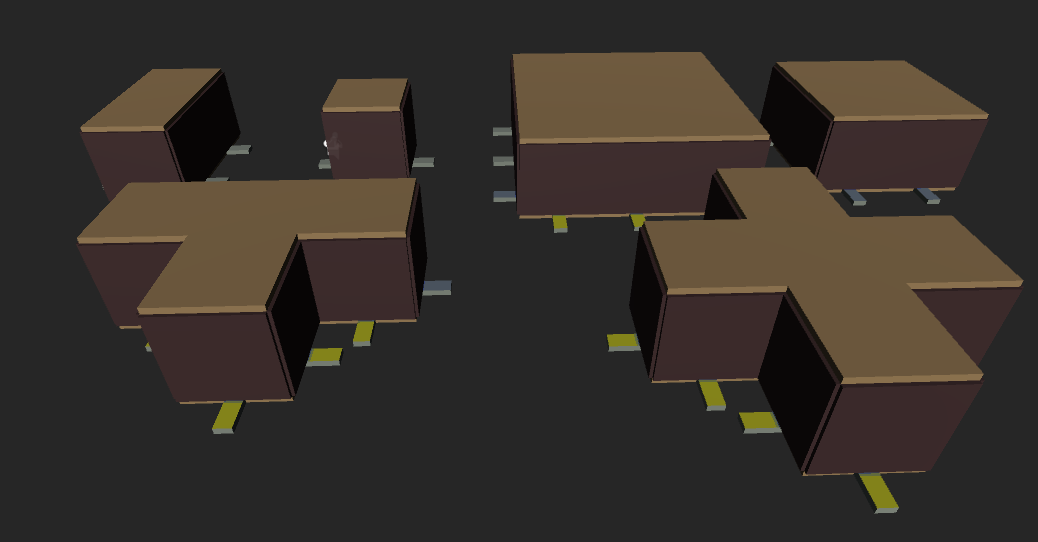
This is such an important aspect of a game, because it can keep a project alive for much longer, still generating revenue. Minecraft would be the best example, still selling copies to this day and it’s the reason a lot of people started making sandbox games after its boom in sales.

# What is Procedurally Generated in the Project and How Does it Work?

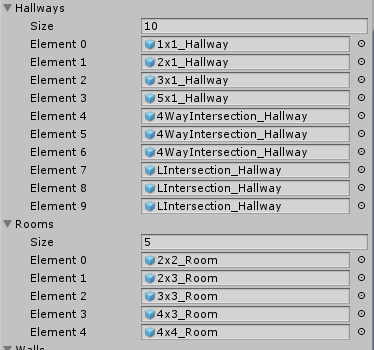
There are about 3 things procedurally generate in this game: the map, the enemy spawn location and the exit spawn location. The way the last two are generated is simple, so I will cover them first.

Once the map is generated, enemies are spawn in some of the last generated structures, which are the rooms, so they are usually spread apart. The exit is even easier, because it replaces one of the walls that is furthest away from the player. These are not perfect, but they do the trick until they get improved later.

Now for the actual interesting part: how does the world generate. The map is comprised of multiple structures attached to each-other. You have hallways and rooms, multiple types of different shapes and sizes. They follow the standard rule of having a floor, a roof and walls, each wall being a separate piece, covering every side.



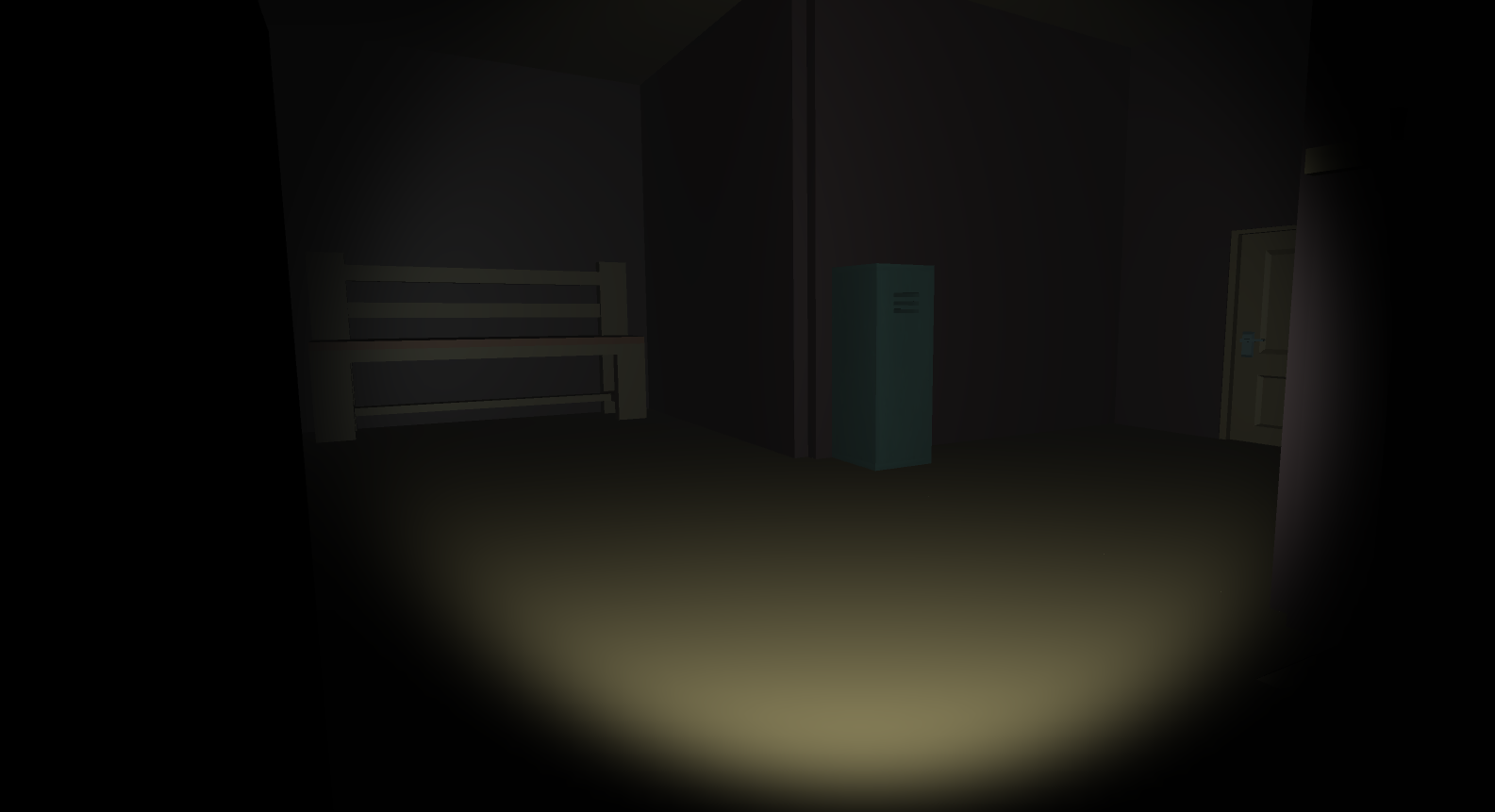
At the level of the floor, in the middle of each wall, lies what I named “Link”. A small yellow rectangle that faces the same direction as the wall and holds a variable containing the wall it is linked to. These are what makes the world generation function. Think of them like the little nubs on a Lego piece. Once a few prefabs of some hallways and rooms have been made and they all have links attached, it is time to move to the world generation.

Now, a lot of tutorials online on how to procedurally generate anything would lead you to an article on Gamasutra about a Procedural Dungeon Generation Algorithm. At first glance it looks nice and works really well, it is highly efficient. However, looking at it closely you can realize a problem. While it works well for things like caves and dungeons, it is horrible at generating houses. No house has rooms separated from each other with hallways connecting them. This meant I had to find another way of creating a house-like structure and luckily I didn’t have to look far. If you were to create the hallways first, then add rooms, it would resemble more of a human-made structure.

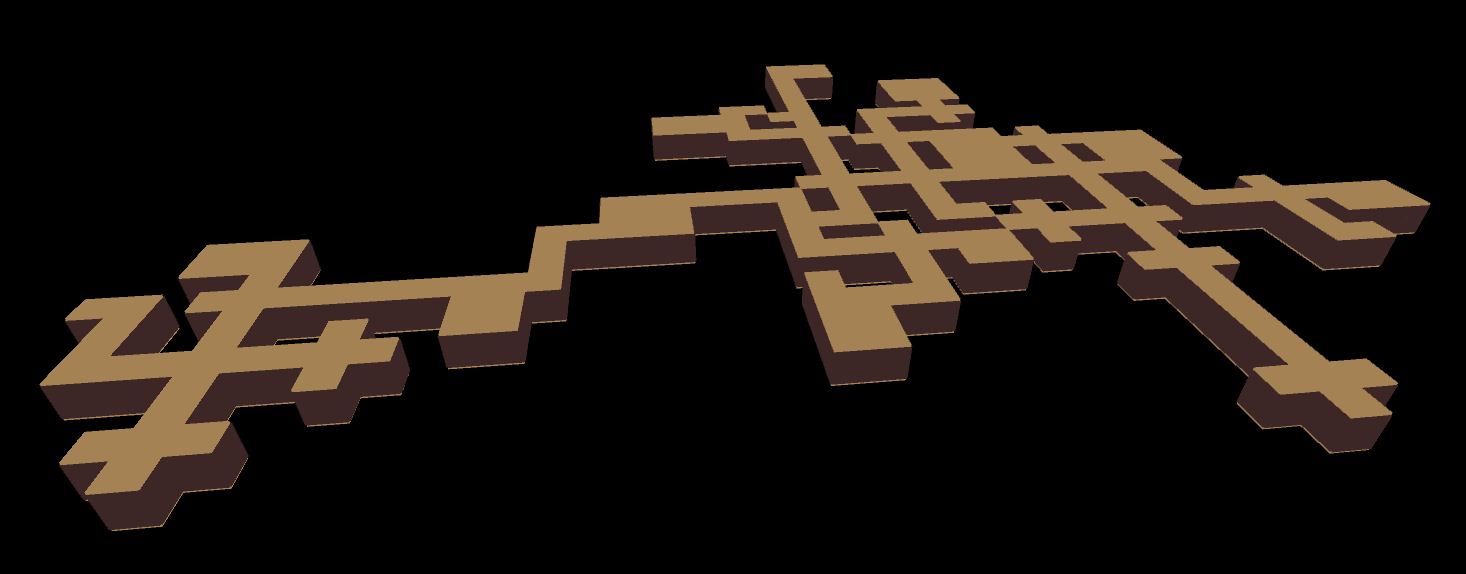
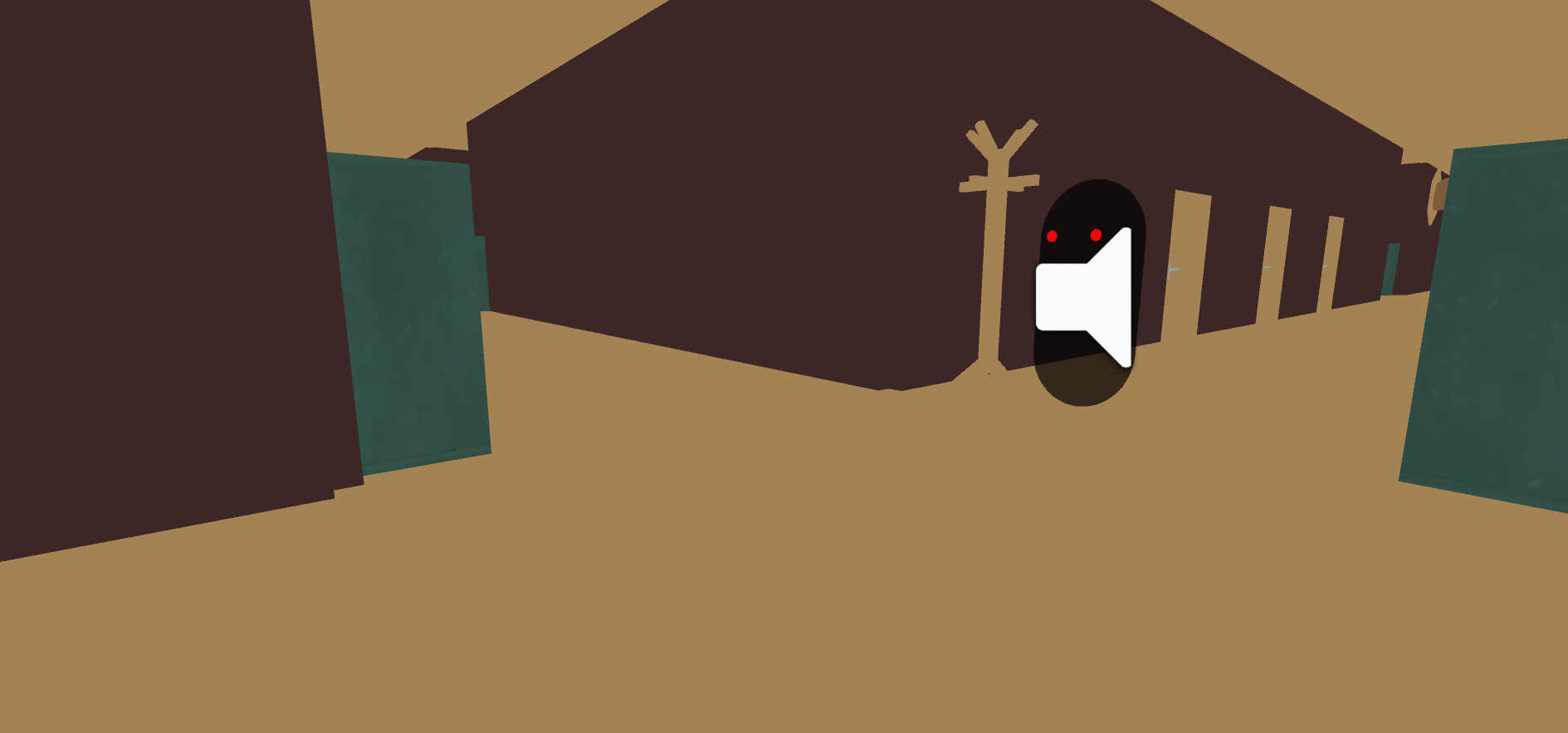
So I start with spawning the starting room, which is a room that will always exist on the map and serves as a starting point for every other room to attach itself to. After that I generate a number of randomly selected hallways from a list. How many? Well in my game I do 50 hallways, more hallway leads to bigger and much more interesting structures, but it also slows down the generation and makes the game a bit too boring, so 50 is a nice number.

I then start placing the structures I spawned, one by one. I go in the order they were generated, take a random link from them and a random link from any of the settled structures. A settled structure is simply a room that has been generated and successfully attached itself to another part. That structure is added to a list named arrangedRooms. Once the links are selected, move the room in that spot and rotate the selected room as such the two link have opposite rotations and move the room so that it connects perfectly with the already arranged one. Then I check if that room intersects any other structure, to avoid overlapping rooms. If it does overlap with anything else, I select another random link and remove the last one from the available ones and continue with this process until the room is properly placed. Continue doing this until all hallways have been placed or removed (in case there were no available places for it to stay without overlapping).

Now that the hallways are spawned, it is time to move onto the rooms, which is easy, because you just have to do the same thing you did with the hallways, but now with actual rooms. This is what allows for the formation of a more human-like structure.

Now, this is nice, but when I originally did this, obviously I forgot about the walls. I had a nice structure, but I couldn’t go anywhere. So what I did is, after I settled a structure I removed the links that I used to connect the two and destroy the walls they are linked to. So now, we have a passage to every part of the map. In case of rooms I replace one of the walls with a doorway. 

Now it is better, however, you can have loops, but walls block them. So I go through all the links that are left on the map and see if they collide with anything. If they are part of a room and are colliding with another structure (be it a hallway or another room), I destroy that wall and replace it with a doorway. If it does collide with something, I destroy it and the wall. So now we are left with a nice, open structure that resembles a mansion, structure-wise.



# Project Planning

The project started strong, as I had an idea in mind and knew the direction I was going in. I chose to go with Agile method, because it suited more my style than the Waterfall method and, looking back at the project, I think I chose properly.

Ideas kept flowing, things changing, nothing ever was definitive, besides on what I wanted this project to be, or so I thought. I originally wanted this to be an RPG, but it turned into a survival horror game, but I am not disappointed. By doing things the way I did, I managed to discover some flaws with my design, mainly how too many monsters is just not that fun when you don’t have a way to defend yourself, so changing it to be only one might have been a better idea. Having more monsters, however, is not something that I completely said no to. I originally thought I could just split the mansion in sections and have each section be populated by a monster, which might be an addition for the future, but right now, one monster is just enough.

While I think I did pretty well, looking back at it, I feel like I could have worked even more and harder. There were times that all I did was take a step back and think, when what I should have probably done is just implemented stuff and see how they worked. That being said, I am really proud of the world generation.

The first generation of the function was long, bulky and confusing, so being able to shorten it and make it faster really makes me proud. As such, I learned how to properly structure my code, critically think of better solutions and take a step back and come up with logical solutions to problems. All it took for me to improve my generation code was to take it to a drawing board. Just lay everything down and visualize how the generation would make sense in a logical manner.

# Conclusions

I started this project thinking I would make a procedurally generated horror RPG in Unity, with plans for smart companions and different types of enemies. Instead I ended up with a procedurally generated Survival Horror game, which might have been a better thing.

For the future, it might be in my interest to actually think harder on project goals, before starting to implement. A lot of the design problems could have been avoided and things could have gone faster and smoother, had I thought of the original choices a bit more.

# References

[1] generator, P. (2013). *Procedural... house with rooms generator*. [online] Gamedev.stackexchange.com. Available at: https://gamedev.stackexchange.com/questions/47917/procedural-house-with-rooms-generator [Accessed 27 Apr. 2018].

[2] Gamasutra.com. (2015). *Procedural Dungeon Generation Algorithm*. [online] Available at: http://www.gamasutra.com/blogs/AAdonaac/20150903/252889/Procedural\_Dungeon\_Generation\_Algorithm.php [Accessed 27 Apr. 2018].