Project Title: Utilizing procedural generation to create a dungeon crawler game

Name: Jack Travis

Supervisors Name: ---

Final Year Project Dissertation

ABSTRACT

This project attempts to expand upon the current ongoing research of procedural generation in games, to create a greater variety of content that provides a constant experience. I aim to utilize procedural generation to create a 3D map that could be used in a dungeon crawler game. This was achieved by using random number generators to randomise various choices, such as the size and position of a room and the pathing of a corridor, which was represented in an array. This array is then later interpreted to create all the required game objects to create the map. Testing and analysis suggested that this solution provides a good starting point for its premise, but has many areas in which it can improve, such as a more consistent 'smart' corridor generator, and the addition of a greater variation of assets and materials. It was concluded that this solution may be suitable for run time map generation, but is not ready for deployment in a live game, due to the suggested changes that should first be made, and due to its high complexity and maintainability.

CONTENTS

1 – INTRODUCTION

Procedural generation is a powerful tool when implemented correctly, allowing for an element of randomness for chosen game aspects, such as the map and lootable objects. This randomness can make every playthrough feel unique and interesting, increasing the likelihood the user will want to replay the game and invest more hours into it.

This can be seen with games such as Minecraft, Terraria and No Mans Sky, which all procedurally generate their maps and remain popular to this day, with Minecraft selling more than 238 million copies to-date [14], making it the current bestselling game. An example of a different successful application of procedural generation is the Borderlands series, whereby all weapons are generated procedurally. [15]

Procedural generation makes this variation possible, which otherwise would not be achievable, nor cost effective to do manually. Despite this, there are limited examples of using such a method in published games.

The aim of this project is to review the current state of procedural generation, specifically for the generation of maps for rouge-like games and to create my own 3D map generator for a dungeon crawler game.

1.1 – PROJECT MOTIVATION

One reason as to why I chose to undertake this project is due to my interest in the games industry. I would like to improve my ability in this field, learning skills that would be transferable to a career. Successfully implementing procedural generation to a game can vastly increase its replay ability, as your experience will differ each time. Therefore, I would like to learn and understand how it is achieved.

Another reason is to investigate and document the current methods of procedural generation and to learn of the difficulties of using said methods in terms of game balancing and implementation. This, along with my own attempt, will help to gain a greater understanding as to why this method is not commonly used in the industry.

The completion of this project should also provide a contribution to the current research into this field, with the intention to suggest further areas of research and to inspire others to attempt their own implementation of procedural generation.

1.2 – OBJECTIVES

After a review of the original objectives, I realized that they were focused on creating a complete game, rather than the actual point of this project: To create a map generator. As such, the objectives of this project have been revised and are listed below:

1.2.1 - CORE OBJECTIVES

1. To research the current state of procedural generation to provide a comparison between suggested solutions.
2. To pick and plan a suitable approach for my project.

3. To generate rooms, of customizable sizes, in valid positions, such as not overlapping with other rooms, with doorway(s) in suitable locations.
4. To generate a network of interconnecting corridors, connecting all generated doorways, to create a traversable map.
5. To distribute objects, such as light sources and chests, throughout the map to provide an example of how decorations/interactable objects could be procedurally placed.
6. To create and place AIs throughout the generated map, to show AI navigation is possible, for applications such as enemies.
7. To create a 'How it Works' option, whereby the user is visually shown the generation steps, to help aid with understanding how procedural generation works.
8. To create a 'Custom Map' option, whereby the user can alter the changeable parameters that affect map generation, creating their own custom map.
9. To create a player controller, allowing for easy traversal and investigation of a generated map.
10. To evaluate and review how successful my project was in comparison to other proposed solutions, providing information such as strengths and challenges, what the next steps would be and what questions arise for future research.

1.2.2 - OPTIONAL OBJECTIVES

1. To introduce multiple interconnected floors to the map.
2. To introduce local lighting, instead of global lighting to illuminate the surroundings, as well as creating darker areas.

1.3 – ADOPTED APPROACH

An experimental approach was taken to accomplish the aforementioned goals. Since this project is focused on adopting and/or creating a map generator, which will then be compared to other documented solutions, this project will mainly be focused on development. An Agile approach was deemed to be the most suitable, more specifically in line with Kanban. This allows for easier task prioritisation; Whereby more important features/fixes are prioritised over less important ones.

1.4 – REVIEW STRUCTURE

This report begins with a review of the current literature on this topic, before deciding on the plan of action that was followed for the development of the software. Details regarding the requirement specification and the design process can be found next, in Chapter 4, followed by the development and implementation process.

Chapter 6 details the performed testing and analysis, followed by a critical evaluation and conclusion of this project.

2 -LITERATURE REVIEW

2.1 - INTRODUCTION

The purpose of this document is to review the current findings and methodologies of procedural generation regarding map generation for video games, more specifically the generation of dungeons for Roguelike games. There are many algorithms that have been created or adapted for this purpose, each with their advantages and drawbacks.

This document will begin by reviewing previous reports on this topic before exploring what research has been conducted since. I will then conclude the state of Procedural Dungeon Generation.

2.2 - OVERVIEW

Procedural generation is a method of creating content, either partially or fully, by utilizing a computer to perform this task, based off a given algorithm. Successfully implementing this can allow for randomised features in a game, such as lootable objects or the map itself, potentially increasing the games replay ability without the cost and time it would take a designer to handcraft this content. However, this has a trade-off of greater development time that ultimately leads to reduced control, making future changes significantly hard or impossible. Balancing the game to create a consistent experience can prove challenging, especially for more complexed applications, requiring a longer development period. As such, most of the current research into procedural generation focuses on providing a solution for a very specific criteria, rather than a more generalised solution.

Amongst the reviewed papers were two previous surveys on Procedural Dungeon Generation; one of which was released in 2014 [1], the other was released in 2019 [2]. Comparing these papers shows the progress made during the 5-year span, the most noticeable being the number of different approach types. The 2014 paper states that the current documented approaches to procedural dungeon generation are: Cellular Automata, Generative Grammars, Genetic Algorithms, and constraint-based methods. The 2019 paper also states these methods, as well as Genetic Programming, Answer Set Programming and Constructive Approach, all of which referencing papers released after 2014. They also classify all algorithms as either ‘Constructive algorithms,’ or as ‘Search-based Algorithms.’ However, despite further research into this field, both papers concluded that more research into 3D map generation is required, as ‘3-D generated dungeons are still far from designer made ones’ [1] and that ‘few works presented solutions for PDG of 3D levels.’ [2].

2.3 - CELLULAR AUTOMATA

Cellular Automata is a method invented by John von Neumann in 1947 and is a semi-autonomous way of generating content. This is done by creating a grid of cells of a finite number of dimensions, with each cell in one of a finite number of states, decided by implemented rules. [3] Below are some examples of recent works that utilised Cellular Automata:

Y. Macedo [4] created a 2D map containing impassable sections.

Figure 1: Y. Macedo's [4] generated map.

H. Fabroyir [5] used Cellular Automata alongside Poisson Disk Sampling to create a map of interconnected rooms.

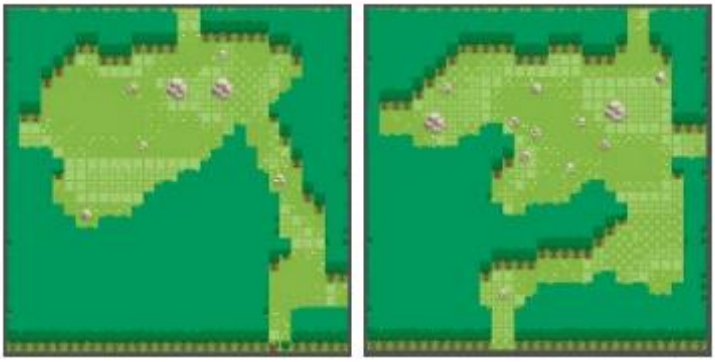


Figure 2: Rooms generated using H. Fabroyir's [5] algorithm.

C. Adams [6] created a maze using this method.

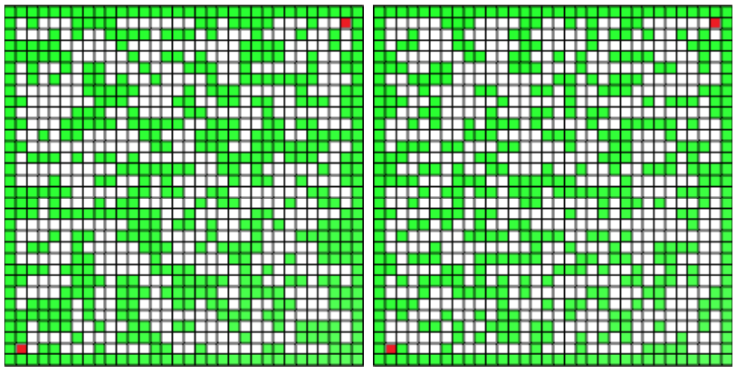


Figure 3: A Maze generated using C. Adams [6] algorithm.

Although the final product for the aforementioned research papers differs, they all attempt to use Cellular Automata to create a world that could be explored that contains impassable terrain.

Y. Macedo [4] states that their paper provides further evidence that Cellular Automata has been 'proven' to be 'extremely flexible,' but states that it 'still has a path to travel before it can safely become an industry standard.' H. Fabroyir [5] also agrees that Cellular Automata shows promise, but still requires further research, concluding their research requires 'a game mechanic that adjusts the shape of the room generated,' to allow the sections to be joined together. C. Adams [6] also states that the 'superior

performance' of this method shows this to be a promising 'avenue for further research.' However, the claim of 'superior performance' was regarding a different approach at generating a maze that met the same criteria, with no comment on whether a different algorithm entirely could yield better results.

Whilst all three papers state that this field is promising for further research into 2D procedural generation, two of said papers [5, 6] also commented that research into using Cellular Automata for 3D map generation shows promise and should be conducted, referencing the same paper from 2005 [7]. Despite this, I was unable to find any publications that attempted this for the generation of dungeons, although there was a paper [8] that used Cellular Automata to create a data visualising tool in a 3D environment. Therefore, it is yet to be seen if this method can successfully generate 3D dungeon environments.

2.4 - GENETIC ALGORITHMS

Genetic algorithms are a method of generating content using a natural selection process that mimics biological evolution, whereby rules from a ruleset are randomly selected and used to create the 'parents,' that in turn produce the 'children' of the next generation. [9] After multiple generations, the created children are used to dictate how the content is generated.

A 2018 paper by A. Kholimi [10] used this method to create a platformer game, whereas a 2020 paper by E. Susanto [11] generated a maze via genetic algorithms, showing that this algorithm can be used for different applications.

A. Kholimi [10] noted that their attempts to create easier levels for the user to complete were successful, but when they attempted to generated harder levels, 'some generated game worlds are not playable.' They decided that more difficult levels would be favourable if this method was to be used in the industry, therefore stating that their algorithms needs 'to be improved.' The author didn't suggest further research on other applications where Genetic Algorithms could be suitable, nor did they suggest any further improvements that could be made to the model.

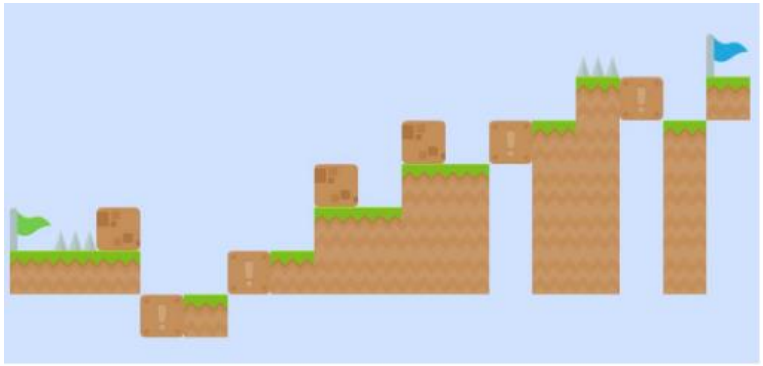


Figure 4: An example of a successfully generated map using A. Kholimi's [11] algorithm.

E. Susanto [11] demonstrated the versatility of their approach by creating two versions, 'one with laser doors and one without.' The author states that their implementation can successfully generated levels that are 'three times more' complexed than their training data. However, they stated that further research should attempt to achieve this with 'fewer or no training data.'

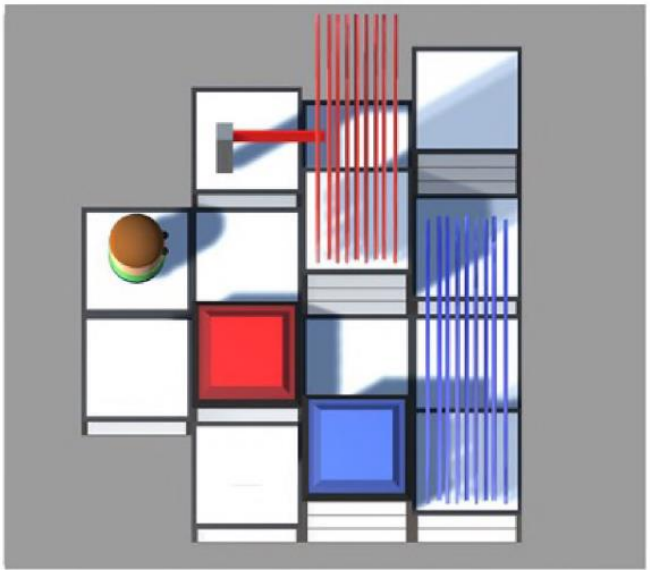


Figure 5: A map created by E. Susanto's [11] algorithm, including a player, goal, laser doors and key panels.

Both papers show a successful implementation of a Genetic Algorithm, albeit both with drawbacks. The platformer game [10] was only able to generate levels up to a certain difficulty before their fitness function began generating impossible levels. Whereas, the maze generator successfully generated levels of a high complexity, with the caveat of first requiring a training set, which may not be possible for all applications. Thus, both require improvements before being options for the industry. R. Linden [1] summarised Genetic Algorithms as 'greatly' dependent on the fitness function, requiring 'considerable knowledge about the entire generation process' to implement. This could see this method as unfavourable in the industry, due to its complexity.

Neither paper commented on generation times, therefore no comment can be made on potential use in a real time application.

The paper regrading maze generation [10] is one of the only papers found that also attempted to procedurally generate other features, such as barriers ('laser doors' in this case) and the addition of a player character. The algorithms compatibility with such features is promising, whereby further research should aim to test the potential of this. The effectiveness of this additional feature generation should also be compared to the potential of other algorithms when such research is done.

2.5 - GENERATIVE GRAMMARS

Generative grammars use linguistic phrases, created from a finite selection of recursive transformational rules, which the computer interprets and generates a level based on said phrase. This process has a prerequisite of requiring training data, to allow the computer to learn what is required to understand the phase/rules.

R. Linden's [1] report reviewed the status of procedural generation using Generative Grammars in 2014, summarising that the research showed that this method allowed for 'versatile results,' but had a 'high complexity in setting up graph and shape grammars,' which could see the industry favouring different

solutions. However, although none of the papers discussed 3-D dungeon generation, the author states that Generative Grammars shows promise for this and 'encourages future work in this area.'

Since the release of this report, I could only find one publication that used Generative Grammars for map generation, [12] which attempted to generate levels similar to that of The Legend of Zelda. They stated that they were able to generate dungeons of similar 'enjoyment, challenge level, and complexity,' successfully creating 'effectively infinite multitude of such solutions.' Although this appeared successful, they also concluded that further research should attempt to replace the Generative Grammar with a 'data-driven' solution, as to no longer require a training set.

The reviewed papers prove that this method can create procedurally generated environments, with a trade-off of complexity and the requirement of a training set. However, this is one of the only methods that has currently been proven to allow for the creation of barriers during map generation, the other being Genetic Algorithms. Therefore, further research into Generative Grammars could yield positive results.

2.6 - GRAPH ALGORITHM

A publication by B. T. Lipinski [13] suggests building an algorithm on 'an extended minimal spanning tree, which can be controlled by a set of intuitive vertex and edge parameters.' This was the only paper that I found that attempted to generate a truly 3D dungeon (the dungeon has multiple floors that can be accessed without a loading screen or transition), and the only one that opted for this solution.

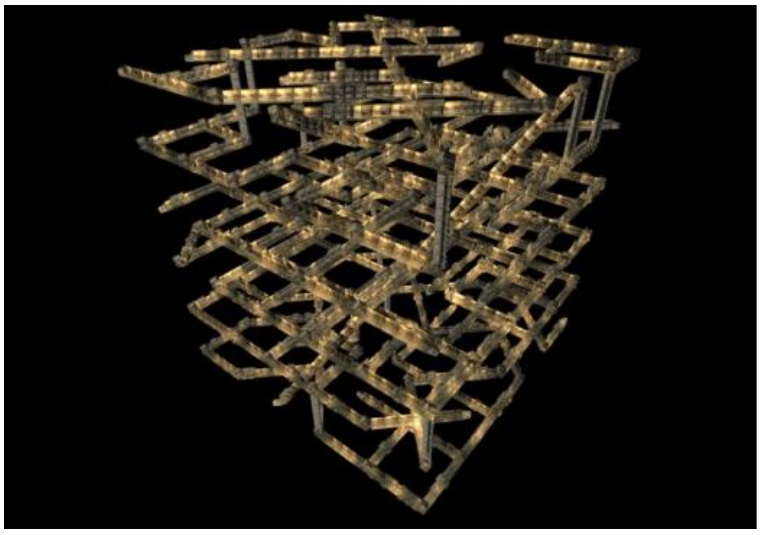


Figure 6: A typical sized 3D dungeon generated using B. T. Lipinski's [13] graph algorithm.

This method was able to create standard sized dungeons in approximately 200-600ms, with a much larger example taking '43.75s' to create. The writer described the performance of this as 'adequate for practical use.' However, the current implementation of this method has a bottleneck, due to the 'current (non-optimized) implementation of the geometric conversion,' which took 'from about 3.5 minutes for the dungeon level to ca. 5 hours for the space station,' to turn the minimal spanning tree into a 3D environment. The write states that this is 'not yet satisfactory and requires optimization.'

The creation of 3D dungeons is a field that hasn't currently had much documented research, therefore not allowing for a direct comparison of this algorithm to another solution. However, this paper provides a great starting point for further research. The time taken to create the minimum spanning tree is acceptable for real time generation, potentially allowing for further applications for this algorithm.

There is no comment in the publication regarding how the dungeon generated by the algorithm compares to that of a handcrafted dungeon. A study regarding this would be beneficial to identify further areas of improvement.

The biggest problem this solution has is the time it takes to create the 3D environment. The use of an optimised geometric converter is required to see how long it would take to create the map, allowing for one to properly comment on the applications this algorithm may have.

2.7 - CONCLUSION

There are currently many ways in which 2D Procedural Dungeon Generation has been approached, each offering various advantages and disadvantages, the main two factors being generation time and control. The most suitable for a given scenario is dependant on what the goals of the application are, and whether they have training data available.

In contrast, there is very little research into 3D Procedural Dungeon Generation, with current solutions being suboptimal. Further research is required in the field before it is at a state whereby it can become an industry standard.

Most publications also didn't consider game difficulty options, adding lootable objects to the map or the addition of NPCs. Research into the compatibility of such features to a procedurally generated map is required.

3 - METHODOLOGY

3.1 - DEVELOPMENT METHODOLOGY

For this project, I intend to use the Agile methodology. Agile allows for more flexibility when developing a system, allowing you to focus on each feature individually, rather than planning everything from the beginning. This allows for alterations to be made if required with minimal impact on the rest of the project.

Since I am creating something that is unlike anything I have done before, it isn't feasible to suggest I can create a concrete plan for the entire project beforehand. It is also not possible to accurately predict how long certain features will take me to develop, therefore dedicating a lot of the finite time I have towards planning is unfavourable. It is possible that I will not have enough time to add all the features I intend to add. Likewise, the project make take less time than expected, allowing for additional features to be added. With this in mind, methodologies such as Waterfall are not compatible, since I would have to start back at the first development stage if any alterations were to occur. Whereas, Agile allows me to respond to such changes with limited impact, providing more versatility.

Agile does come with its downsides, such as limited documentation. However, I do not believe these issues will be that significant, since most of the downsides are only noticeable as apart of the team, or for continued development of the system. Since this is a project with an end goal, and there only being myself working on it, I do not believe such downsides will affect me to a degree that a different methodology would be more favourable.

Originally, I intended to use the Agile method, Scrum, to structure my progress, splitting my tasks into sprints and reviewing my success after each sprint. However, since I cannot accurately calculate how long each task will take me, or detail specifically how I intend to achieve said goal, along with other unknowns, such as how long other assignments will take to complete, I will instead opt for a Kanban approach. This will allow me to visualise my current workload and allow me respond to my availability to work with greater ease.

3.2 - ALGORITHMS AND TECHNIQUES

Although the Literature Review I conducted detailed many methods which are being researched to create a procedurally generated dungeon, each had drawbacks or were only built for their specific requirements. Some of the solutions also required a training set to learn from beforehand. Such drawbacks would hinder what my design would be capable of, causing a conflict with what I originally set out to accomplish.

Only one solution provided could create a truly 3D dungeon, but did not populate the dungeon with lootable objects and hostile NPCs, therefore not creating a functional game. Their method also took a significant time to create each map, which wouldn't allow me to generate the map in real time (when the player presses 'play'). Therefore, this solution isn't viable for my intentions either.

Instead, I will have to attempt to create my own algorithm for this project. I intend to create my own constraint-based method, the exact nature of which is unknown currently. I will require experimenting with different approaches until I find one that appears to meet my requirements, before committing to that approach.

3.3 - SOFTWARE REQUIREMENTS AND LANGUAGES

For my project, I will use Unity version 2021.3.5f1. I believe this will offer all the functionality I require, whilst remaining compatible with the software on the University machines. Since I am using Unity, I will be using C# for coding. The IDE I will be using is JetBrains Rider, due to my preference regrading layout and autocomplete suggestions and its integration with Unity itself.

This project will also require the acquisition of animations and assets, for which I plan on utilizing Mixamo and the Unity Asset Store respectively.

3.4 - HARDWARE REQUIREMENTS

The finished product for this project will be a 3D dungeon, in which the player can navigate around and interact with, including combat with hostile NPCs. As such, a dedicated GPU will most likely be required to run this game.

The generation of the dungeon may be computationally heavy, both on the GPU and CPU, and as such would require a relatively modern PC to ensure compatibility. Below is the current specification of the device I will be working on. I do not expect this to change during the project, and believe it has sufficient processing power:

- CPU: AMD Ryzen 7 3700X: 8 core, 16 threads, 3.6 GHz base clock speed
- Memory: 24 GB DDR4 RAM
- GPU: NVIDIA GeForce RTX 3080 Ti 24 GB

3.5 - INVESTIGATIVE METHODS AND EVALUATION

I aim to create an application whereby a truly 3D dungeon is procedurally generated, whereby there are multiple floors that can be explored. As such, I only found one paper [13] that had a similar goal. As such, the success of my project will partially be dependent on how it compares to the aforementioned publication.

Unfortunately, I was unable to obtain any publications regarding 3D dungeon generation that also included lootable objects and NPCs, therefore not allowing a direct comparison to other research. However, I can investigate the time complexity that adding such features has on the map generation time.

I also intend to collect information from participants to see how the game is received, compared to what they expect a typical Roguelike game. I will also be able to collect information regarding lootable objects and the NPCs added. This information will allow for the evaluation of how successful I was in creating a Rouge-like game, as well as highlighting areas for improvement for further research. Feedback will be collected in the form of a questionnaire.

4 – REQUIREMENT SPECIFICATION AND DESIGN

This section will outline and justify the finalised decisions that created the basis for the development of this project, in terms of requirement specification and the overall design.

4.1 – REQUIREMENT SPECIFICATION

Since this project aims to create maps procedurally, there will not be many user interactions, only requiring the user to navigate the scene and to use an interface to allow them to generate maps and/or customise the map generator settings.

Due to the nature of this project, there is a more pressing question than how will the user navigate this application: What does the map generator need to do to successfully generate a dungeon map? Therefore, a crucial first step was to identify these requirements. This was done, based off my knowledge of this genre of games, as well as the mentioned strengths and drawbacks of other proposed solutions.

4.1.1 – PROCEDURAL GENERATOR REQUIREMENTS

4.1.1.1 – REQUIREMENT 1 – ROOM GENERATION.

A room, in this case, is an enclosed area of space, accessible via doors/doorways. These will be points of interest for the user, since this is generally the source of lootable objects, and therefore need to be identifiable as such. They are generally larger than other map sections, such as corridors, and should be created in a range of sizes, to allow for a greater variety.

Other differences might be a variety of decorative objects, such as different floor/ceiling/wall textures and a change in the local lighting. Rooms would serve as a good place for AIs to guard, or to walk between. There should be a variety of room types, each with their own set of decorations and lootable object types. Rooms may also serve as a spawn or exit point for the player.

As such, it is necessary for the generator to not only generate areas that can function as a room, but to also remember where the rooms are, so they can be treated differently.

4.1.1.2 – REQUIREMENT 2 – CORRIDOR GENERATION

Corridors are pathways that connect areas of importance together, such as rooms. Each generated corridor should ideally connect 2 or more rooms together or connect to another existing corridor. Not all corridor sections are required to be connected directly together; A room with multiple doors is also valid.

They should be relatively easy to follow, not maze like. This may be accomplished by reducing the number of times a corridor splits into multiple directions, therefore reducing the likelihood of becoming lost. Instead, a more direct path towards the desired destination, should be favoured such as a door/doorway to a room.

Corridors do not generally have many objects throughout them, with just lighting sources being the most common, such as wall mounted torches. They should be traversable by both the player and AI's.

Based off the above requirements, the generator must be able to generate corridors, choosing what to add next (continue straight, change direction, or split into multiple directions) based off various available information, as well as retaining some way to remember the location of all previously connected corridors. The generator must also be aware of doors/doorways present, to connect to them.

4.1.1.3 – REQUIREMENT 3 – NO OVERLAPPING

The generator needs some way of knowing the location of everything it has previously created, to prevent creating new items in already occupied or invalid areas. This applies for both generating the map (rooms and corridors) and decorating the map (adding objects and decorative items).

4.1.1.4 – REQUIREMENT 4 - THE MAP MUST BE COMPLETE

The finished map must all be connected; All areas must be accessible. This may not be guaranteed during the initial generation, and therefore must be checked and rectified at a later stage if required.

4.1.1.5 – REQUIREMENT 5 - GENERATION TIME

Since a map is generated in real time upon request, it's imperative that it's created quickly. If the generator is slow, the player would be forced to wait for an extended period of time, most likely looking at a loading screen. This will decrease the users satisfaction and could cause them to give up and quit the game if left waiting too long.

Therefore, it should be my goal to keep generation time down to a minimum, ideally taking no longer than a few seconds. If generation times turn out to be longer, I need to identify what part(s) of the generation process are taking a significant amount of time and decide whether this can be improved to a suitable degree, or whether this method of generation is actually viable for real time generation.

4.1.1.6 – REQUIREMENT 6 -FLEXIBILITY

As I develop the generator, I will be required to make decisions on many parameters that will directly impact how the overall map will turn out, such as map size, room sizes and chances of specific room types). The numbers that I initially pick may not turn out to be ideal, or become no longer suitable later in development.

Where possible, values that will have a weight in how the map will generate, should be implemented in a way to allow their value to be easily changed for further adjustment/future experimentation. These variables should also be editable in the final product, to demonstrate what each does and what affect they have on the map generation.

4.2 - DESIGN

This section covers and justifies the development process of the created UI and the software architecture.

4.2.1 – DEVELOPMENT OF THE UI

This section details the design development of the UI, showing the initial sketch and the prototype design.

4.2.1.1 – MAIN MENU

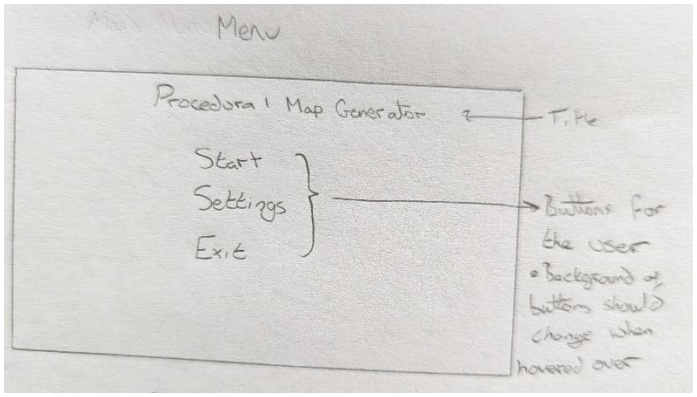


Figure 7 – A sketch of the menu

For the main menu, I decided I wanted to keep it relatively basic, for easier use. A title would be present at the top, followed by three centred buttons to start and exit the game, and to allow for changing any settings. Below shows the sketch for the settings menu:

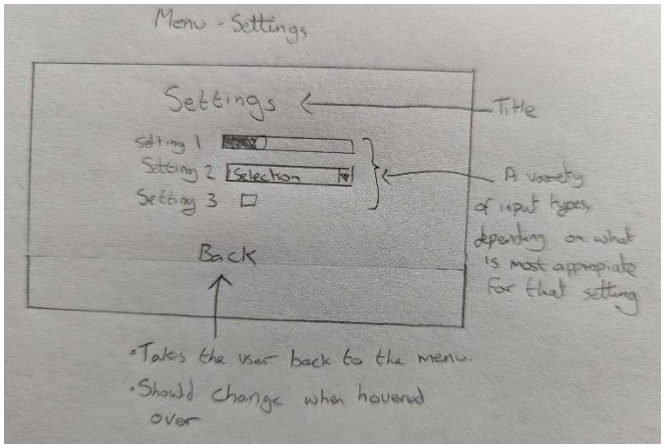


Figure 7 – A sketch of the Settings menu

The settings page would again keep a simple layout. A title, detailing the nature of the page would be located at the top, followed by a list of changeable settings, with the most suitable input types. A button to take the user back to the menu is present at the bottom of this page.

Below shows the prototype design for the menu:

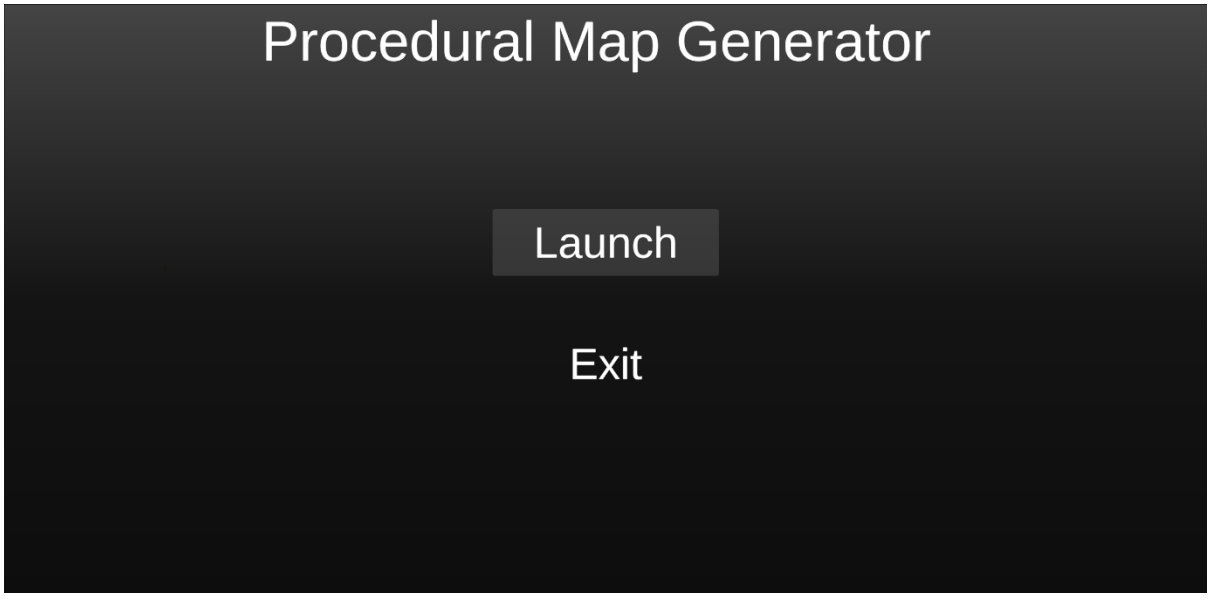


Figure 9 – The prototype design of the menu

A gradient background was used to improve the appearance of the screen. When buttons are hovered over, as shown with the 'launch' button, an outline becomes present, providing visual feedback to the user.

The 'Settings' option was not added, since I didn't believe it would serve any use for the current implementation of the application.

4.2.1.2 – MAIN SCENE UI

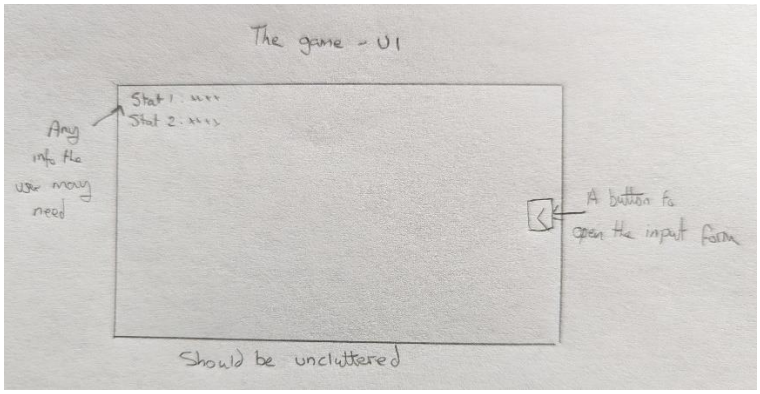


Figure 10 – A sketch of the UI

I decided that any information that the user may need during exploring the map should be in the top left corner of the screen. A button should be present on the right-hand side, that will open the menu to alter generator settings. Below shows the prototype of this screen:

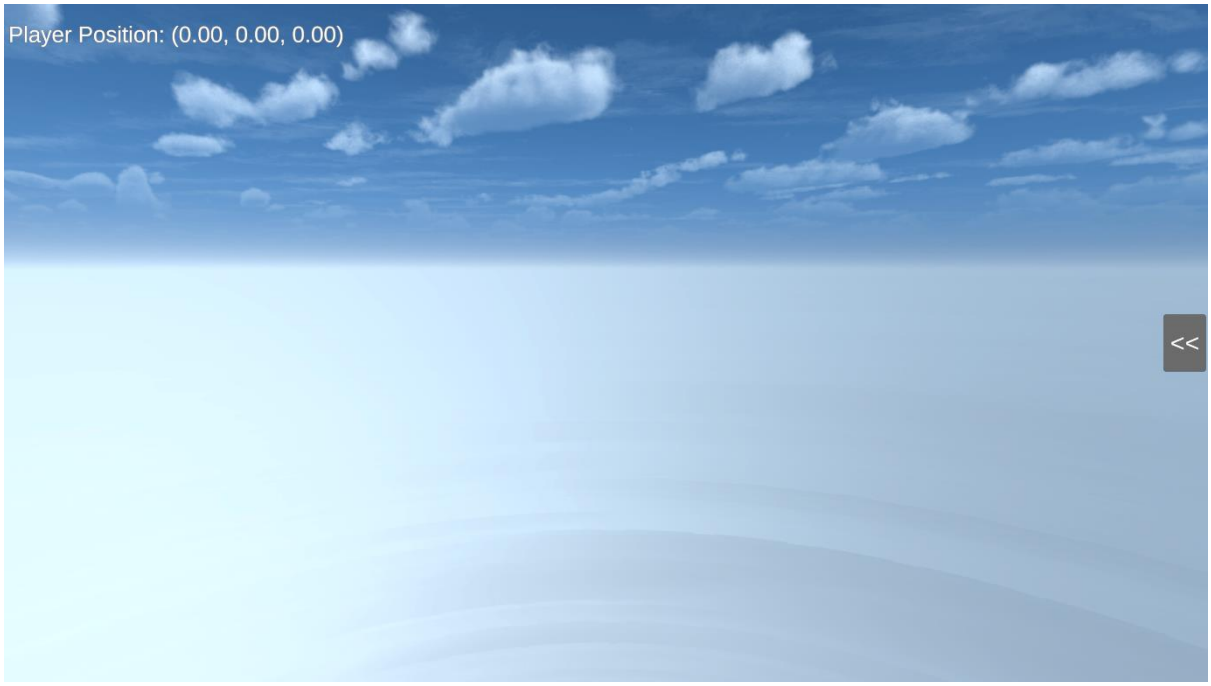


Figure 11 – UI of the main scene

It was decided that the only information the user may need is the players current position, so they can report the location of any bugs found. Other information to recreate the map where the error occurred is located in a generated log file. This was deemed a better approach, as to not clutter the screen or confuse the user with too much information.

The initial design of the input form, where the user can alter the generator settings, was designed as shown in the image below:

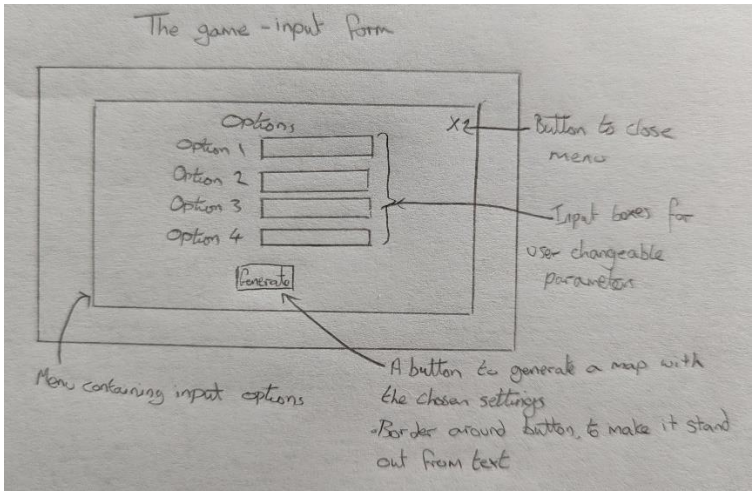


Figure 12 – A sketch of the input form

This consisted of a list of options that the user can fill in, along with a button to close the form down, and a button to begin the generation of the map. The prototype design is as follows:

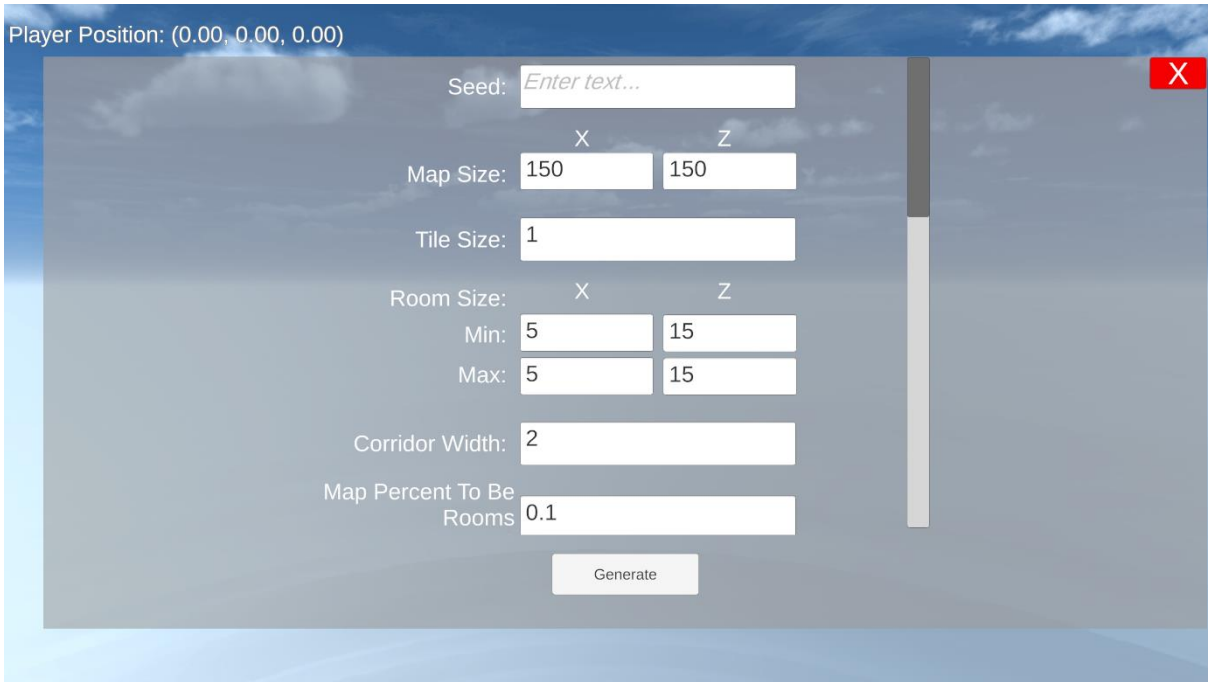


Figure 13 – The prototype design for the input form

The design is similar to the sketch, with two differences: Extra labels were included to make the form easier to read, and a scrollbar was added, since there were too many options to fit on a single page.

4.2.2 – SOFTWARE ARCHITECTURE

This section provides a class diagram of the software architecture, and details the function of all created scripts:

4.2.2.1 – CLASS DIAGRAM

Below shows the class diagram for this project.

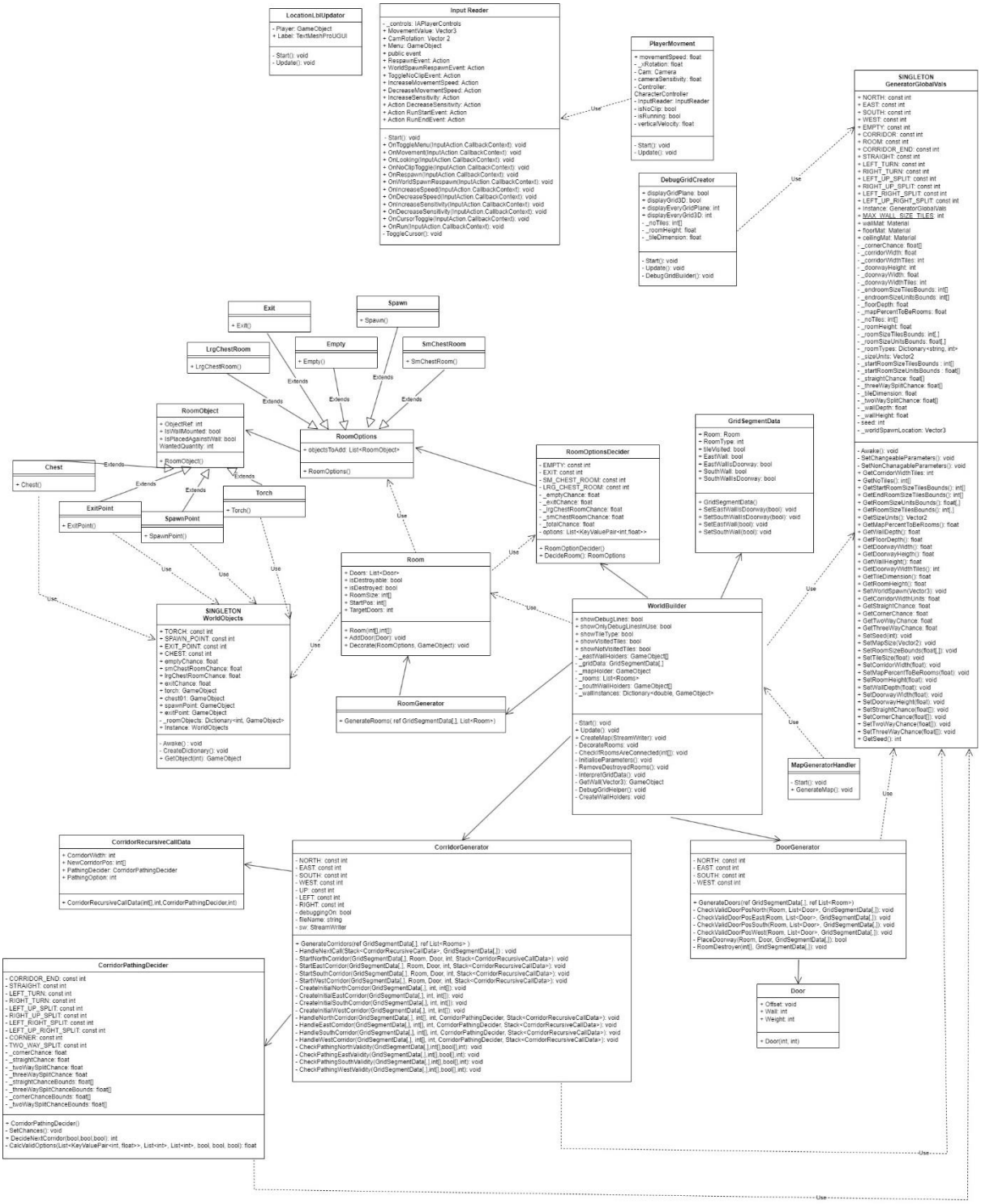


Figure 14: Class diagram

4.2.2.2 – MAPGENERATORHANDLER

The job of the script is to set up and execute map generation upon the user's request, by pressing the 'Generate' button. It also populates all input boxes with the default value for their respective parameter.

This script holds a private global copy of the default value of all user changeable parameters, set at instantiation. Global fields for references to all input boxes are also present. These are Serializable,

allowing them to be set in the Unity editor, and have a public get and a private set. There is also a reference to the used instance of the WorldBuilder class.

The class has 1 function: GenerateMap. This is executed when the user presses the 'generate' button. It checks the input boxes for any values, and alters the values of their respective parameters, located in GeneratorGlobalVals. If the input box is empty, the default value of the parameter is used. The selected values are written to a file, beginning the log file for this map.

A call to the WorldBuilder class is then made, which handles generating the map.

4.2.2.3 – GENERATORGLOBALVALS

This singleton holds the values of all user settable parameters, other values that are derivative of the user settable parameters, and other parameters that will be used throughout the generation of the map. It has the function of validating user input values, and setting the values of the derivative parameters.

This class also holds constants, used to identify choices when there are more than two options. For example, it holds integer constants for each of the corridor segment options (including straight, corners and splits) used when creating corridors, of which there are 8 possibilities.

4.2.2.4 – WORLDOBJECTS

This singleton holds variables related to room decoration, such as the chances of each room type and a dictionary holding references to used Game Objects, which can be retrieved using an integer value, relating to constants also located in this class. For example, the using the constant 'TORCH' will in the dictionary will retrieve and instantiate a copy of the torch Game Object.

4.2.2.5 – WORLDBUILDER

This script oversees creating the map and destroying the previous map, if present. This destruction is done by calling 'Destroy' on the parent object that holds every object that's created by the generator.

When the method 'CreateMap' is called, this class will generate the map, instantiating required parameters before sequentially calling every class that has some role in the generation of a map. Additions to the log file, created by MapGeneratorHandler, are made as required.

This class also holds the method 'InterpretGridData,' which translates the 2D array of GridSegmentData classes, '_gridData,' into objects, such as the floor, ceiling and walls.

A method used to create debug lines for debugging purposes is also located in this file.

4.2.2.6 – GRIDSEGMENTDATA

This class holds all information regarding each segment of the map, which will be altered throughout the maps creation and interpreted by WorldBuilder's 'InterpretGridData.'

4.2.2.7 – ROOMGENERATOR

This is the first method called by WorldBuilder that allocates areas for rooms. Rooms are continuously generated until a set percentage of the map is allocated as rooms, or 1000 attempts at the placement of a new room fail. Details, such as the position and dimensions of each room are stored in the Room class, which is added to an array.

4.2.2.8 – ROOM

Contains all the information needed regarding the room it relates to, such as its size and door positions. This class also places decorations, when called at a later point.

4.2.2.9 – DOORGENERATOR

This script cycles though every room and calculates where doors could be validly placed. For a door to valid, it must be completely on the wall of the room, not overlapping with any other added doors and be in a place where a corridor can begin generation. Up to the number of doors specified in the corresponding Rooms class are placed.

If there are no valid places for a door, the room is destroyed, unless it is an important room (such as the first 2 created rooms, which are designated the spawn and exit room), in which case a different room is destroyed that will free up space for the placement of a door on this room.

4.2.2.10 - DOOR

Holds the information of a valid door position on a given room. Created and used by DoorGenerator to pick door locations.

4.2.2.11 – CORRIDORGENERATOR

This has the task of generating corridors from every door not already connected. Decisions based off the available space and what would/wouldn't be a valid option are used to choose the initial starting position and the preceding decisions to create a corridor network. The creation of the corridor is done recursively until all branches are connected to something else, or cannot path to anywhere else (creating a dead end).

4.2.2.12 – CORRIDORPATHINGDECIDER

Used by CorridorGenerator to decide what to add next (continue straight, turn left/right or split into multiple directions). It is provided information by CorridorGenerator, which it uses to calculate which options are valid. It then randomly picks an option, taking into account the weights assigned to each choice, and returns an integer value that indicates what choice was made.

4.2.2.13 – CORRIDORRECURSIVECALLDATA

Holds data, used by CorridorGenerator, to allow for recursive calling. This was required to fix a problem due to a Stack Overflow error when calling recursively normally.

4.2.2.14 – ROOMOPTIONDECIDER

Used to decide the nature of every room, based of weights and a random number generator. This dictates which RoomOption to use.

4.2.2.15 – ROOMOPTION

The inherited class of all room options. Holds a list of RoomObjects, that are required to be placed in the given room.

4.2.2.16 – EMPTY, EXIT, LRGCHESTROOM, SMCHESTROOM, SPAWN

These are all classes that inherit RoomOption. They define what RoomObjects are to be added to that room type, using a random number generator to randomize the quantities of said RoomObjects.

4.2.2.17 – ROOMOBJECT

The inherited class of all objects that may be placed in rooms. This holds information relating to the id of the corresponding object, so it can be fetched and instantiated from WorldObjects and what restrictions there are to its placement, such as whether it needs to be wall mounted.

4.2.2.18 – CHEST, EXITPOINT, SPAWNPOINT, TORCH

These are all classes that inherit RoomObject and detail the specifics of that object, as mentioned above.

4.2.2.19 – INPUTREADER

This script utilizes Unity's InputActions, defining what to do, or what event to trigger, for every created input action.

4.2.2.20 – PLAYERMOVEMENT

This script subscribes to InputReader events and moves the player controller, based on what the user presses or moves.

4.2.2.21 - MAINMENUCONTROLLER

All menu options that don't toggle the visibility of said menu or other menus are handled here.

4.2.2.22 – LOCATIONLBLUPDATER

Updates the label located in the top left corner, detailing the current position of the player in the scene.

5 – DEVELOPMENT AND IMPLEMENTATION

This section will detail the development and implementation process that occurred throughout this project, justifying my choices I made throughout. I will also discuss the issues that were encountered and the found solutions.

5.1 – THE FIRST ATTEMPT

5.1.1 - DEVELOPMENT AND IMPLEMENTATION

Development first began by figuring out how to create a room of a specified size, in a specified place. Although the task seems to be rather trivial, this step took me many hours to complete, due to my relative inexperience with using Unity. Some formulars also had to be calculated, to place walls in the required places to create an enclosed space of a specified size.

After some time, I succeeded with this task, creating the following:

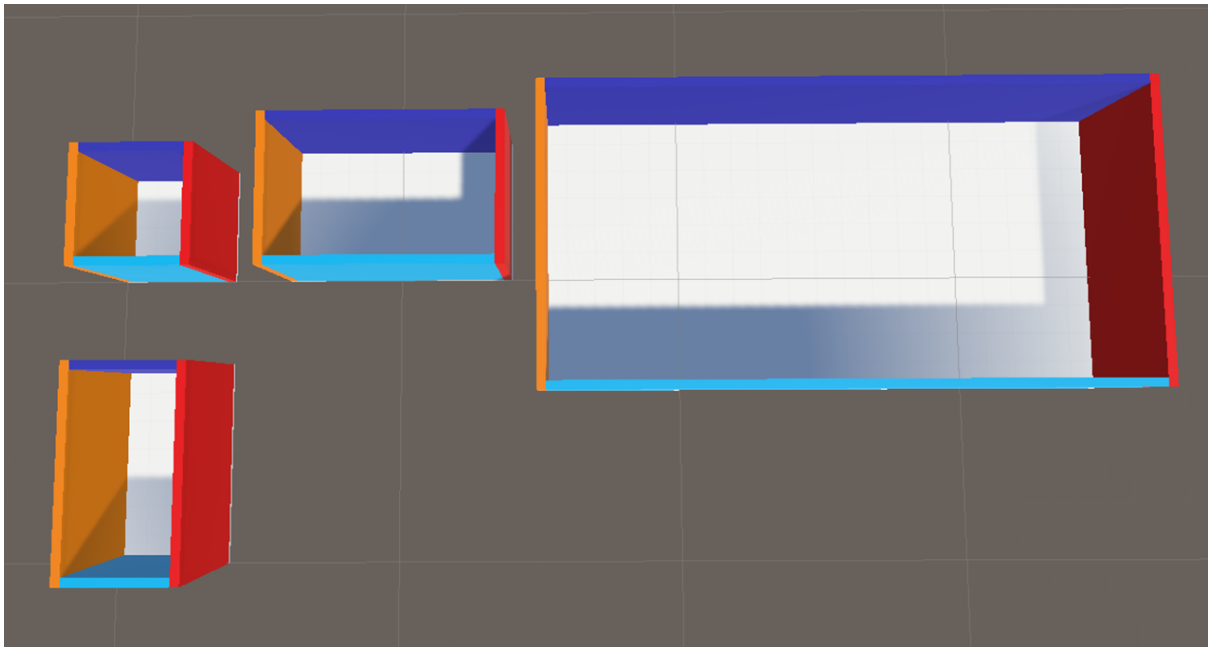


Figure 15 – Generated rooms of different sizes in different places

As the above image shows, I had managed to make multiple areas that could function as rooms, of different sizes, in different places. Currently, all placements are fixed: There is no level of randomness yet implemented, since I needed to work out how to do it manually before I could do it procedurally.

5.1.2 – ENCOUNTERED ISSUES

Developing this helped me identify three issues with my proposed solution early on:

The first was that not all the walls can be the length of the room. Doing this causes the walls to overlap in the corners; Something that would be noticeable from the outside, as the meshes merge together. The solution had been implemented in figure 15: Two walls (the dark and light blue walls in the image above) are required to be shorter by twice the width of a wall. This make it so the walls are now touching instead.

The second was that there is no way for two rooms to share the same wall. This was less of an issue with the generation of rooms but could prove to be a big issue when corridors are generated, where sharing the wall may be required. However, no solution for this was created, due to the far more pressing issue detailed below.

The final issue was considering how will the generator know what spaces were already occupied, and where is still free? Without any proper structure to guide the generator, this would be very challenging with my current approach. Keeping track of every occupied area of space would be costly, and checking would get continuously worse throughout the generation, as every area would require checking. Likewise, using colliders to work out what areas are already taken would be costly, and not be very useful in working out where there is space left.

If a solution was found, the likelihood that it would be fit for a real time map generator seemed slim. As such, I started again, this time opting for a grid-based solution.

5.2 – CREATING A GRID-BASED SYSTEM: THE INTERPRETER

5.2.1 - DEVELOPMENT AND IMPLEMENTATION

Although such a solution is more restrictive, I believe that it was the only way to approach this task with the solution I had in mind. I decided that each section of the grid would be 1 Unity unit in length and width, but I would implement it in such a way that everything would still work if this number was to change. This way, the option to regain more freedom of placement and sizes would still be possible, at the cost of generation speed and the required memory to keep track of more segments.

This created grid was handled by a 2D array, named _gridData, which help GridSegmentData objects. The idea was that GridSegmentData would hold any information that related to the segment in the grid it represented.

Before I began attempting to create rooms with this new solution, I first needed to implement 2 features:

I needed a visual representation of the grid, that could be toggled and customized as required. After experimenting with various solutions, I settled on using Debug.Draw lines:

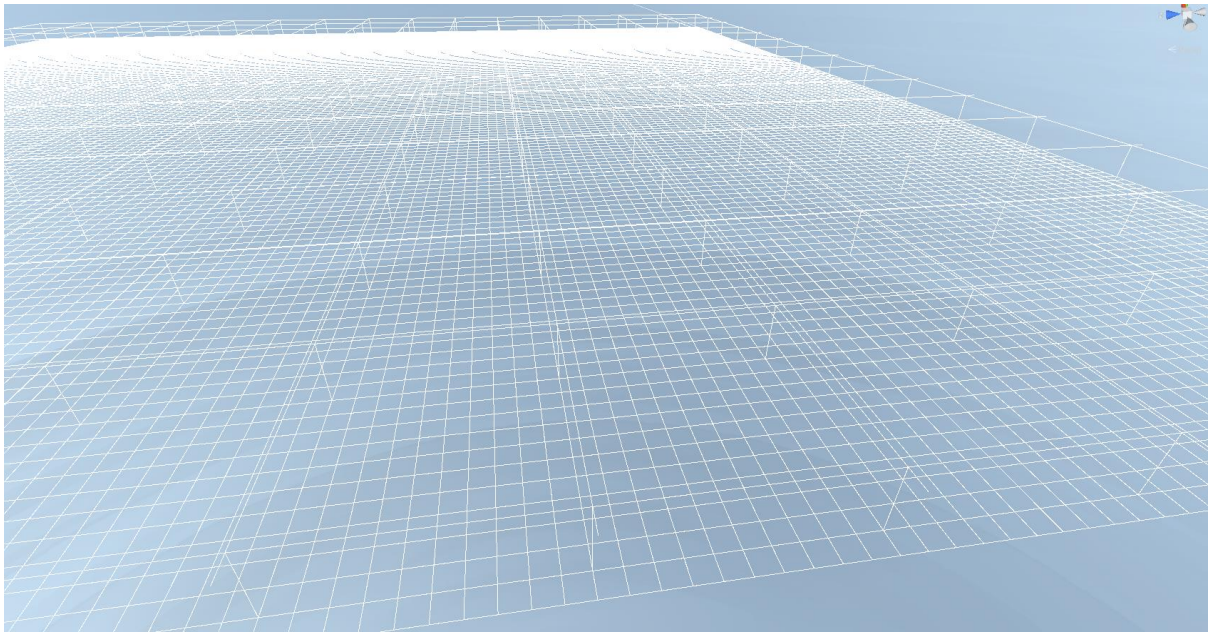


Figure 16: A visual representation of the created grid

A version that could be seen in the game view without toggling Gizmos would've been favourable, however when attempting the same grid with Lines, there was a significant fps drop, making it hard to navigate around the scene.

With this grid now implemented, it was now possible to ensure objects were being drawn in the correct places.

The second required feature was a grid interpreter. Since changes of the world were being represented in an array, the objects to visually represent what is being created are no longer done at the same time. Instead, all objects are now to be added to the scene once the design of the map has already been decided. This also made more logical sense, since any changes needed throughout the generation only had to affect the array, and not already instantiated objects in the scene.

It was decided that each grid segment would oversee defining the presence of a wall to its South and East border, with the border rows that connect to segment [0,0] would not be allocatable as a traversable area. This allows for all walls in potentially traversable areas to be accounted for, whilst having no tile that is in charge of the same wall section as another tile.

To explain what is meant by 'South' and 'East,' the 2D grid is only interested in 4 directions: up, down, left, right. Since using up, down, left and right are subjective to the direction you are facing, I instead opted for compass direction. 'North' is defined as facing the 0^{th} row of the grid that is perpendicular to 'x' axis Unity uses.

With this in place, development of the interpreter began. It works by iterating over the array, looking for where walls in the Southern direction need to be placed. When one is found, it checks if neighbouring tiles also want a wall to be placed in the same direction. This continues until the end of the row is reached, or a tile that doesn't want a wall is found. A wall is created, of the size that is needed to represent all the tiles that wanted the wall in the specified direction. This process continues until the end of the array is reached, in which it is iterated over again, this time adding walls in needed in the Eastern direction.

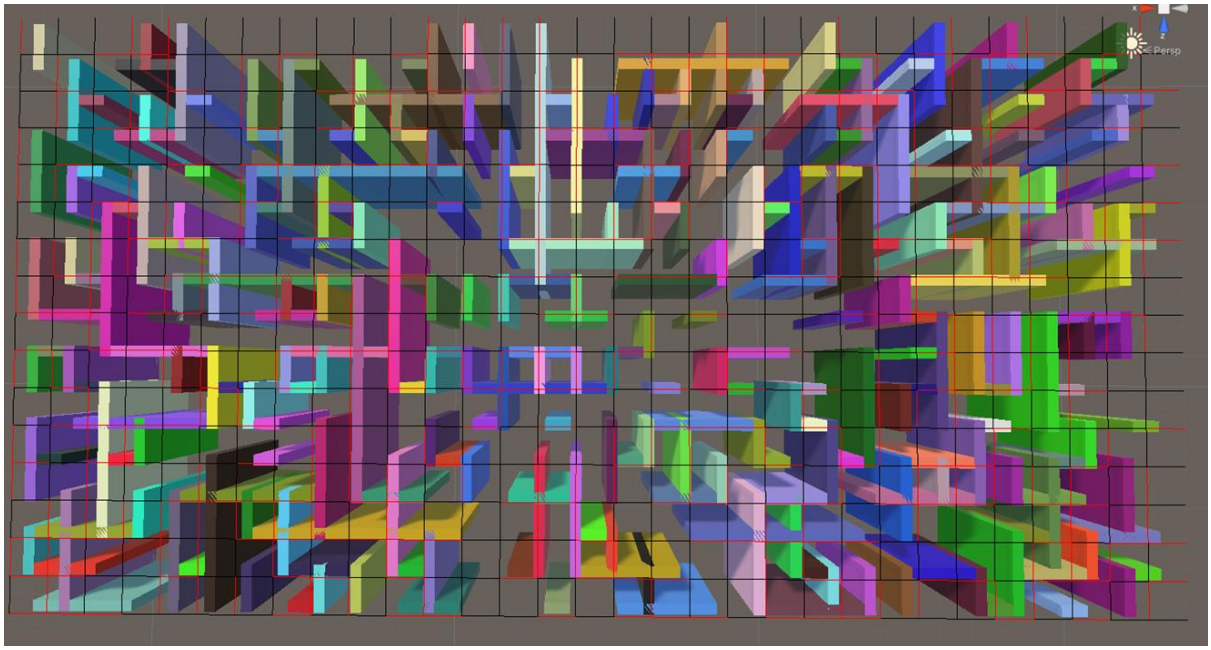


Figure 17: Wall sections created by the interpreter after development.

Figure 17 shows the interpreter drawing walls, which positions were randomly chosen. Another togglable Debug grid was also created, which only focuses in 2 dimensions. The purpose of this grid is to show where walls should be (indicated with red) and shouldn't be (indicated with black).

5.2.2 – ENCOUNTERED ISSUES

As figure 17 shows, there were issues with walls drawing inside of each other again. This was rectified by making Southern walls always dominant, taking priority, and connecting to other Southern walls if possible. Whereas Eastern walls will only connect to other neighbouring Eastern walls if there isn't a Southern wall in the way. They are also shorter, so they touch connecting walls, rather that going through them. Figure 18 shows this change in place.

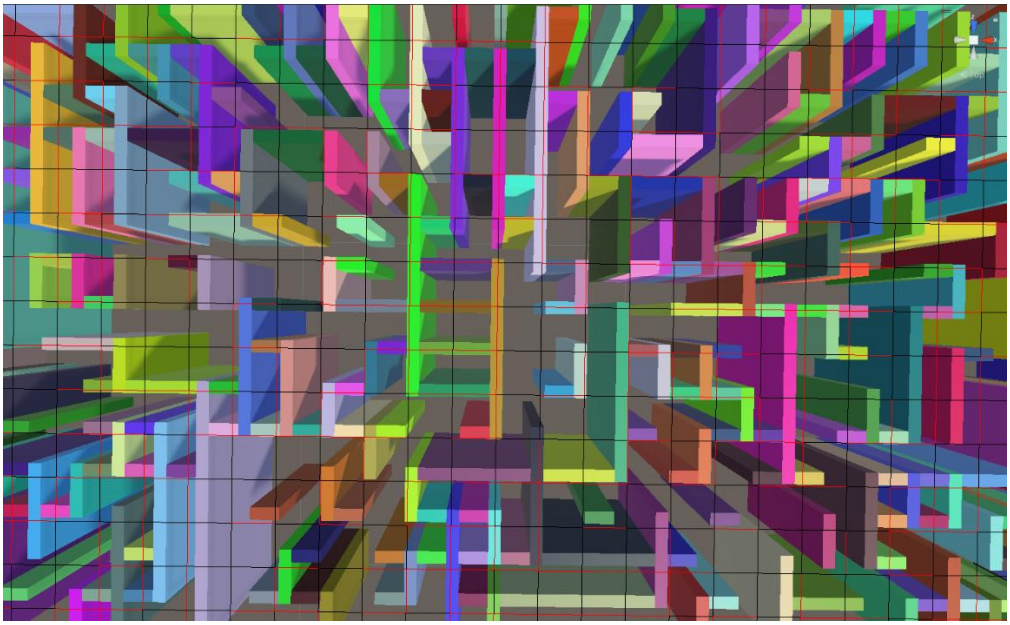


Figure 18: The grid interpreter once domination was implemented.

5.3 – REIMPLEMENTING ROOM GENERATION

5.3.1 – DEVELOPMENT AND IMPLEMENTATION

The next step was to reimplement a method to add rooms to the map. This required picking a size for the room and the place in the grid it is to go. Once picked, modifications to the grid array are carried out to represent the positions of the walls, with all tiles in the contained area being set as type 'Room,' represented as an integer number.

To generate multiple rooms, the same process is carried out. However, before confirming the placement of the room, the to-be occupied area is checked to ensure none of it has already been assigned. Since only rooms are generating at this stage, rather than having to check every tile in this area, only every x tiles are checked, along with the extremities (furthest Southern tiles and furthest Eastern tiles), where x is equal to the smallest size of a room in that given direction. The reduction in checks allows for faster overall execution.

Rooms are continuously placed until one of two conditions are met:

- The total percentage of tiles that are assigned to rooms surpasses the target goal.
- 1000 unsuccessful attempts at placing a room are made.

In the case of the latter, the upper bound for the room size is decreased and further attempts are made. This continues until the minimum room size is equal to the maximum room size. If the target goal cannot be reached at this stage, room generation stops, proceeding with the next step in generation. This is usually only the case when the target goal exceeds 60% coverage of the map. However, this will vary with different settings.

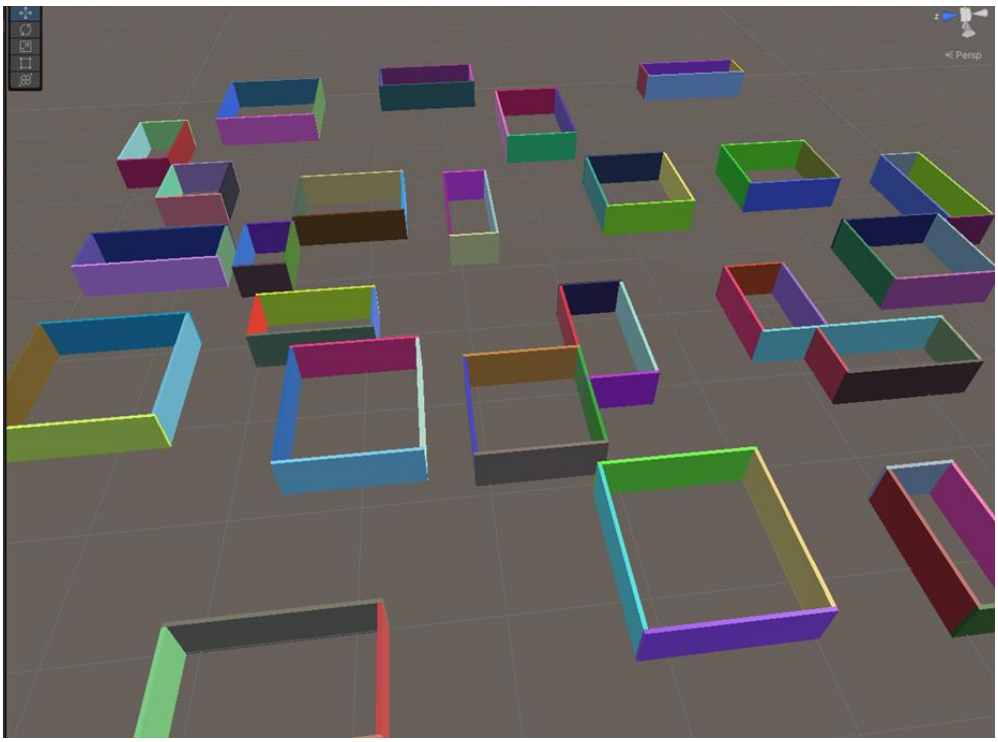


Figure 19– An example of rooms being generated.

Once room generation is complete, the interpreter is run, creating walls where required, as shown in figure 19.

5.3.2 – ENCOUNTERED ISSUES

At this point, I now switched to adding a material more suitable to this game to the wall. However, this outlined two major issues, both relating to the same problem:

The problem was with how Unity handles Cube objects, which were used to create the walls. In order to alter the size of these objects, you have to 'Stretch' the shape. However, this also stretches any material applied. The material is also applied in its entirety, on every face. This created issues that figure 20 illustrates:

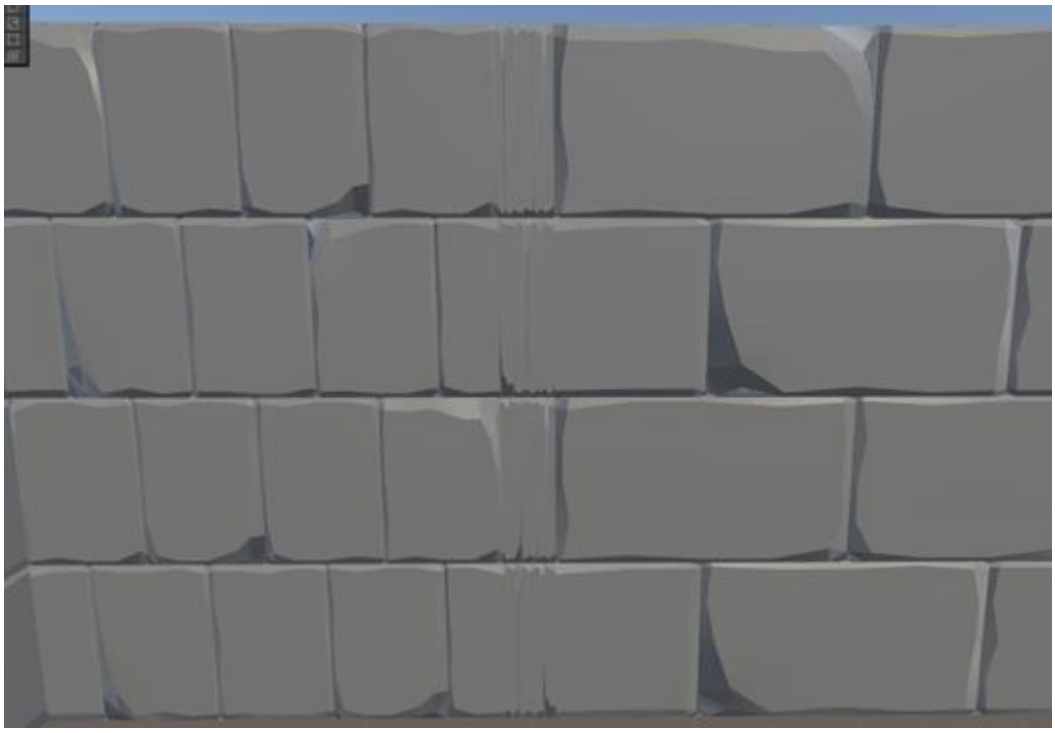


Figure 20: The issue with using Unity Cubes.

As the above image shows, the size of the material is inconsistent with the same texture no longer being the same size, creating visual issues.

One explored solution was to alter the tilling of the material. However, this causing the size to change on every face, making it not suitable unless your shape is still square. Each copy of a material with a different tiling value would also require its own draw call, harming the fps of the final solution.

Therefore, a different solution was required. After hours of investigation, I found a suitable solution: To utilize object creation from the Unity package ProBuilder. Every object takes longer to create using ProBuilder, but provides the trade-off of creating a custom mesh that scales materials appropriately on all faces.

To alleviate the reliance of using ProBuilder to create all of the walls, since it sometimes crashed after too many requests, as well as reducing the total custom meshes created, I saved the first copy of every sized wall in a dictionary during generation. If the same sized wall was required again, the dictionary would instead be used to instantiate a clone of the previous wall.

This solution allowed for correctly textured walls, with a minimal effect on overall generation speed, reducing the wall generation time by 30%, compared to just using ProBuilder.

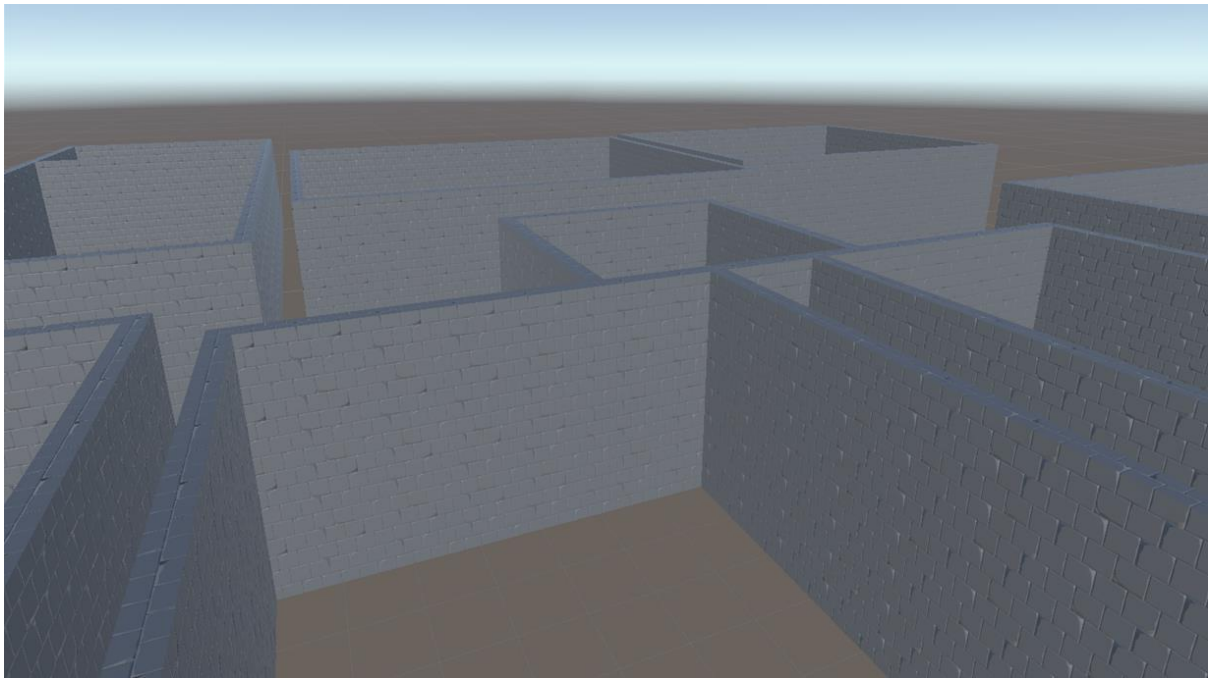


Figure 21: Generated walls after utilising ProBuilder

5.4 – ADDING DOORS

5.4.1 – DEVELOPMENT AND IMPLEMENTATION

The next logical step was to create doors for every room. This was done by iterating over the array that contained data relating to every created room, adding the required number of doors to each room if possible.

There were conditions that had to be met for a position to be deemed valid:

- The door must be completely on one of the walls of the given room.
- The door cannot be placed overlapping another existing door.
- A currently unassigned area of the corridor width by the corridor width (a value that was decided at an earlier stage, which can be changed if deemed necessary) must be present in a position that would allow it to connect to that door.

All viable locations were added to an array, with the one(s) chosen being selected at random. If a room has no valid positions for a door, the room is destroyed: The grid array is altered to make area the room had occupied now marked as unassigned, and all non-shared walls are removed.

However, there is an exception to this; The first two created rooms. These are to be designated as the starting and exiting area, and therefore cannot be destroyed. Instead, obstructing rooms on a randomly chosen side are destroyed instead, with doors then being added to the current room.

5.4.2 - ENCOUNTERED ISSUES

Apart from some logical errors, which were fixed, there were no major issues encountered with this stage.

5.5 – GENERATING CORRIDORS

5.5.1 – DEVELOPMENT AND IMPLEMENTATION

Now that every room has at least one door, the next step taken was to create a network of corridors, which connect to every door. Development of this took a significant amount of time, turning out to be complexed, with many cases that needed to be handled. The process of generating corridors is:

Iterating though every door for every room, a starting position for the corridor is determined, that will allow it to connect to the door, based off where it can be validly placed. There are two exceptions to this:

- The door is already connected to a corridor, therefore not requiring any further changes.
- A corridor cannot begin generation, due to a corridor being present close to, but not connecting with, the door. In this case, the door is connected to the existing corridor.

Once the position of the corridor is chosen, the job of creating corridors is delegated to the corresponding recursive method, detailed below.

Corridors can generate in four directions: North, East, South and West. Each ‘choice’ will result in the corridor to claim one extra row of tiles in the chosen direction. As this is the case, the exact steps to make this happen are different for each direction, so each is required to be handled separately.

A corridor is each direction can choose one of many options: End the corridor, continue straight, change direction, and split into multiple directions. There are eight possible options for the next corridor piece, all requiring to be handled in each direction.

Once the next segment of the corridor is added, the next segment needs to be handled, providing the choice wasn’t to end the corridor. This requires calling the method, or methods in the case of a split, that will handle the next segment of the corridor in the given direction.

This process continues, recursively calling the four methods for corridor generation as required, until the either the corridor ends, the corridor connects to another door, or the corridor connects to an existing corridor.

Figure 22 shows a map once corridors have been generated, indicated with green lines.

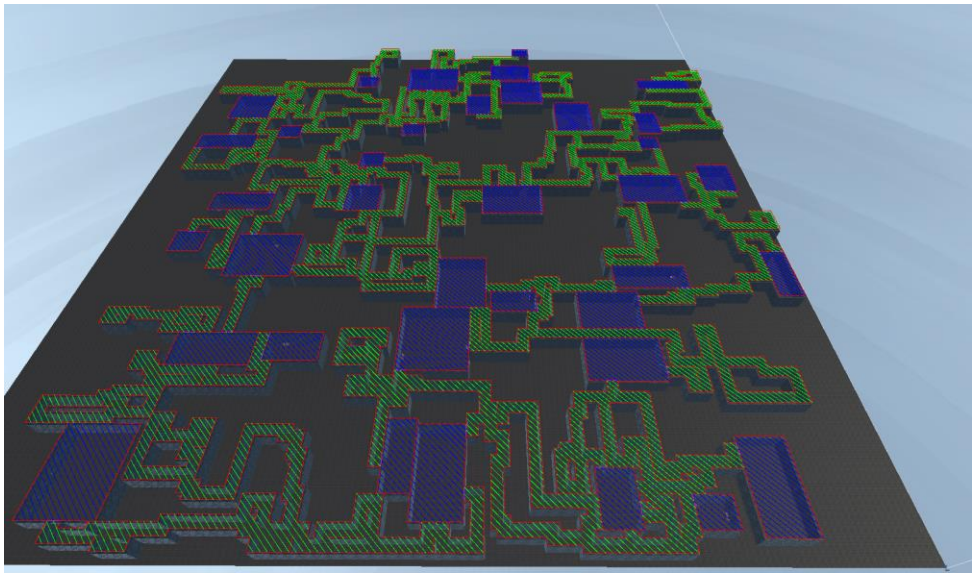


Figure 22: The map once corridors are generated.

5.5.2 - ENCOUNTERED ISSUES

The recursive nature of corridor generation proved to be a significant problem during development. Since may calls are made, the likelihood of reaching a high recursion depth is almost guaranteed. Unfortunately, Unity has a max recursion depth limit, which triggers a Stack Overflow error. Nothing can be done about this, since it's a Unity internal setting.

After a significant amount of time, I was able to implement a solution that allowed me to continue calling recursively, circumnavigating Unity's stack. The solution required making my own stack of CorridorRecursiveCallData. This is a class that I created that contains all the information needed to preform the next recursive call: The data that the recursive function needs, along with an integer that represented which of the four methods needed to be called.

With this solution, the previous method finishes before the next is called. This allows for the same calls to occur as needed, without creating any further recursion depth. This prevented the Stack Overflow error from occurring, allowing for all corridor pathing to execute unhindered.

5.6 – DECORATING ROOMS

5.6.1 – DEVELOPMENT AND IMPLEMENTATION

Now that the structure of the map was complete, I turned my attention to the last aspect of this project: Decorating rooms. I decided to go about this by defining what the contents of a specific room has in a class, which inherits the superclass RoomOptions. Each room is assigned a RoomOptions, based off weights assigned to each room type.

Each object that may be added also are defined in their own classes, inheriting the superclass RoomObject. This defines the reference to the object, and well as placement restrictions.

Once the interpreter finished execution, each room is iterated over, placing the objects defined as wanted by that given room.

This method of defining rooms and objects should allow for more room options and objects to be added with relative ease.

5.6.2 - ENCOUNTERED ISSUES

Originally, the plan was to also spawn AI's that would guard rooms and patrol the map, since a combat system is very common in this genre of game. However, I found out there was a critical issue that I would be unable to fix.

In order for AI's to path find and navigate around the map via a Nav Mesh agent, the map must be baked: A process that calculates where the AI can and cannot go. However, I found out at this stage that Unity doesn't support baking the map at run time, making adding AI in this way impossible.

Unfortunately, I was not able to find a solution to this, apart from using a different game engine, and therefore was unable to add any AI.

5.7 – CREATING THE UI

This final step involved preparing the application for user interaction, for the presentation. This required the addition of a main menu, a basic UI detailing anything the user may need to know, a way to alter changeable options for the generator, and means of navigation around the scene.

Details on the created design can be found in Section 4 of this report.

6 – TESTING AND ANALYSIS

This section will cover testing performed during development, created test plans for the final solution and user testing.

6.1 – TESTING DURING DEVELOPMENT

Testing proved to be quite challenging throughout development, since everything is procedurally generated, and therefore you have limited indications as to whether everything is working. Similarly, not every seed will see every possible option, requiring the testing of many seeds to hopefully identify all the bugs.

Most testing relied on seeing if objects were being placed in the correct place. As such, I created two toggleable debug grids, which help me visualise the boundaries of each segment, to ensure objects are indeed being placed where they should.

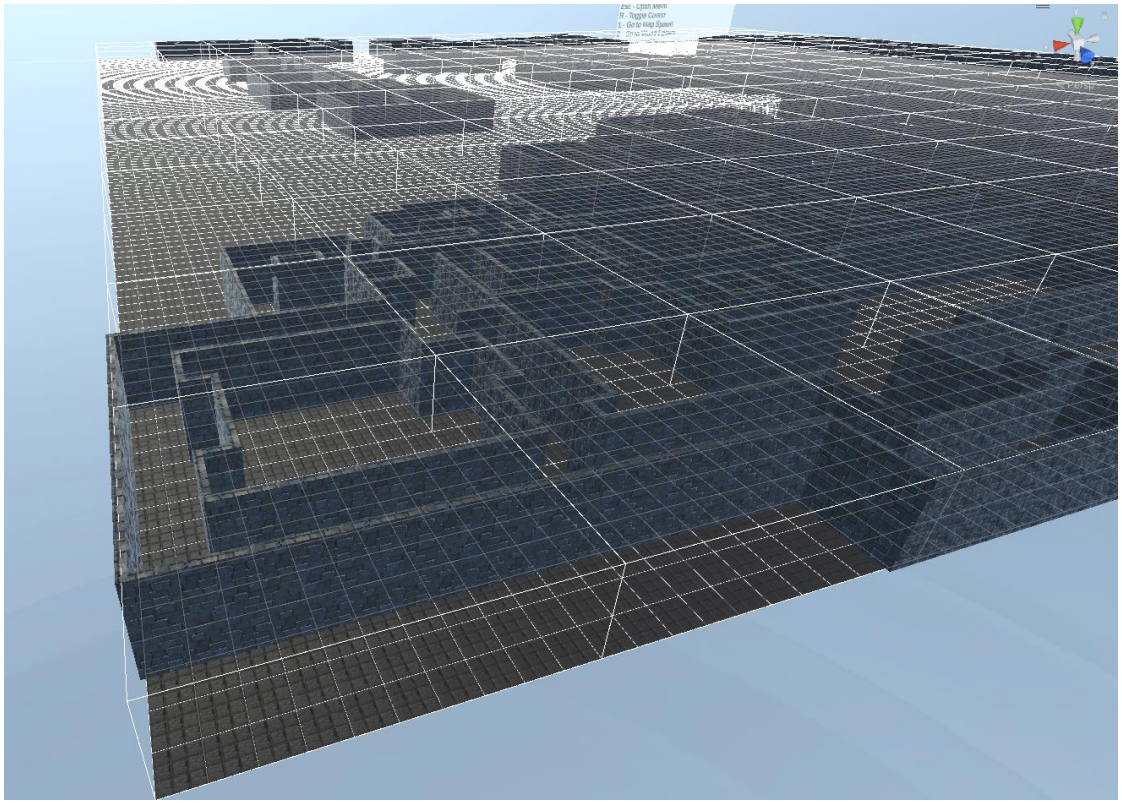


Figure 23 – The first debug grid

Figure 23 shows the first developed grid, which was used to check if the interpreter was drawing objects in the correct places. Lines are generated every x segments (a value that can be changed in the editor), showing the boundary of that area. A 3D representation can also be used, showing what height each room should be.

This provided vital for developing the interpreter, since there would be no way of knowing whether objects were being generated in the correct space and of the correct size without it.

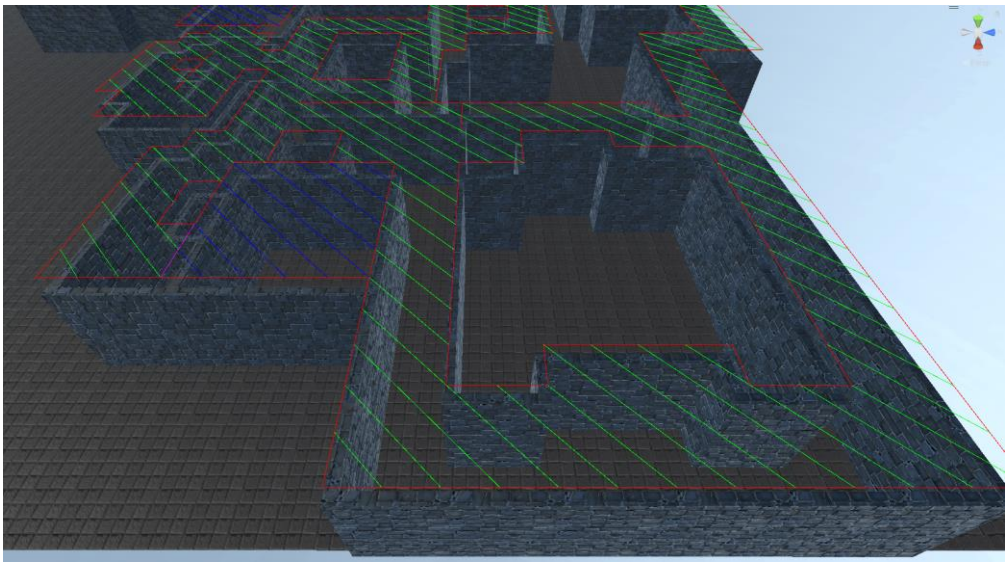


Figure 24 – The second debug grid

Figure 24 shows the other created debug grid. This one, rather than focusing on the boundaries of each segment, is only interested in the tiles that have been assigned. As the above image shows, a red line was used to represent a wall, a pink line was used to represent a doorway and the tile types are represented with different coloured diagonal lines.

This proved very useful throughout development, as it visually showed that all assigned tiles were correctly bounded. This helped to identify many cases whereby the walls were assigned to the wrong tile, failing to correctly bound a room or a corridor.

This also helped to identify multiple issues during corridor generation, when walls were removed in places they should've remained, and a case where corridors began pathing into a room.

During development, there were two kinds of kinds of errors: Fatal errors that would throw an exception, and logical errors that would see something unintended happen.

My strategy for testing fatal errors was to run the program over 1000 randomly selected seeds. This process is automated and takes a few minutes. If no error messages appear, then no fatal errors were encountered. I also retest multiple times, after altering the changeable parameters. If no fatal errors occur, then I can say that there are most likely none or are very rare to be encountered.

However, most errors fall into the other category. Unfortunately, I do not believe an automated test can be carried out for these cases, since the final product changes every time. Instead, I must manually look across the map for anything that doesn't look right. This is very time consuming, only allowing me to test a small quantity of seeds, reducing the likelihood of finding every issue.

6.2 – TESTING PLAN

There are two areas that require testing: User controls, and the success of the map generator with different settings. Below details the Testing Plan used.

6.2.1 – PLAYER CONTROLS TESTING

Test No	Test Type	Test Name	Purpose of test	Expected Result	Actual Result	Outcome and Required actions
1	Player Controls	Player Exiting game	To ensure the player can exit the game, since this cannot be tested from the editor	The application closes	As expected.	None
2	Player Controls	Player Movement	To ensure that player inputs are working correction.	The player can move around.	As expected.	None
3	Player Controls	Player Movement during low fps.	To ensure the player inputs are still useable in the case of	The player can easily move around.	Camera rotation becomes very hard to control. Other movement as expected.	Difficulty navigating the scene occurs during low fps. This should be investigated to make camera sensitivity a more consistent experience.
4	Player Controls	Player teleports to a spawn	To ensure that the player successfully teleports to either the world or map spawn, depending on their input.	The player is teleported to either the world or map spawn. No clip and gravity are toggled as appropriate.	World Spawn working as expected. Map Spawn can be chosen before there's a map, causing them to go to world spawn and start falling. Where there is a map, it works as intended.	Confusion may arise if 'Go to Map Spawn' is chosen before a map is present. This option should be disabled until a map is present.
5	Player Controls	Player opens and closes the menu	To ensure the menu shows and hides, and that the cursor is enabled and disabled.	[When opening the menu] Menu becomes visible. Cursor becomes visible and usable. [When closing the menu] Menu becomes invisible. Cursor becomes invisible and no longer usable.	As expected.	None.

Figure 25 – Player Controls testing.

The aim of this testing is to ensure that the player controls are functioning as expected. Any errors could cause the user confusion or be unable to perform the task they intend to do.

Some issues were identified and subsequently fixed.

6.2.2 – MAP GENERATOR TESTING

Test No	Test Type	Test Name	Purpose of test	Expected Result	Actual Result	Outcome and Required actions
1	Map Generator	Generate a normal map	To ensure the default map is generated.	A map is generated	As expected.	None.
2	Map Generator	Generate a custom map	To test the functionality of the input boxes	A map is generated, with the new settings applied.	Appeared to work.	None.
3	Map Generator	Generate a map providing no parameter values	To ensure that the default settings are used in place of a blank box.	A map using the default settings in generated. All input boxes now show the default values.	As expected.	None.
4	Map Generator	Generate a map with invalid values – Map size	To ensure that user inputs are overwritten where necessary.	The map size is overwritten, allowing for the successful creation of a map.	As expected.	None.
5	Map Generator	Generate a map with invalid values – Tile Size	To ensure that user inputs are overwritten where necessary.	The tile size cannot go any smaller than the width of the wall. This value is used instead, and a map is successfully created.	As expected.	None.
6	Map Generator	Generate a map – Different Room Sizes	To ensure rooms of many different sizes successfully spawn	A map is generated.	There appeared to be a case where a room was constructed, overlapping with another room. No other issues found	A room was placed in an invalid location, for an unknown reason. This requires investigating.

Figure 26 – Map Generator testing.

The aim of this testing is to ensure the input boxes the user may use to affect the generator are functioning as intended. This was also a further opportunity to visually check over more maps.

No issues were found with the input form during testing. However, an error with the generator was found when changing the room size bounds. The cause of this error is currently unknown and will be investigated.

6.3 – STRESS TESTING - THE LIMITS OF THE GENERATOR

The main drawback with my solution is the requirement to create the whole map in one go, which also requires the state of each segment to be tracked. As such, there is a physical limit to the maximum size a map can be on a given system. This maximum is bounded by the amount of available ram a system has. For this test, I decided to see what was the biggest map that I could make.

My pc has 24gb of ram. I opted from the greatest precision in the map generator settings, making each tile 0.25 Unity units in size. With this, I was able to generate a map of size 3000x3000 Unity units. This map is 400 times bigger than a standard map and successfully generated.

This test proves that the generator can generate maps significantly bigger than necessary, given the user has enough ram. A standard map size uses less than 1gb of ram to complete, making it suitable solution for most users.

6.4 - USER TESTING

To gain insight from others, a questionnaire was sent out to consenting participants, asking them questions regarding the generated maps. Overall, the generator was well received, in terms of its customisability and generation speed. The main suggested improvements are follows:

- Some participants suggested that the default values could be improved upon, suggesting that some rooms were too big, the corridors were too tight.
- Comments about certain rooms being too rectangular (long and thin) and didn't feel like they made for a good room.
- Some connects said that long corridors to nowhere were encounter too frequently, whilst others suggested removing this possibility completely.
- Some commented on the lack of things to do, such as fighting enemies.
- Comments about large rooms being very sparsely decorated were made.
- A bug was found whereby a small section of wall was missing.
- Some suggested that there needs to be more room types, providing greater variation.
- It was also suggested that there should be a variation in the floor, ceiling, and wall textures, especially for rooms, to create greater variation and to make the rooms stand out more.

This helped to identify many areas whereby I could improve upon in the future, to create a better map generator. Some of the above comments would require large changes to the way in which certain aspects of the generator works. An example of this is the comment regarding a variation in textures, which would require the rewriting of the interpreter, and the comment about how corridors sometimes create dead ends on long paths, which would require a major change to how the corridors generate.

Some changes would require less changes, such as undesirable room shapes, whereby a bound may be added, forcing the width and height of a room to be with x apart from each other. Similarly, a greater room variation can be achieved with relative ease.

The comment regarding there not being an enemies unfortunately cannot be acted upon, as mention in section 5.6.2.

Other comments, such as undesirable default values, can be acted upon, requiring some investigation to find more ideal numbers.

7 – CRITICAL EVALUATION

This chapter will focus on evaluating the overall success of the project, in terms of the product produced, time management and the lessons learnt. The first section will focus on seeing how well the created procedural generator achieves the initial proposed objectives, before reviewing the strengths and weaknesses of the generator. A review on the projects time management will then follow, finishing with the lessons learnt.

7.1 – REVIEW OF THE PROJECT OBJECTIVES

7.1.1 – Researching the Current State of Procedural Generation

This objective was achieved in Chapter 2, whereby extensive research regarding the many current proposed generation methods underwent. I believe this object was well achieved, as it helped to outline the strength and weaknesses of each solution, giving me a better understanding of what my solution must achieve. It helped to provide the basis of the idea I had, considering possible challenges I may encounter ahead of time. Without initially researching this field, I may have encountered many more issues during development, failing to create a final solution.

7.1.2 – PICKING AN APPROACH FOR MY PROJECT

I successfully picked an approach that allowed me to procedurally generate map as intended. Therefore, I would state that I was successful in achieving this objective. There perhaps could have been more planning before developing, that would have identified my original approach as not suitable. Nonetheless, I have successfully built a procedural dungeon map generator.

7.1.3 – GENERATION OF ROOMS

My solution can generate multiple rooms in valid positions, of different sizes. The width and depth of each room is decided randomly, based off set bounds that can be modified before execution the map generator. The height of each room is fixed and can be defined before creating the map. This should allow for any room size to be produced as wanted.

Feedback detailed in Chapter 6.4 suggested that the sizes of the rooms should be somewhat bound, to prevent too many long, thin rooms. I would also suggest adding a variable height option, allowing for tall rooms in places, such as a boss room. This is not currently a feature.

Doorways are generated at valid places on every room, of a specifiable quantity.

Overall, I would say that this objective was achieved, with some areas of improvement.

7.1.4 – CORRIDOR GENERATION

Corridors are generated from every created doorway, and path in the available space, connecting to other doorways and created corridors, creating a mostly connected network.

Currently, some sections of the map remain disconnected from the main section, resulting in them not being accessible. I did not have time to add a feature whereby disconnected sections would be connected.

The current implementation of the corridor generation is ‘dumb,’ in the sense that it doesn’t have any objective. It works out where it can path next, and picks one of the options based off their assigned weights. Ideally, it would be smarter, picking a doorway or an existing corridor it wants to connect to, and favouring options that would make this happen.

However, due to the complexity of getting any form of adequate corridor generation, along with the many bug fixes that were required, I did not have time to modify this any further. As point of reference, handling every necessary case for corridor generation to work took just over 2,400 lines of code. Further modification to this may take a significant amount of time.

Overall, although the corridor generation isn't as sophisticated as I planned, I believe it meets the requirements of this objective.

7.1.5 – WORLD DECORATION

I successfully managed to implement a system that would allow for the decoration of rooms, and the defining of room types. However, there is currently no way of decorating corridors. I do believe this to be possible, given more development time. As such, this objective was only half met.

7.1.6 – ADDING AI'S

Unfortunately, this proved to not be possible with Unity, as detailed in Chapter 5.6.2. This was a huge oversight when picking a suitable game engine to use. Not being able to bake a map at run time was not something I thought would be a problem, assuming it would be possible. As such, this objective was not met.

7.1.7 – CREATE A 'HOW IT WORKS' OPTION

This feature was not added, due to the premise not being compatible with how the generator works. For this to work, a visual change must occur at each important stage of development. This is not the case with my generator, whereby objects, such as floors and walls, are only added once the whole map has been created. Therefore, this objective wasn't met.

7.1.8 – USER CHANGEABLE PARAMETERS.

An interface was created, allowing the user to change one or more of multiple options that affect the generation of the map. The changeable options are: The seed of the map, the map size, the size of each tile, the size bounds for each room, the amount of doors each room may have, the width of the corridor, the percentage of the map to be allocated as rooms, the depth of the walls, the width and height of the doorway, the weights assigned to room types and the weighted bounds assigned to corridor segments.

Overall, I believe this creates a very customisable map generator, fulfilling the requirement of this objective.

7.1.9 – CREATING A PLAYER CONTROLLER

The player can navigate around the map, fly, toggle no collisions, teleport to both the map and world spawn, and can generate maps. I believe this is sufficient for a user to traverse and investigate a generated map, fulfilling this requirement.

7.1.10 – EVALUATION ON THE SUCCESS OF THE PROJECT

I believe the completion of this dissertation fulfils the requirement outlined by this objective.

7.2 – PRODUCT EVALUATION

The software I created provides a great start for the development of this method of procedural generation. I believe it is very versatile, allowing for many aspects of generation process to be easily customised, by altering the changeable parameters. Each map, using default settings, generates in under 1 second, providing very little delay for the user when used at run time.

There are a few alterations that could be made to further improve the success of the success of this generator, such as the ones mentioned in Chapter 5.6.2, and the addition of a more sophisticated corridor generator.

The greatest drawback of this project is its maintainability. Many sections of the generator are very complexed and are not very flexible outside of some parameters. For example, if I wanted to have some rooms/walls have a different material, a complete rewrite of the interpreter would most likely be required. This is not ideal but is the unfortunate nature of procedural generated content. It is very hard to add more content that you hadn't originally planned for.

Overall, I would state that the product was successful in what it set out to accomplish, but there are many alterations that can be made to further improve it.

7.3 – REVIEW OF THE PROJECT PLAN

Below shows the original time plan that was set for this project:

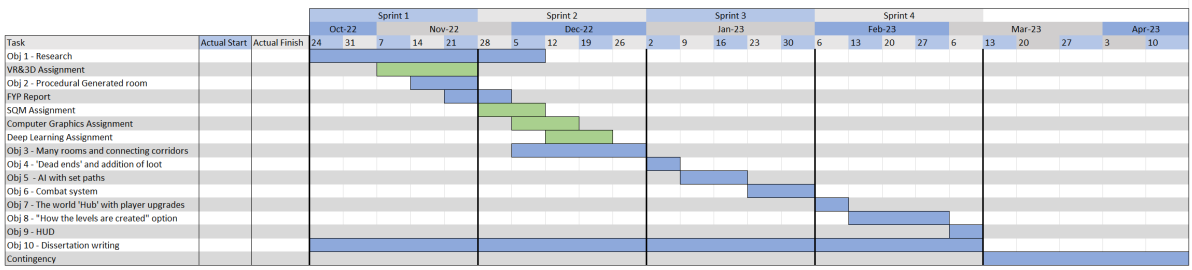


Figure 26 – The original time plan

However, the result turned to be significantly different from that plan. This was due to many factors, the most notable one being that the plan was made with little understanding of what this project would

entail, and how long other assignments would take to complete. Attempting to plan out the entire timeline for a 6-month period is not ideal and is likely hard to stick to.

Another reason as to why I ended up behind the original schedule was to do with multiple encountered issues, as detailed in Chapter 5. It was not possible to account for the severity of some of these issues, such as the Stack Overflow error. Although I did plan some contingency with the set time bounds, it was not enough to deal with significant problems such as that.

However, with this considered, I was able to finish on time with a product I can say I am happy with. For future products, I know that a greater contingency at each step is needed, rather that just one at the end. Keeping an up-to-date time plan would most likely be a useful tool to aid development, and to pick more realistic time boundaries.

7.4 – LESSONS LEARNT

This project has been a great test of many skills, as I went through all the processes of software development and project management. Doing so has given me many opportunities to improve upon my soft skills.

One of the lessons learnt was the importance of time management and organisational skills. This was a large project, requiring hundreds of hours to complete, which had to be done alongside other modules and life itself. In order to complete all of this, I had to balance my time between everything that needed doing, keeping notes of where I am up to, allowing for me to get back to working faster. I believe my organisational skills require further improvement, since I only relied on physical notes that I left for myself. I may have benefitted from using a Trello board, which can quickly represent what is done, what is currently being worked on and what needs to be done next.

Another key skill required was research and information gathering. This is something that I don't have much experience with, and therefore having to undergo a research section of the development process helped to improved upon these skills. I am now more proficient at finding useful sources on research paper search engines, such as IEEE and Google Scholar.

I believe this project has also increased my ability to code, and my understanding of how Unity works. Having to fix challenging bugs helps you realise what causes them, allowing you to re-evaluate how you might approach future sections to reduce the likelihood of running into the same error again. For example, the Stack Overflow error I received due to a high depth of recursion was a problem I didn't expect, with the solution taking long to implement, as I had to restructure how sections of my code worked. When I had to do recursion again later on, I used what I learnt to prevent getting this error again.

8 – CONCLUSION

This section will focus on reviewing the conducted work, comparing any findings to work of a similar nature, comments on legal, social, ethical, and professional issues and finally focusing on what further work should be conducted.

The beginning of this project began with a Literature review, looking over the topic of Procedural Generation, specifically regarding map generation. This identified many solutions, each with their

strengths and weaknesses. This information was then used to decide the approach I intended to take, before planning the development process.

Development then started, focusing on creating the generator, feature-by-feature, fixing any encountered bugs and issues as they were identified. Testing was conducted throughout development, with user testing being conducted once the product was completed. The details of development and the evaluation of the system was then conducted, in the form of this dissertation.

I believe that my generator has a lot of merit, successfully being able to create a 3D dungeon crawler. The generation speeds of the map are sufficient for a run time map generator, with maps of a sufficiently large size taking under 1 seconds to create, such as the image below shows:

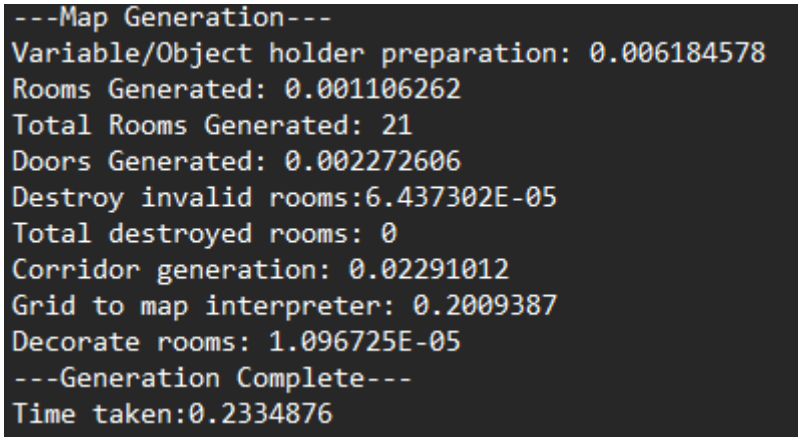


Figure 27: Recorded generation times of a map, using defualt settings, in seconds.

As figure 27 shows, the majority of development time, appoximately 86% in the example above, is spent interpreting the gird and creating the necessary objects. It may be worth instigating methods that may improve upon the interpreting speed, although I believe most of the time taken was due to the use of ProBuilder: A Unity addon that had to be used to scale materials appropiately. Better performance may be found if a different game engine is used instead. Below shows the results of a stress test, creating a 2000x2000 Unity units sized map, with the tile width set to 0.25:

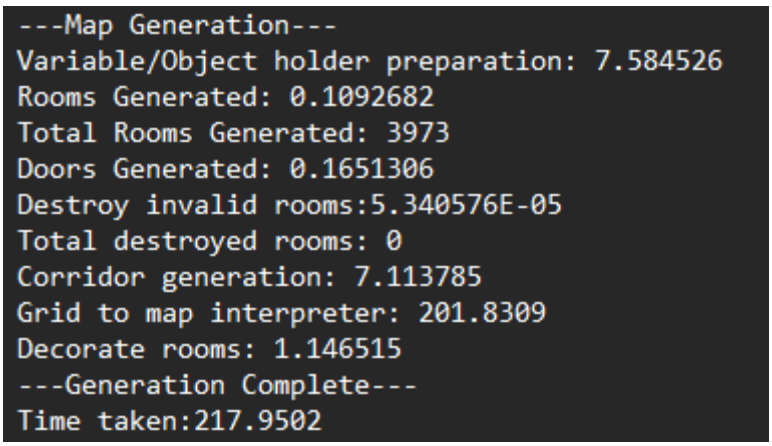


Figure 28: Recorded generation times of a much larger map.

The same can be seen in figure 28, whereby the most time-consuming stage is the interpreter. Its worth noting however that a map of this size is significantly bigger than a standard map. If maps of this size needed to be created, I would advise altering the generator to develop the map in sections, reducing the amount of ram required at a given time.

Unfortunately, most found papers didn't comment on generation time. However, a paper on generating maps using Minimum Spanning Trees [13] details map creation times of around 600ms, before the interpreter is run. Therefore, my solution is faster, however both would be suitable for usage at run time. However, their interpreter was significantly slower, taking "about 3.5 minutes" to interpret the map. However, they also commented that the interpreter was "unoptimized."

A notable issue with my design is regarding the corridor generation, which can lead to disconnected, and therefore inaccessible, sections of the map. This appeared to be a problem with multiple approaches to generation, such as Cellular Automata [6], which required connections to be made at a later stage. This is something that needs to be added to my project.

Although I didn't have the time to add multiple floors to my solution, I believe that this would be possible, making it one of the few algorithms capable of this. However, further investigation is required to ensure this is the case.

8.1 – COMMENTS ON LEGAL, SOCIAL, ETHICAL AND PROFESSIONAL ISSUES

I do not believe this project violates any Legal, Social, Ethical, or professional issue/legislations.

In terms of legal, my proposed solution isn't derivative off any other work, and I do not believe there are any legal violations with the project. However, if this was to be sold, the licensing of used assets would need to be checked to ensure that this would be allowed. If they are not allowed for use in a commercial product, a licensing agreement would first have to be negotiated with the creator of that asset, or a different asset would need to be used in place.

In terms of social issues, one may argue that generating content procedurally may have Economic implications, since there are no jobs required for hand designing a map. However, I do not believe this is to be the case. The amount, complexity, and maintainability of the required coding for a functioning game of this nature is far greater than if a map was hand drawn, potentially creating jobs. Furthermore, assets and proposed room designs are still needed for the generator to work; Tasks a map designer could be allocated instead.

In terms of Ethical and Professional issue, I do not believe the premise of the game will cause any. However, as with the already conducted survey, ethical approval and participant consent should be sought after before conducting any surveys regarding any project.

8.2 – FUTURE WORK

Regarding my proposed solution, I would suggest further research into the following:

- Investigating the options for a more sophisticated corridor generator, which may vastly improve the feel of the map.
- An investigation into a more efficient interpreter may prove to be useful, cutting down on generation speeds even further.
- An investigation into attempting to modify this solution to work on multiple floors, accessible without the use of a loading screen.
- Investigating a method to reliably decorate corridors with objects, such as light sources.
- Figuring out some method of adding AI's, or rewriting this program is a game engine that supports this should also be investigated further.

I would also encourage any research to be conducted in this field, to help aid its development, to eventually create solutions that are suitable for deployment.

REFERENCES

[1] R. van der Linden, R. Lopes and R. Bidarra, "Procedural Generation of Dungeons," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 78-89, March 2014, doi: 10.1109/TCIAIG.2013.2290371.

[2] B. M. F. Viana and S. R. dos Santos, "A Survey of Procedural Dungeon Generation," *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2019, pp. 29-38, doi: 10.1109/SBGames.2019.00015.

[3] John Von Neumann, Theory of Self-reproducing Automata. Urbana : University of Illinois Press, 1966.

[4] Y. P. A. Macedo and L. Chaimowicz, "Improving Procedural 2D Map Generation Based on Multi-Layered Cellular Automata and Hilbert Curves," *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2017, pp. 116-125, doi: 10.1109/SBGames.2017.00021.

[5] N. M. Husnul Habib Yahya, H. Fabroyir, D. Herumurti, I. Kuswardayan and S. Arifiani, "Dungeon's Room Generation Using Cellular Automata and Poisson Disk Sampling in Roguelike Game," *2021 13th International Conference on Information & Communication Technology and System (ICTS)*, 2021, pp. 29-34, doi: 10.1109/ICTS52701.2021.9608037.

[6] C. Adams and S. Louis, "Procedural maze level generation with evolutionary cellular automata," *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017, pp. 1-8, doi: 10.1109/SSCI.2017.8285213.

[7] T. Bäck and R. Breukelaar, "Using Genetic Algorithms to Evolve Behavior in Cellular Automata," *Lecture Notes in Computer Science*, pp. 1–10, 2005, doi: 10.1007/11560319_1.

[8] P. Davies, "3D cellular automata," *BPC Research Bulletin*, no. 5, Feb. 2011.

[9] “Genetic Algorithm,” uk.mathworks.com. https://uk.mathworks.com/discovery/genetic-algorithm.html

[10] A. S. Kholimi, A. Hamdani and L. Husniah, "Automatic Game World Generation for Platformer Games Using Genetic Algorithm," *2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2018, pp. 495-498, doi: 10.1109/EECSI.2018.8752741.

[11] E. K. Susanto, R. Fachruddin, M. I. Diputra, D. Herumurti and A. A. Yunanto, "Maze Generation Based on Difficulty using Genetic Algorithm with Gene Pool," *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*, 2020, pp. 554-559, doi: 10.1109/iSemantic50169.2020.9234216.

[12] J. Gutierrez and J. Schrum, "Generative Adversarial Network Rooms in Generative Graph Grammar Dungeons for The Legend of Zelda," *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020, pp. 1-8, doi: 10.1109/CEC48606.2020.9185631.

[13] B. von Rymon Lipinski, S. Seibt, J. Roth and D. Abé, "Level Graph – Incremental Procedural Generation of Indoor Levels using Minimum Spanning Trees," *2019 IEEE Conference on Games (CoG)*, 2019, pp. 1-7, doi: 10.1109/CIG.2019.8847956.

[14] “Minecraft franchise fact sheet April 2021 - news.xbox.com,” Minecraft Franchise Fact Sheet April 2021. [Online]. Available: https://news.xbox.com/en-us/wp-content/uploads/sites/2/2021/04/Minecraft-Franchise-Fact-Sheet_April-2021.pdf. [Accessed: 13-Apr-2023].

[15] W. Yin-Poole, “How many weapons are in borderlands 2?,” Eurogamer.net, 16-Jul-2012. [Online]. Available: https://www.eurogamer.net/how-many-weapons-are-in-borderlands-2. [Accessed: 17-Apr-2023].

APPENDIX

A – PROJECT LOGBOOK

Logbook

21/09/2022

Today was the first lecture regarding the Final Year Project. As of now, I am unsure as to what I intend to do for this project. My main goal currently is to narrow down on a field/idea for my project.

26/09/2022

I attended the lecture and Team's call regarding FYP. I am still uncertain as to what I want to do, but it will most likely be to do with 3D games. Since this is already a

module, I will require to go further than what is already taught. Some ideas of what I can add are realistic lighting/ray tracing, adding animations to actions, creating camera paths and procedural generation.

I am currently unsure as to which path to go down, as I don't want to scope this project under/over. I plan on looking further into potential options, as well as looking at the example dissertations and projects and posters provided.

28/09/2022

I have decided that I want to go down the route of 3D games and have scheduled an email with my top 5 group picks (hopefully I can rely on automation to send the email at the correct time).

03/10/2022

I received confirmation that I am a part of the "Mobile and 3D Games" group and attended the corresponding lecture/teams call. I am leaning toward procedural generation for my project, the nature of the game yet to be decided. Provided there is time, I may also attempt to add ray tracing. However, this is a secondary objective, and only if I have time/if it's worth adding. I plan on speaking to Dr Murray during the 3D lectures to refine my idea into a project.

10/10/2022

I attended this week's lecture and team call. There appears to be a lot of contradiction between provided information, that is becoming confusing and making me question my ideas. In the lectures, the lecturer is always stating that you must aim to answer some 'big question,' such as a comparison between two concepts/software, or something that will add to a field that requires more research. On the other hand, the meetings with my supervisor seem to suggest that making an app with a given purpose is also good. This contradiction is causing lots of confusion. Perhaps the differences in what each expects/thinks is a good idea is biased, based on what they think makes a good Final Year Project.

To alleviate this confusion, I plan to read all the example dissertations in depth, to see if those that made an app/website answered a 'big question,' or what that did instead. Following my findings from this, I will commit to what direction I want to take my idea, and begin to write my proposal. The deadline for the proposal is 23/10/2022.

13/10/2022

I spoke with my supervisor today about the concern I mentioned above. He doesn't think that it will be a problem. I will start writing up my proposal tomorrow.

14/10/2022 - 16/10/2022 (5 hours)

During this time period, I began and finished the first draft of my proposal. I sent the finished draft to my Supervisor to acquire feedback on what I have done upto now.

18/10/2022

I received feedback from my Supervisor today. This suggested feedback included an alteration to some of my objectives, providing an example of a game that uses procedural generation currently, and an adjustment to the layout to make it easier to read.

19/10/2022 (2 hours)

I attended a meeting of sorts with a few colleagues who are also taking this course to discuss the Project proposal. During this period, we discussed the general layout of what the proposal should look like, based on the provided information, and what should go where.

This meeting highlighted a couple alterations that I could make to my proposal to make full use of the maximum word count, to produce a better proposal. I plan on adding these changes tomorrow

20/10/2022 (1 hour)

I asked my Supervisor a couple questions I had regarding the proposal. I acted on previously given feedback and notes that I made to make my completed project proposal, ready for submission. From this point on, my focus will change to learning Unity and beginning research regarding my project.

26/10/2022 (0.5 hours)

I completed the risk assessment & ethical approval submission for my project.

28/10/2022

I've completed all of the provided tutorials from the 3D&VR module, providing me with an understanding of how Unity works. This is a good first step towards the prerequisite of my project, which is knowing how to use Unity. To consolidate my knowledge further, I plan on finding some more tutorials, to further my understanding.

29/10/2022 - 07/11/2022 (10 hours)

I have started a Unity tutorial to increase my understanding of Unity and its features, which will become very useful for later on in this project. The tutorial is learning how to use animations and creating an enemy AI for a 3rd person combat game. Most of what I am learning will be somewhat transferable to my project. I am approximately 60% through this tutorial.

Due to deadlines on assignment submissions, learning Unity is of a lower importance than completing the literature review and methodology, and therefore my main focus has shifted towards this.

10/11/2022

Today, I received my feedback and marks from my Project Proposal. I was hoping to achieve a higher score than what I did, but ultimately it doesn't make that much difference, considering the 10% weight. I wasn't happy with the criticism in the feedback provided, since none of the points were brought up when I submitted a draft for review. I also don't see how it would be possible to have acted on such feedback without going over the maximum word cap.

Regardless, I will continue to review literature for the next assignment.

10/11/2022 - 18/11/2022

I have completed extensive research into Procedural Generation, exploring the different algorithms which have been used to create dungeons, mazes and maps. I have made notes on most of these and now have a good idea of how I intend to lay out my Literature Review. I am concerned that I will be unable to discuss everything I want to however, given the word cap. I may have to disregard some of the algorithms unfortunately.

Going forward, I will begin to write my Literature Review and Methodology, acquiring more papers in regards to specific sections where I see fit. Some of my focus will also be shifted toward other assignments, requiring more time to write this than usual.

As for the proposed Gantt chart I submitted in my project proposal, I highly doubt I will be able to stick to it. It was created with many unknowns, such as what are my other assignments and how long will they actually take to complete. I may have to create a new Gantt chart, or go about this assignment with a less strict structure.

22/11/2022 - 02/12/2022

During this period, I have completed my Literature Review and Methodology. I intend to review it once more, looking for any required alterations, before submitting.

Unfortunately, I was unable to cover everything I wanted to in the Literature Review, as expected. The word count simply didn't allow for it. I attempted to balance out the amount of topics covered with how much detail/comparison was done.

Going forward, the next step would be to start creating the game, as well as splitting the tasks required into smaller, more concise tasks. However, I only intend to start this once all my other assignments are complete. Therefore, there will be no further logbook entries until I revisit this module. I am unable to accurately predict how long it will take me to complete the other assignments.

22/01/2023

I have begun to continue working on this project. The next steps will be to begin work on level generation, starting with the creation and placement of rooms. Today, I made some notes regarding the ideas I have for generating levels, since I did not find a suitable algorithm when researching. This included some basic notes on everything I would like to add to the game, so that I can begin development with them in mind.

The next step I need to take is deciding the size of the room. I intend for the height of rooms to be the same everywhere, with variations in the length and width of rooms. However, I have not picked an exact height yet, nor have I chosen a minimum and maximum size a room can be.

Therefore, choosing such parameters and creating a single room via code is my next step.

30/01/2023

Today, I successfully generated a room via code (using cubes of different dimensions to create the floor and walls. I can create square rooms of any size in any place using the code that I have created. I have also decided on the following parameters:

- The height of a room will be 4 units.
- The width of a wall will be 0.3 units.
- The depth of the floor will be 0.2 units.

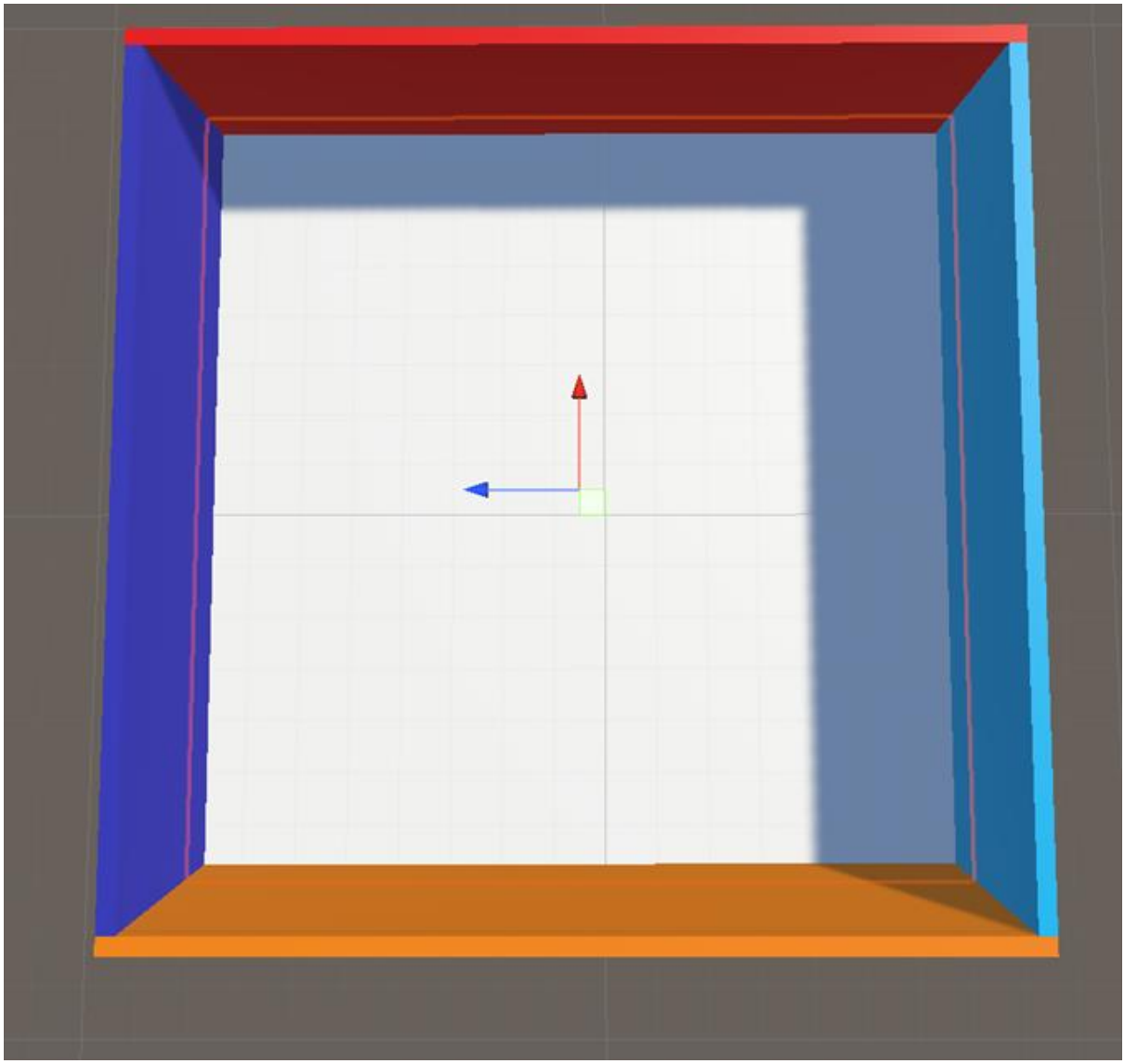
Where the units refer to the local scaling Unity uses. However, there are still a couple problems that need to be addressed before moving to the next stage of production.

Currently, there is some overlap with the walls and floor. This can cause visual issues, with unity trying to render two textures in the same place. This requires fixing, to ensure there is no overlap of the walls.

The rooms generated are currently square. I intend to also allow for rectangular rooms. The fix for this should be relatively simple.

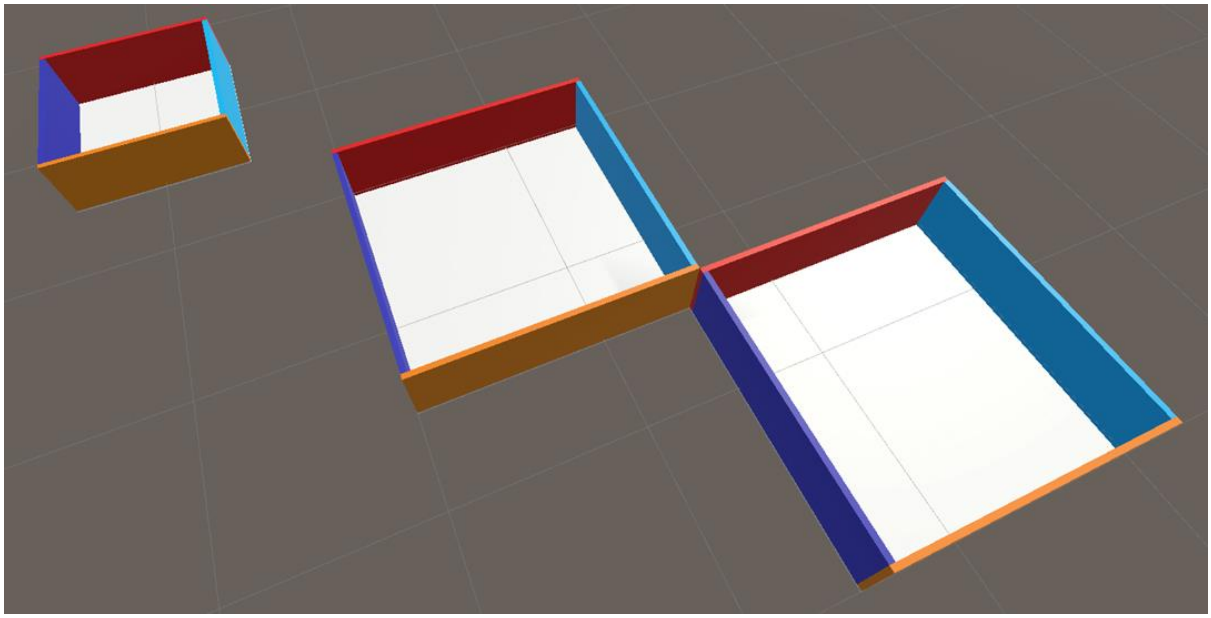
01/02/2023 - Update 1

I altered the length and positions of the walls, so that there is no overlap. I have also colour coded the walls using textures to be representative of the X and Z axis, as shown below:



These textures are temporary to ensure the walls are being created in the correct places. As the image shows, the Northern and Southern walls are longer than the Eastern and Western walls. I believe that this solution to prevent overlapping will be the most suitable for my solution.

As shown below, the created code can create multiple rooms of different sizes in different places:

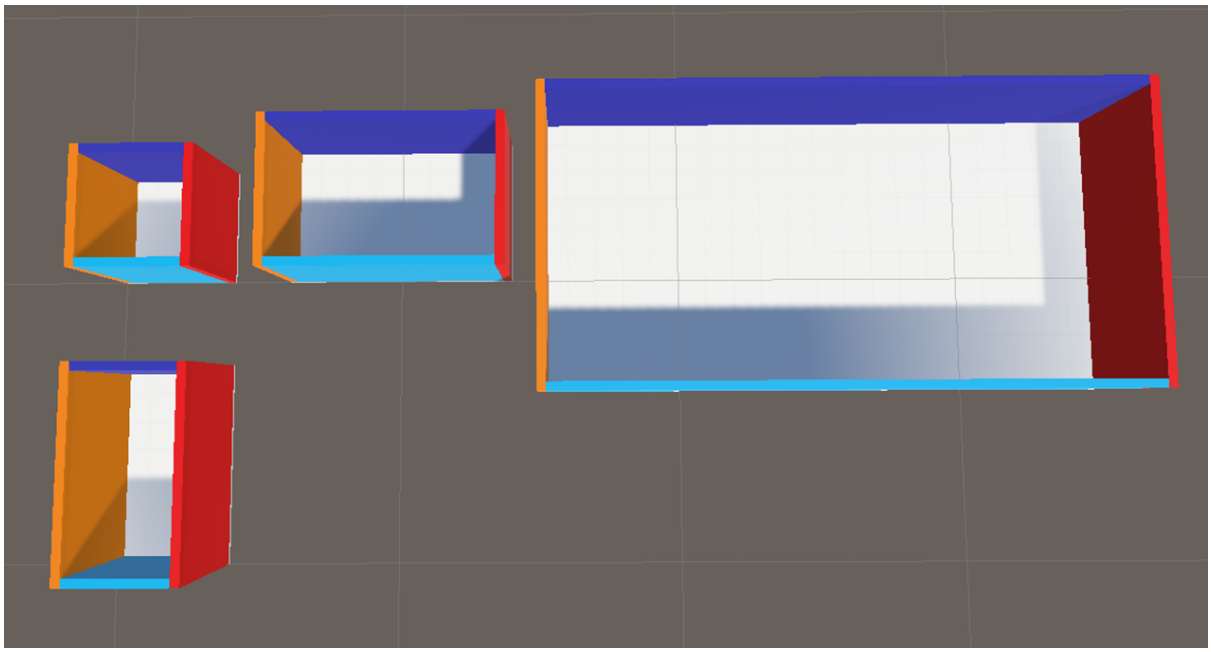


There are still a few issues at this stage:

- As mentioned before, the rooms generated are still square, and the x and z offset also have to be the same (where the offset is how far the building is from the (0,0,0)).
- Currently, the walls are on top of the floor. This could cause issues when adding multiple floors, since the floor texture will be present between the walls of each floor. This will be fixed at a later point, if it does become a problem.
- The current implementation does not support the addition of doorways. This will have to be addressed at some point.

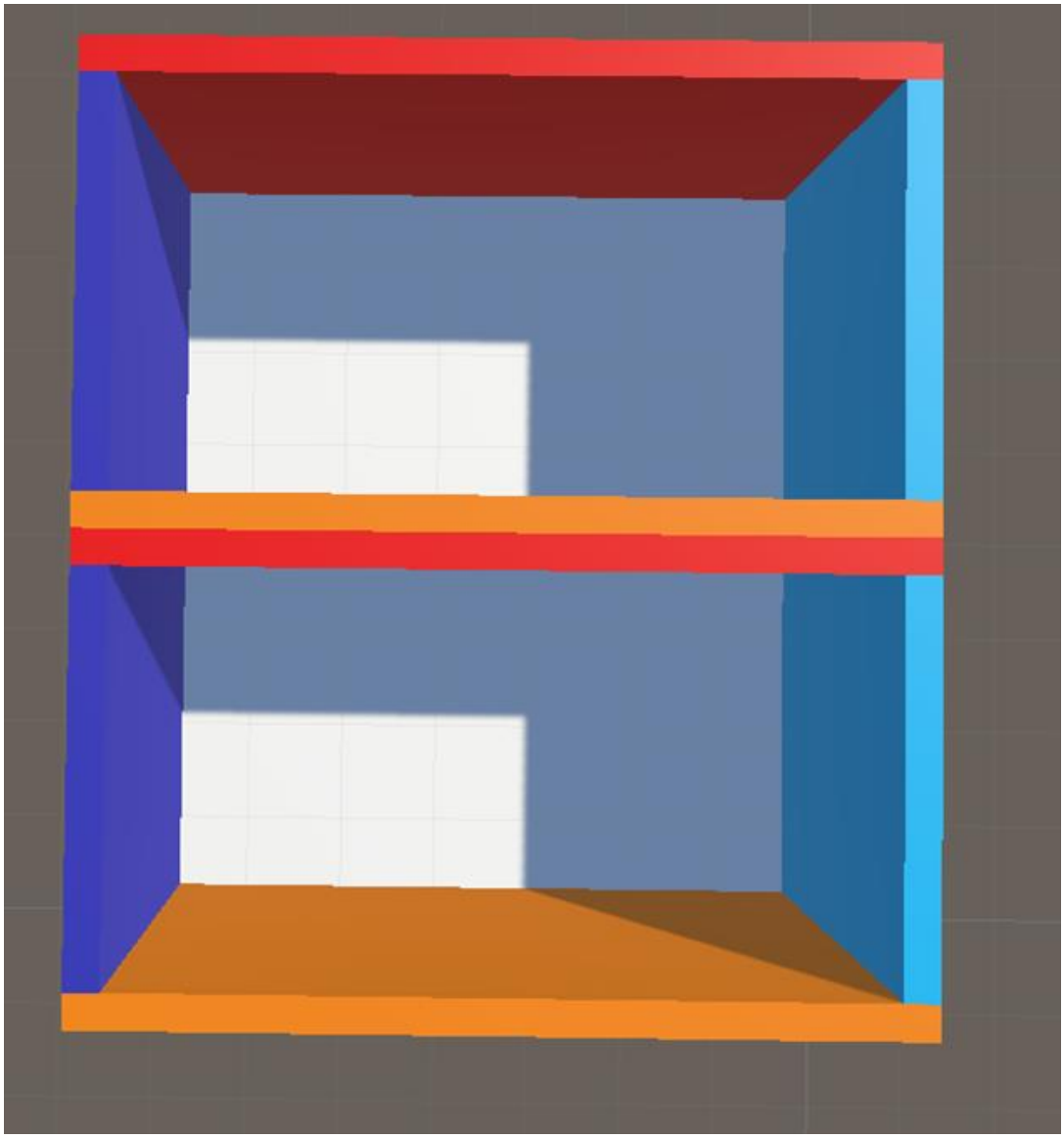
01/02/2023 - Update 2

Both the room shape and offset X and Z values are no longer tied together.



Although this was successful, more issues with this solution were found:

- Each room has no idea where any other room is. This will be an issue when trying to randomly place rooms of different sizes, as checking to see if the selected area is taken could be very costly. This will be more of a problem when generating paths to each room. I plan on dividing the allowed generation area into a grid, represented as a 2D array, which will represent the state of each tile (whether it contains a room, path, ect.)
- As of the current design, rooms that are next to each other have their own walls, as shown below:

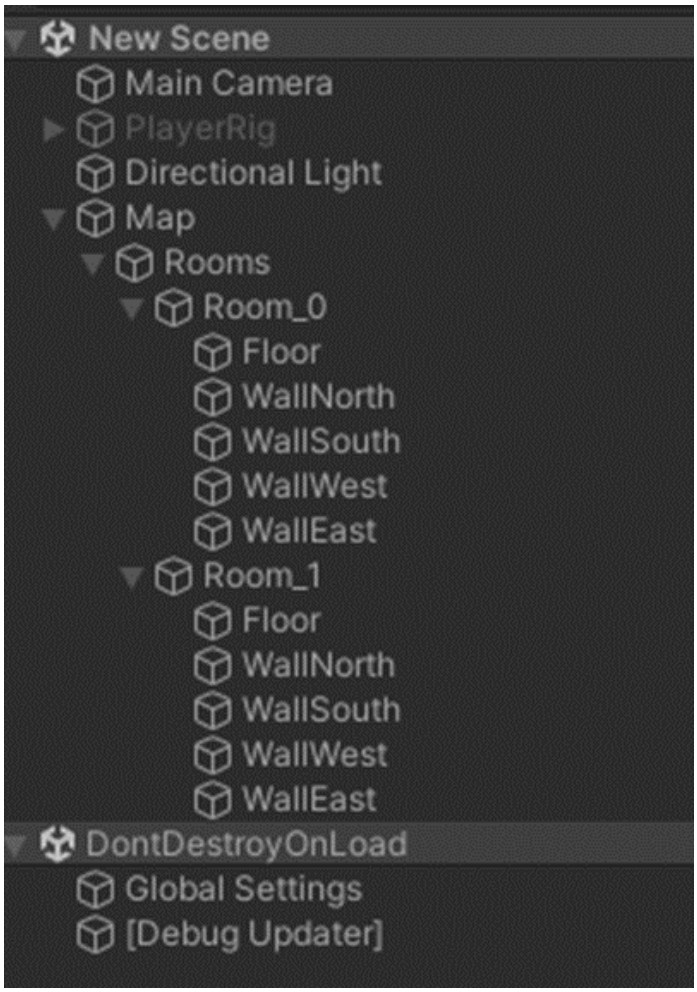


Ideally, the wall should be shared and centered between the two rooms, to prevent a wall that is too thick. Since I am altering the way generation is done (using a grid), there is no point trying to solve this issue now, but to bear in mind when changing to the new solution.

I have also decided that the width and depth should be between 5-15 units. I will implement this in such a way that I can alter these bounds if deemed necessary, like

I have done currently with room height, the thickness of the walls and the thickness of the floor.

I have also altered the creation of objects so that Hierarchy is tidier and therefore more readable by grouping related objects together by making them a child of an empty object:

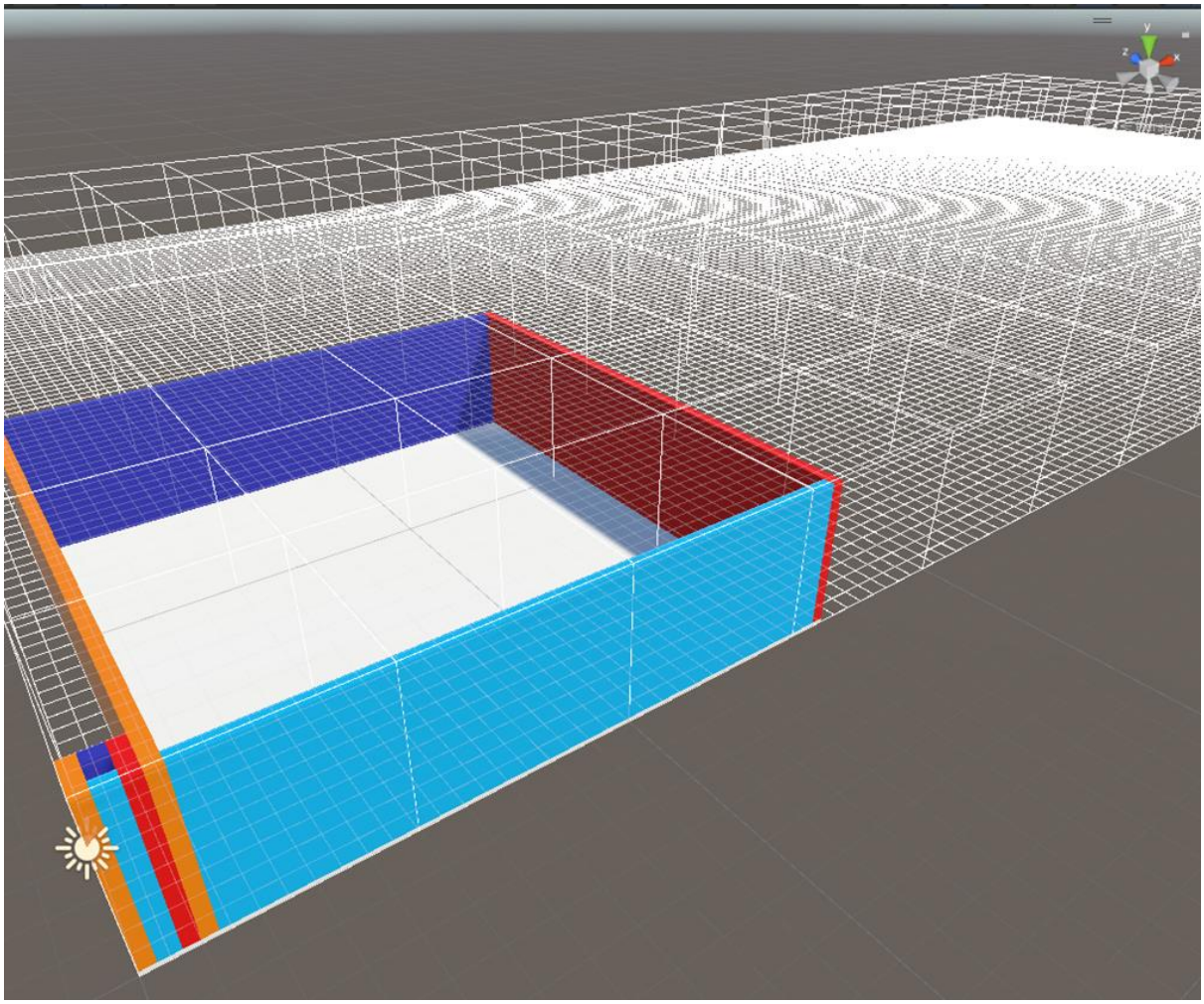


01/02/2023 - Update 3

Using Debug.DrawLine, I have created a visual representation of what the grid will look like. The size of each grid section and the grid size can be altered in the code. There are 3 options for visualizing this grid:

- Grid lines are hidden
- Only show the lines on one plane
- Show the gridlines to represent the height of the rooms.

The frequency of the grid can also be changed (only shows the lines for every X grid section). The below picture shows what setting the plane gridlines to show every line and setting the gridlines for height to show every 10 looks like.



As of current, this is only an outline of a non-functional grid. The next step is to make it functional.

04/02/2023

I attempted to create the visible representation of the grid using LineRenders, so the grid may be seen without using Gizmos. As expected, requiring so many objects to allow this to happen had a significant effect on performance. As such, I will not be using LineRenders to draw the grid. I will instead keep using Debug.DrawLine.

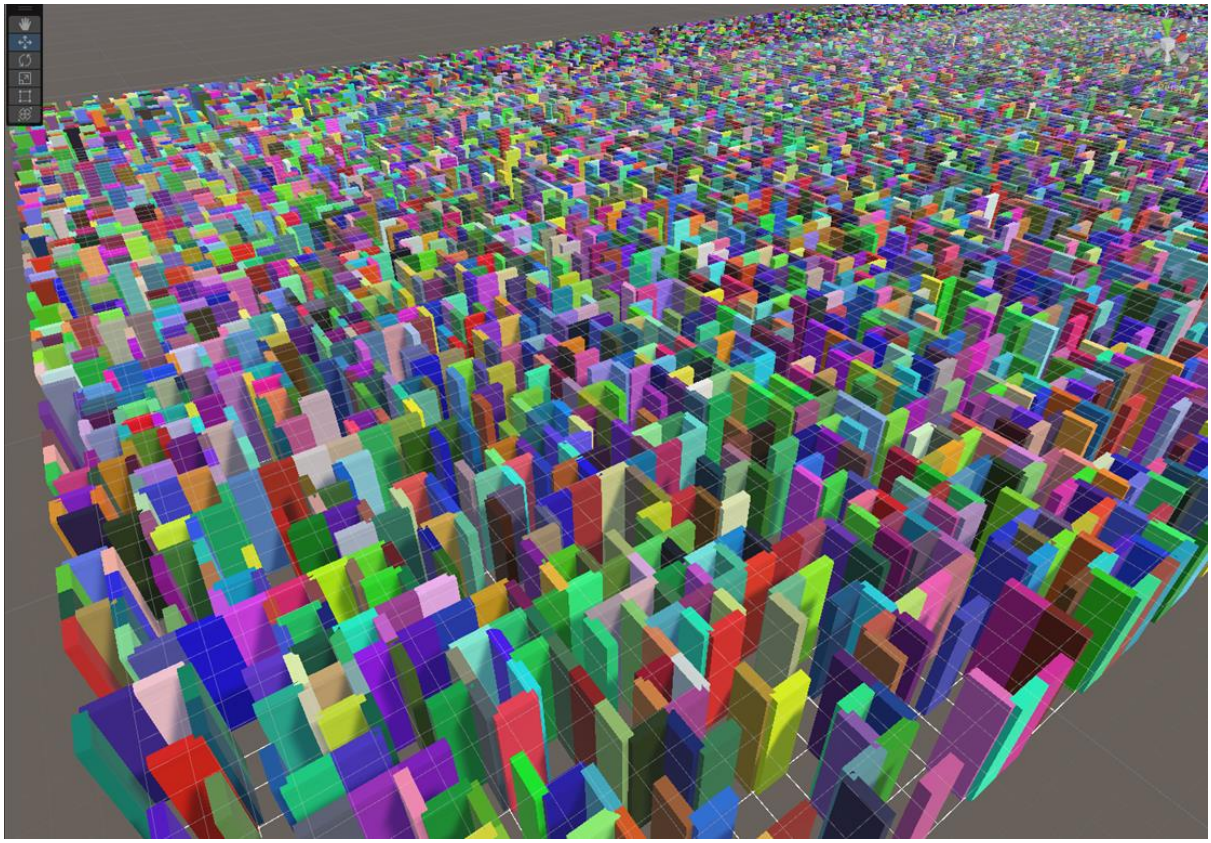
05/02/2023 - Update 1

After some consideration, I have decided that the best way to overcome the ‘shared wall’ issue is to have each grid segment state whether a wall is present on the East and South side of the tile. I will also include an initial line and column that cannot have a room, nor a corridor on, that will indicate the presence of the East or South wall where applicable.

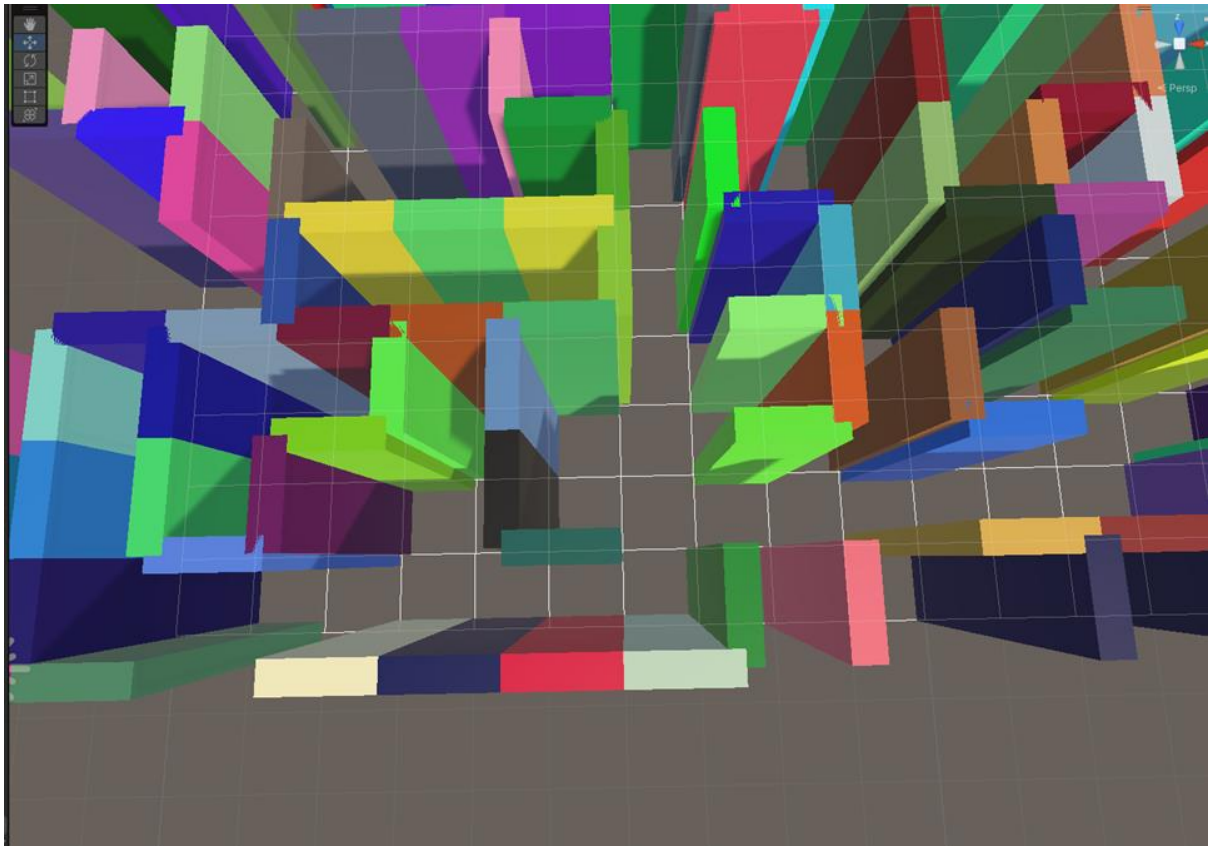
This will be later adapted to deal with other types of walls, such as doorways.

05/02/2023 - Update 2

Since the array interpreter will be used to draw all wall types, including ones for corridors, I opted to draw walls randomly whilst developing said interpreter. The image below shows the current state of development:



Positioning of the walls was achieved by modifying the code that was previously used to generate rooms. The main changes are the size of the wall to fit the size of one tile, and the position of the wall, to fit in the middle of the two connecting tiles. There are the following issues with the current design:

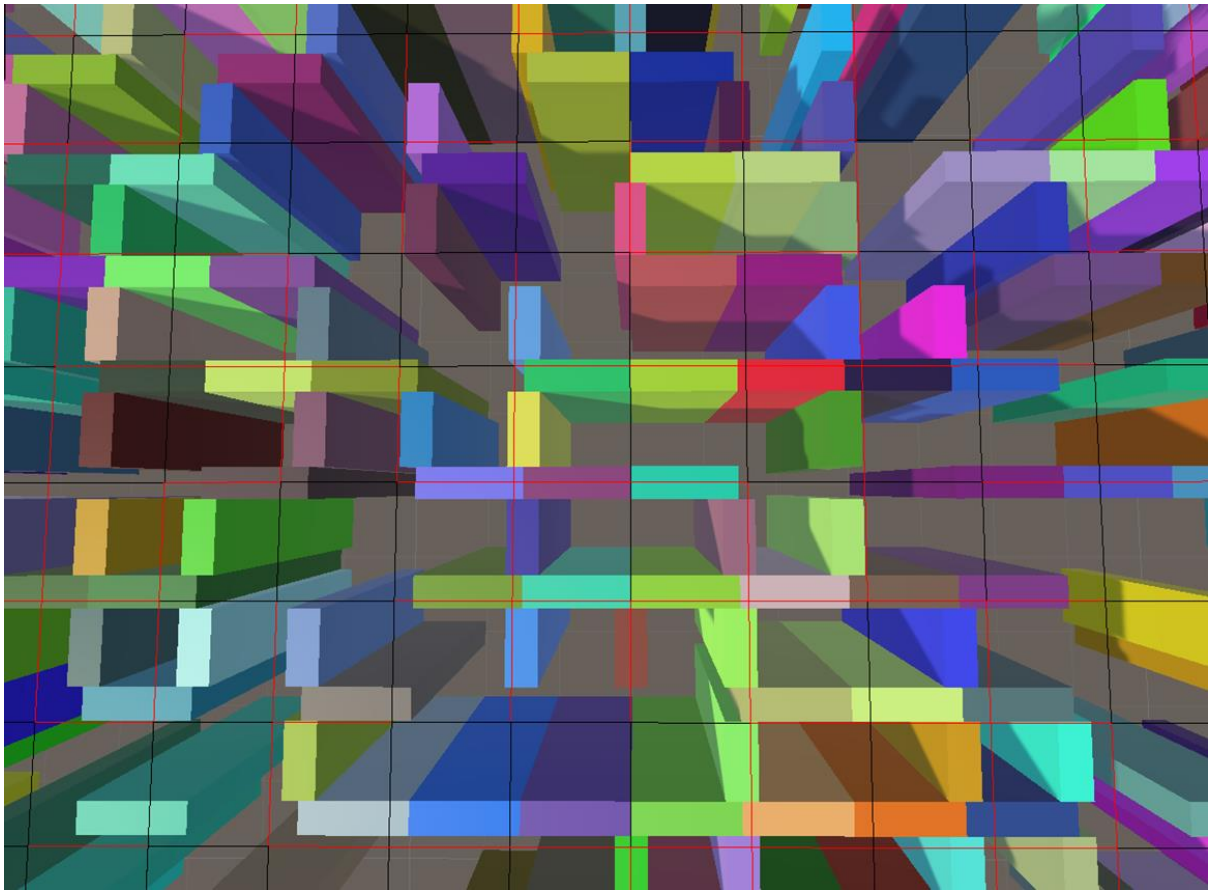


- Wall segments overlap and no longer fit square.
- Each object covers 1 tile, resulting in a significant amount of objects created (represented above using different, randomly assigned colours), Walls that are next to each other could become one wall instead. I believe combining wall segments will increase performance.

As such, these issues will be addressed next.

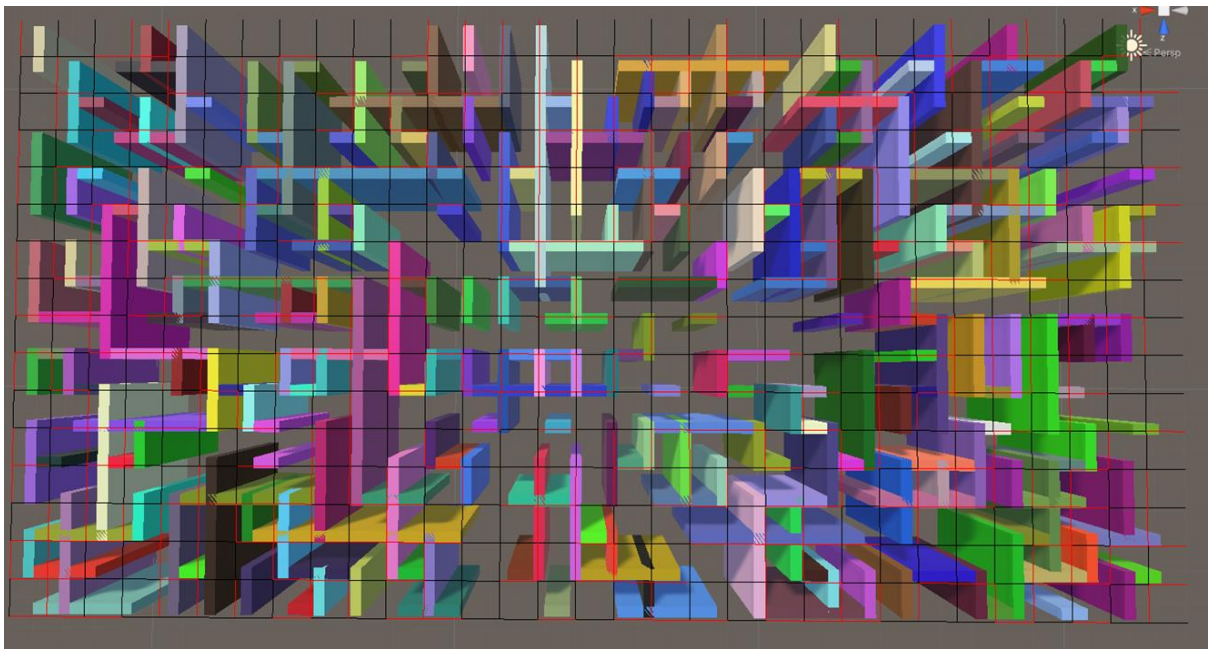
05/02/2023 - Update 3

I've created a grid using Debug.DrawLine that is colour coded to indicate whether a wall should be present (red) or not (black). This will be necessary to ensure the interpreter is working correctly.



06/02/2023 - Update 1

Grid tiles that have walls in the same direction as neighboring tiles now become 1 object, rather than one per tile, as the image below shows:



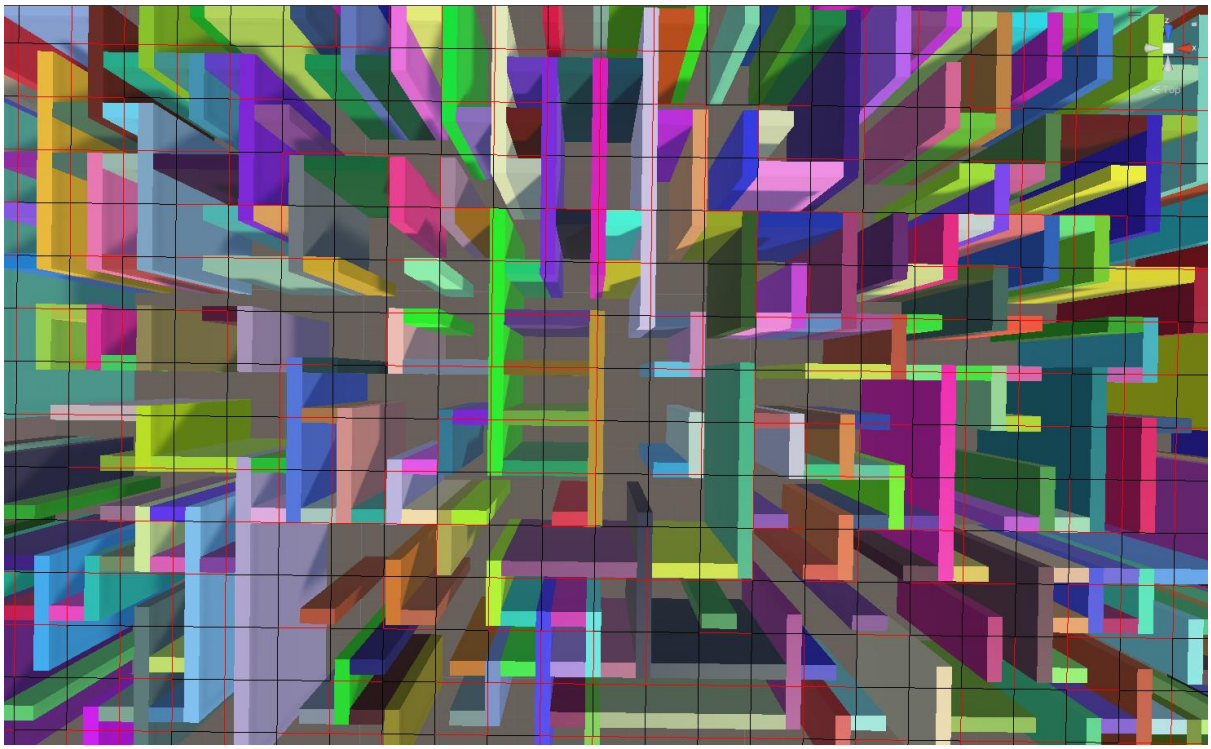
I have also made it so the end of an eastern wall does not overlap with a southern wall. This was done by making eastern walls smaller and southern walls larger

(equal to the width of the wall, which is currently 0.3f). However, overlapping now occurs when a section of the eastern wall (not the end points) draws through the southern walls.

To fix this, I will need to check for the presence of southern walls, or lack thereof, to see if it's valid to connect the Eastern walls, or whether they are required to be drawn separately.

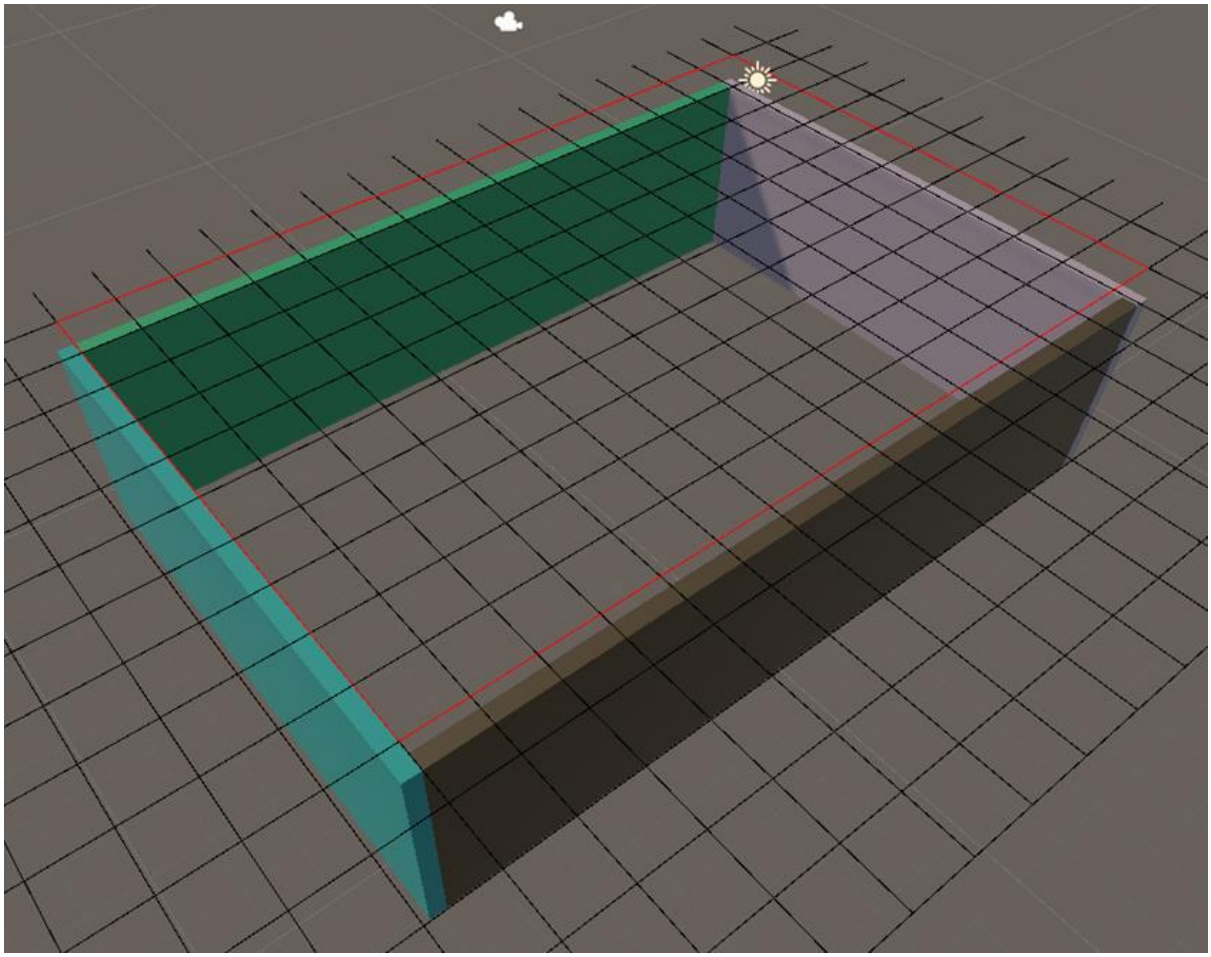
06/02/2023 - Update 2

All walls are now being drawn correctly. I can now move on to implementing a room creator. It's worth noting that the floor is not currently generated. This will be added after generating rooms.



08/02/2023

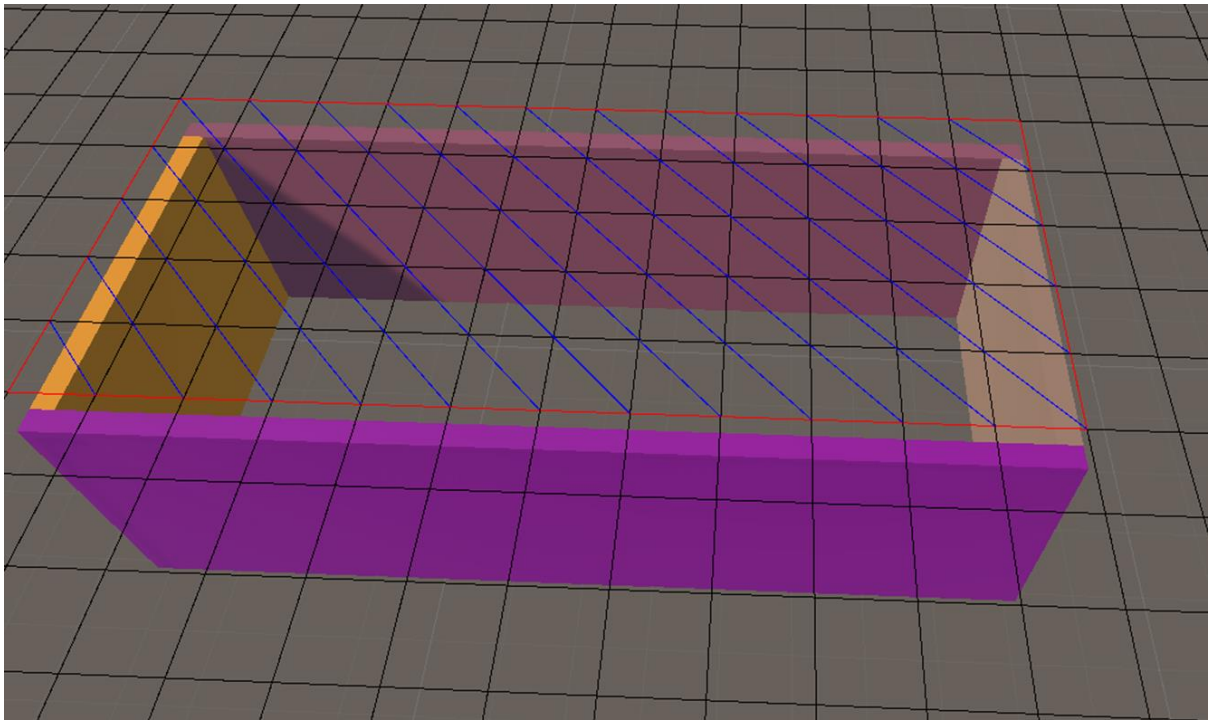
I have created the logic that allows for the drawing of a room within the set room size bounds, such as the one shown below:



11/02/2023 - Update 1

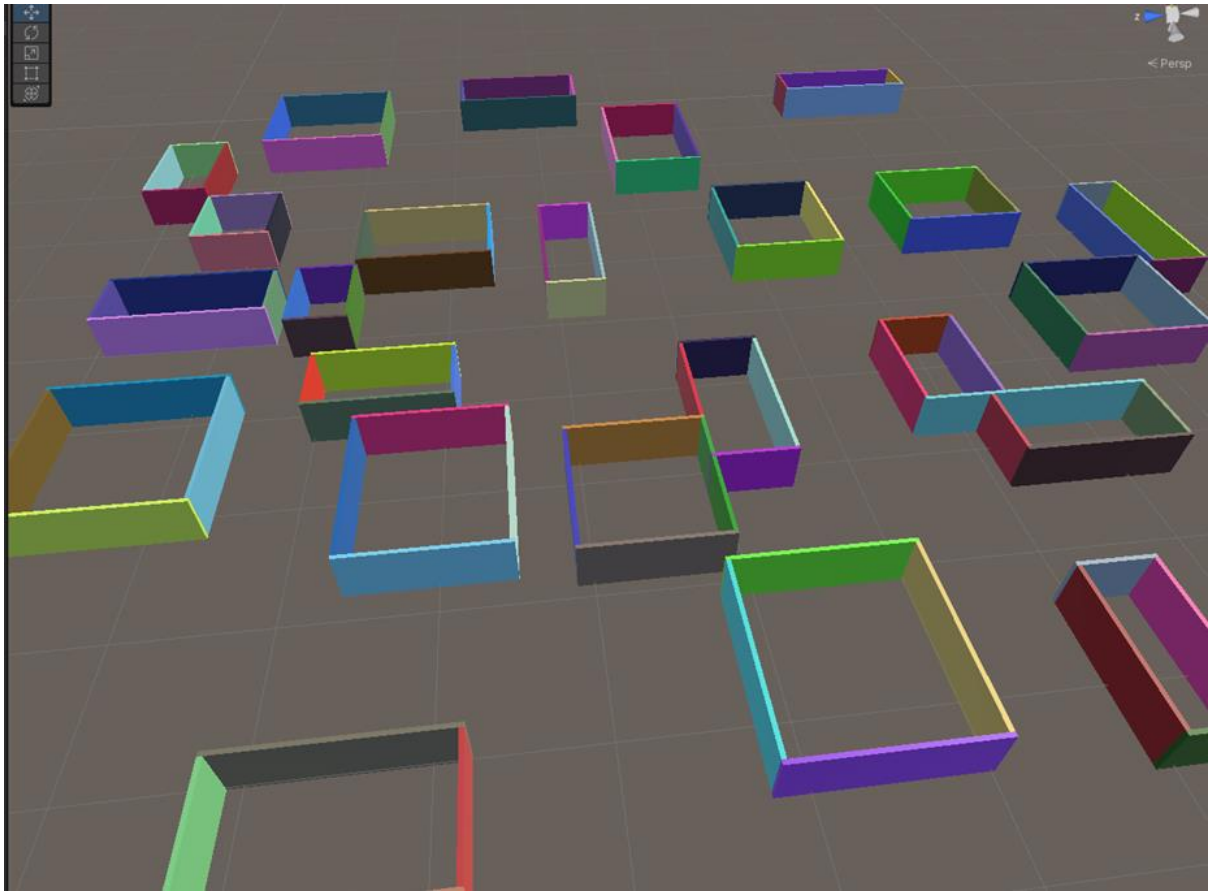
The generated room can now be placed randomly in a valid spot on the grid. All tiles that are contained in the area of the room are now allocated an integer value, indicating that they are a part of a room (rather than not yet set. To ensure this is working correctly, I have also modified the debug line grid, drawing a colour coded diagonal line if the tile is in use. This should prove to be useful as development proceeds.

The next step is to add multiple rooms that do not overlap.I have decided that the best approach would be to generate rooms until x% of the map is filled, rather than generate y amount of rooms. This way, the generation process will be able to adapt with greater ease to a change in the grid size.



11/02/2023 - Update 2

Multiple rooms are now generated, attempting to reach a specified % of the map to be filled. The next steps will require logic to allow for the addition of doors, and the generation of corridors.



,

13/02/2023

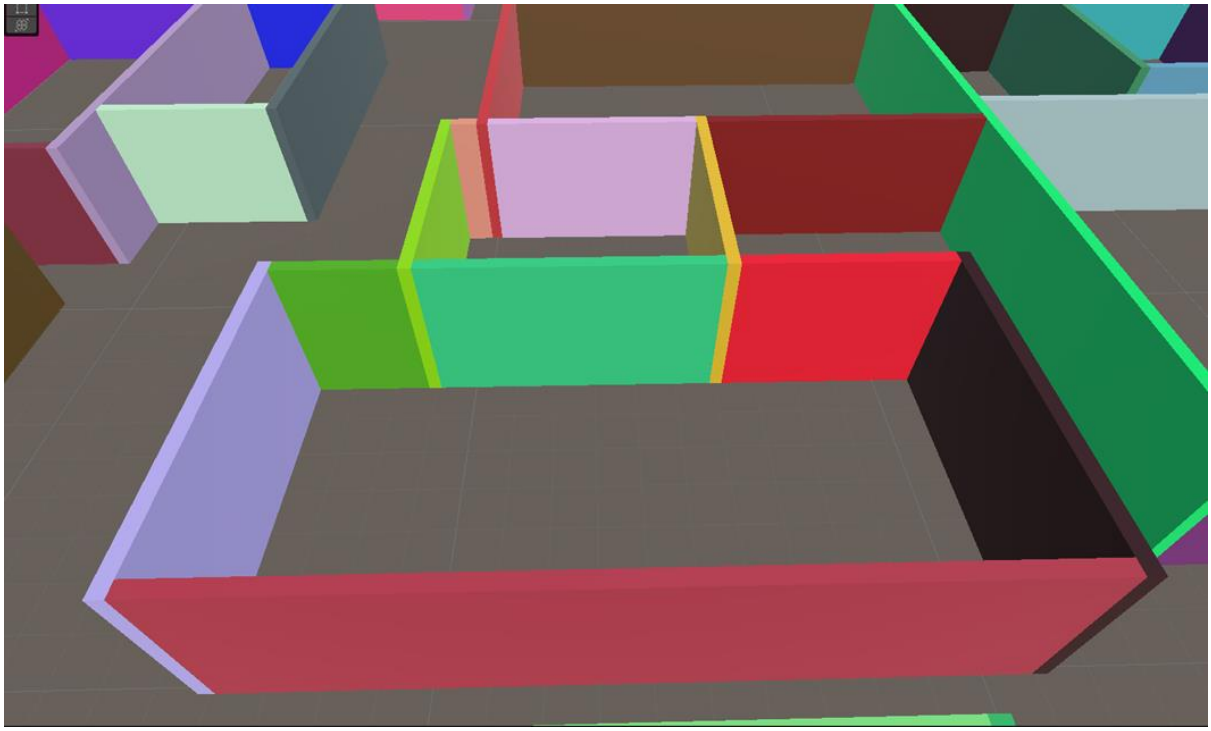
After some consideration of how to approach corridor generation, I have realized that my original plan of linking it to a minimum spanning tree/a shortest path algorithm may not be the best approach. The paths to and from rooms may become too predictable, not creating much variety. I need to rethink how I intend to approach this.

In the meantime, I will clean up my code where I can, and ensure generation is always successful (since the final build will allow for user changeable parameters, with some combinations potentially leading to a failed generation (for example, the map is too small to place 2 rooms, the minimum quantity required).

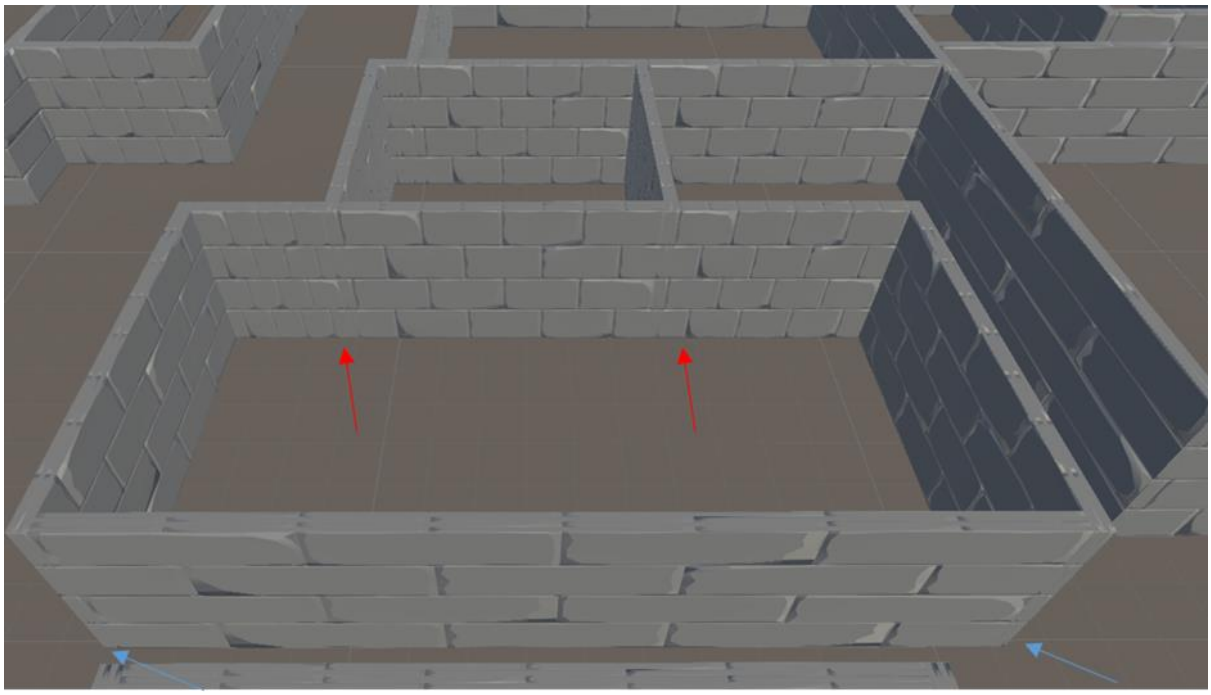
Progress on creating a staircase to allow for the creation of multiple floors can also be conducted.

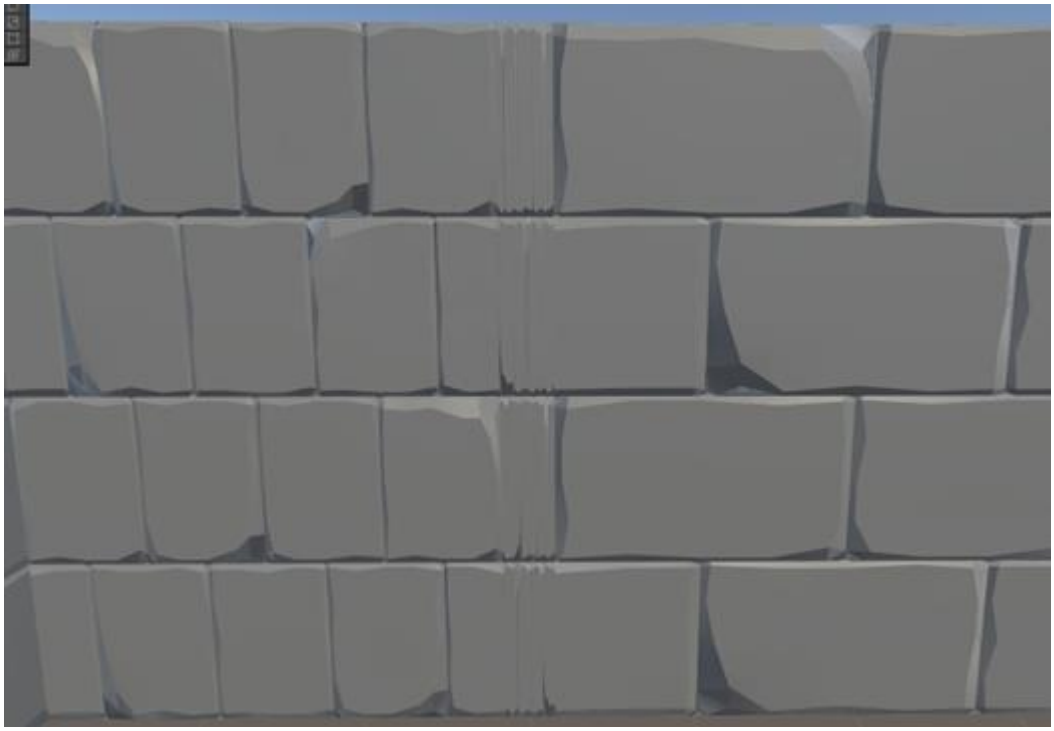
17/02/2023 - Update 1

Little work has been done over the last few days, due to being busy. Some minor changes have been made, which are not particularly noteworthy. A fairly substantial issue has been identified however, one which may take a significant amount of time to fix. The issue is in regard to how walls are generated, show in the example below:



With how the current wall generation is, some walls are broken up as shown above. When using solid colours, this doesn't pose a problem. However, when using Materials, a problem arises, as shown below:





As seen above, Unity attempts to show the entire texture on the surface, resulting in an inconsistent design for the wall. This will be an issue for both walls inside rooms, as shown using red arrows, and interior walls, shown with blue arrows. The interior wall issue will become a problem when generating corridors.

Unfortunately, I did not spot this earlier, due to my lack of experience with Unity/making games. I cannot use tilling to fix this issue for two reasons:

- This would change the tiling on all sides of the wall, breaking the appearance on other sides.
- All texture walls share the same texture currently, as this is more preformant that multiple instances of a texture, requiring a draw call each.

Another issue that can be seen is that the textures are stretched onto longer wall segments. The only fix I know of for this would be to use tilling, to resize and repeat the texture as wanted. This would again require a different instance of the material for every size of wall. Since everything that uses one material is drawn on one draw call, the following question arises: Is it more performant to have more objects, all sharing one material, or to have less objects, but more materials? Allowing for my interpreter to create larger sections of walls, to reduce the total created object, could be detrimental to the overall performance of the game. I will need to investigate this further.

There are 2 potential solutions I can think of, both of which could take a considerable amount of time to fix:

1. Alter how the walls are generated, ensuring walls surrounding a room are whole, not broken up. This would fix this issue outlined by the red arrows in the image above. However, for interior walls, outlined by the blue arrows in the image above, there will always be one wall that protrudes out, creating an

inconsistent design for the wall. This solution would not fix the issue. This method of generation would be more computationally expensive.
2. Create a custom mesh for the walls. I have no experience doing this, and have no clue if I can get it to work as I require. I also want to keep the option to alter the wall depth and height, so adjustments can be easily made if necessary. This may cause issues for the custom mesh too.

Further investigation into this is required.

17/02/2023 - Update 2

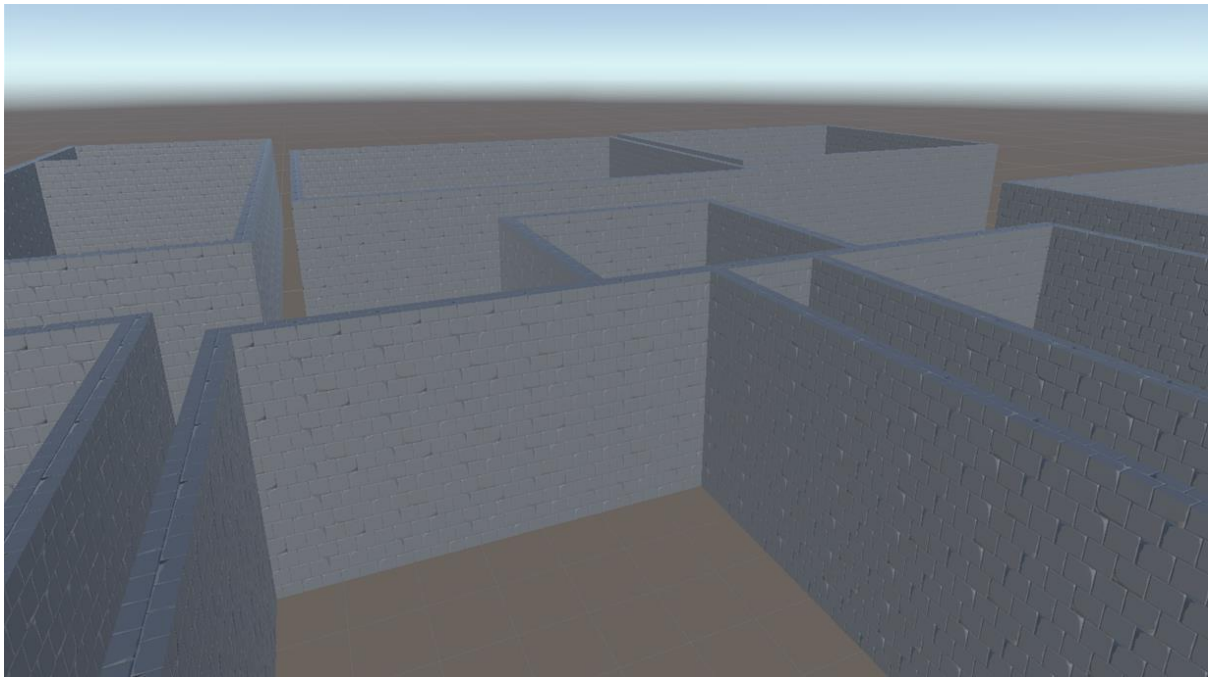
The solution appears to be to use custom meshes. After some investigation, it appears I can utilize ProBuilder to create the walls I require. Texture scaling is also handled by ProBuilder, fixing the issues with inner wall joins (represented using blue arrows in the image above). It is possible it will also fix the issues with broken up walls in the rooms too, however this is yet to be seen.

My current task is to figure out how to generate walls utilizing ProBuilder from a script, and modifying my current interpreter to utilize this feature instead.

17/02/2023 - Update 3

Wall generation now uses ProBuilder cubes. The issue with broken up walls in a room not having aligned textures is still present in some areas, but is significantly less noticeable. Areas where inner walls join now have the correct textures. ProBuilder meshes do require more time to create, increasing the total map generation time. However, I believe this is a worthwhile tradeoff for what it fixes.

I will leave this as is for now, since changing the interpreter logic in regard to how walls are generated may potentially cause issues for corridors. Therefore, I will revisit this at a later point instead.



22/02/2023

I've attempted to add doors to rooms over the last few days, but I have not been able to do so successfully. There are many checks that are required to ensure that the door is in a valid place, such as not being obstructed by other walls, not pathing into a different room, placed in a spot that allows for a corridor, not pathing out of bound, ect. The code for all of these validation checks was getting out of hand, and I noticed there were some redundant checks too (if one condition is met, another one is always the same value, or does not require checking).

I believe it will be best to start again with adding doorways and corridors, now I have a better idea of the checks that are required.

However, not everything I've done over the last few days is a complete loss. Below are a couple added features that will remain:

- If there is no valid place for a door in a room, it can now be destroyed via a function. This will prevent adding unnecessary objects, whilst also potentially freeing space up for corridor generation.
- The generation of walls using ProBuilder is significantly slower than using Primitive types. However, since this fixes the issues with material scaling, it is a worthwhile trade. Since I am using a grid to generate rooms, it is common for there to be multiple instances of walls that are the same size. Rather than calling the ProBuilder API to create another wall of the same size as a previous one, I now save the first wall of a specific size to a dictionary and create an instance of it every time another one is required. This method improves generation time by about 30%.

03/03/2023

Not much to note since the last time I added an entry. I am attempting to add doors in valid positions in rooms. A position is valid if a corridor can be generated next to said door. Originally, I had it so the door will always be centered. I got this mostly working, with some cases still adding doors where a corridor could not be placed. The checks required to ensure a door can be placed were getting out of hand, and the code was becoming very messy. This also had the disadvantage of never being able to generate an off centered door.

As such, I started over, trying to allow for the generation of doors in any valid spot for rooms. I intend to weight this toward a centered door once successfully developed, but still have the option of an off centered door for greater variation.

The validity check is slightly different for all 4 walls of a room, each with multiple checks required to ensure a door can be placed in each grid segment of the wall. This is mostly working for North and South walls, with some errors appearing during testing. I intend to get this working for all 4 walls by the end of today.

04/03/2023

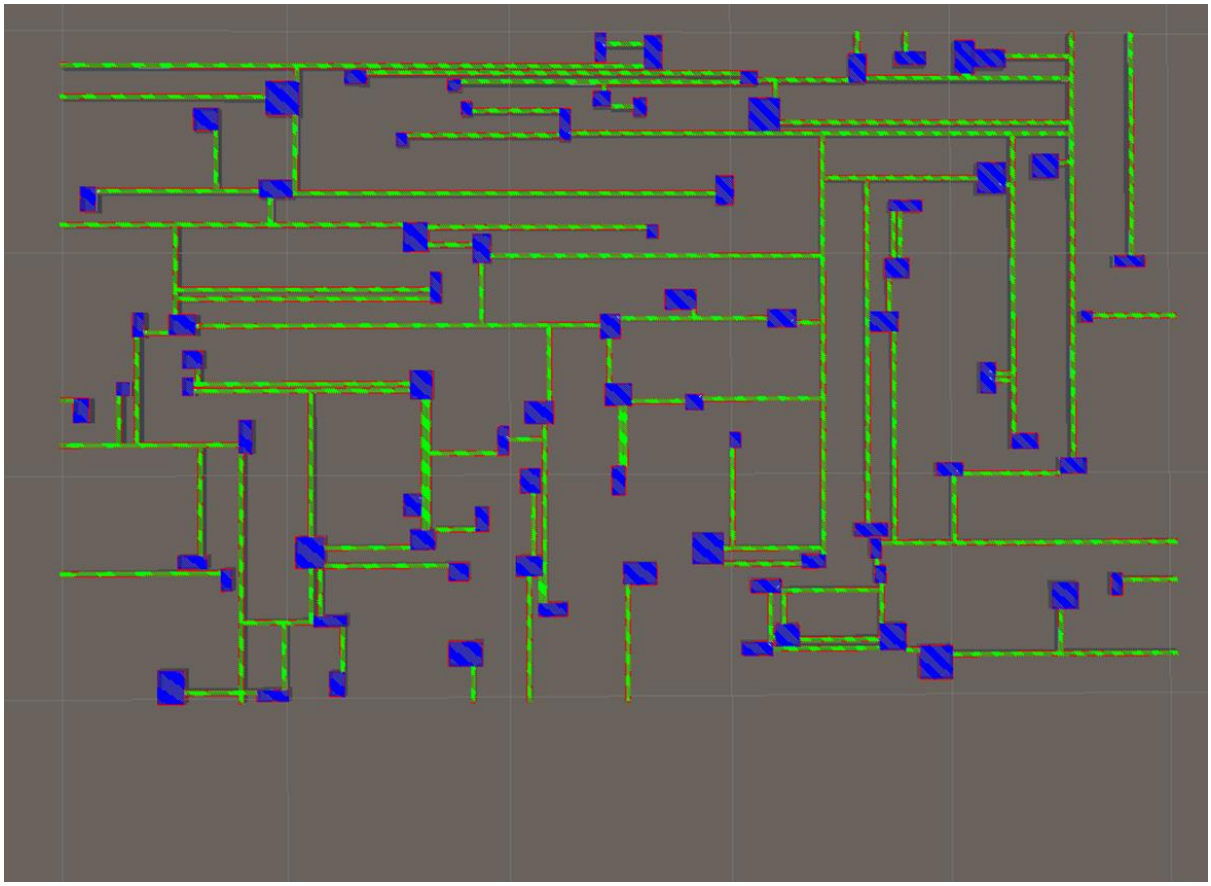
All issues outlined in the previous entry should now be fixed.

14/03/2023 - Update 1

I have now begun the development of corridor generation. When I was testing room/door generation, I usually opted for the map to be as densely packed as possible, since that is where errors were most likely to occur for the aforementioned steps.

However, since I am interested in seeing how the corridors will generate, which will only occur in the remaining empty spaces on the map, most maps will be loosely packed instead.

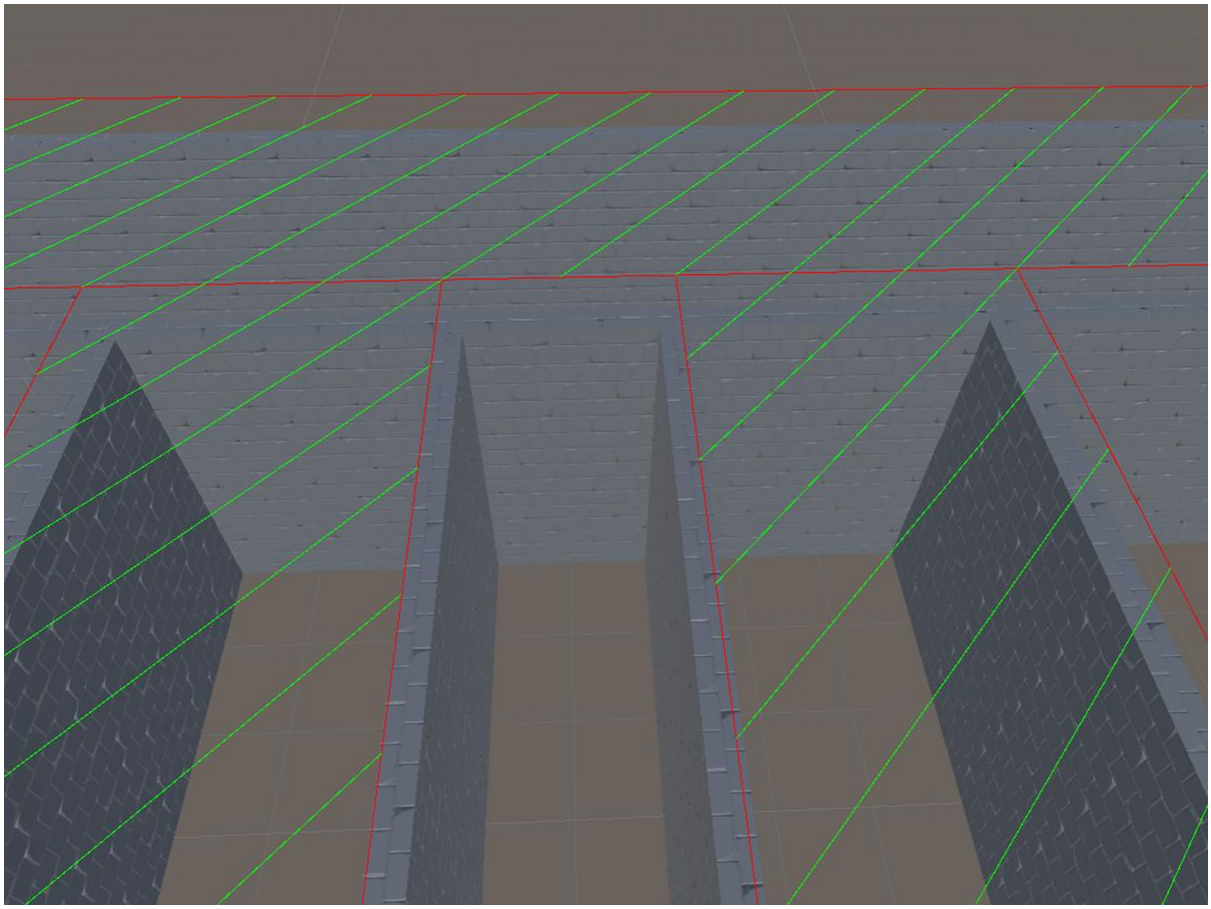
I have begun by allowing for corridors to path forward. From any given door, a corridor will path forward until it cannot path any more, such as there being an obstruction in the way, or reaching the end of the map. In the below image, I have represented the current generation with the colour coded debug lines I created earlier (blue represents rooms, green represents corridors):



Currently, the corridors have no way to connect to each other, and since only forward pathing is available, I have opted to prevent corridors from joining to other doors. As such, corridors stop x tiles away from any other door, x being equal to the width of a corridor. This is only temporary, to be improved when a more sophisticated corridor generation system is in place.

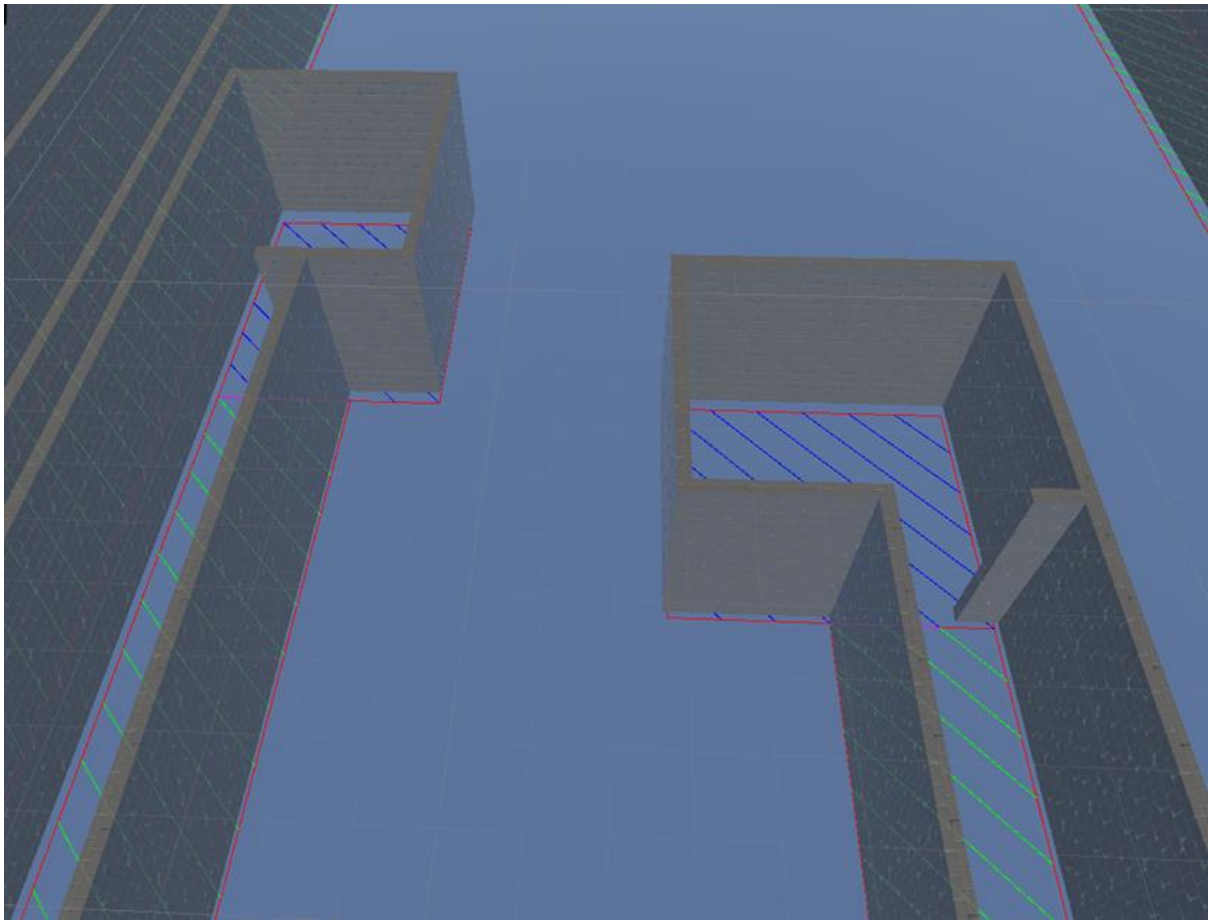
For each change, I check several thousand seeds for any ‘hard errors’ (ones which Unity throws), and a few seeds for any ‘soft errors’ (any generation issues that are incorrect/unintentional, but don’t cause an error to throw).

Currently, corridors cannot connect to other corridors; They path up to the existing corridor, then stop generating, since there is an obstruction in front, like the image below shows:



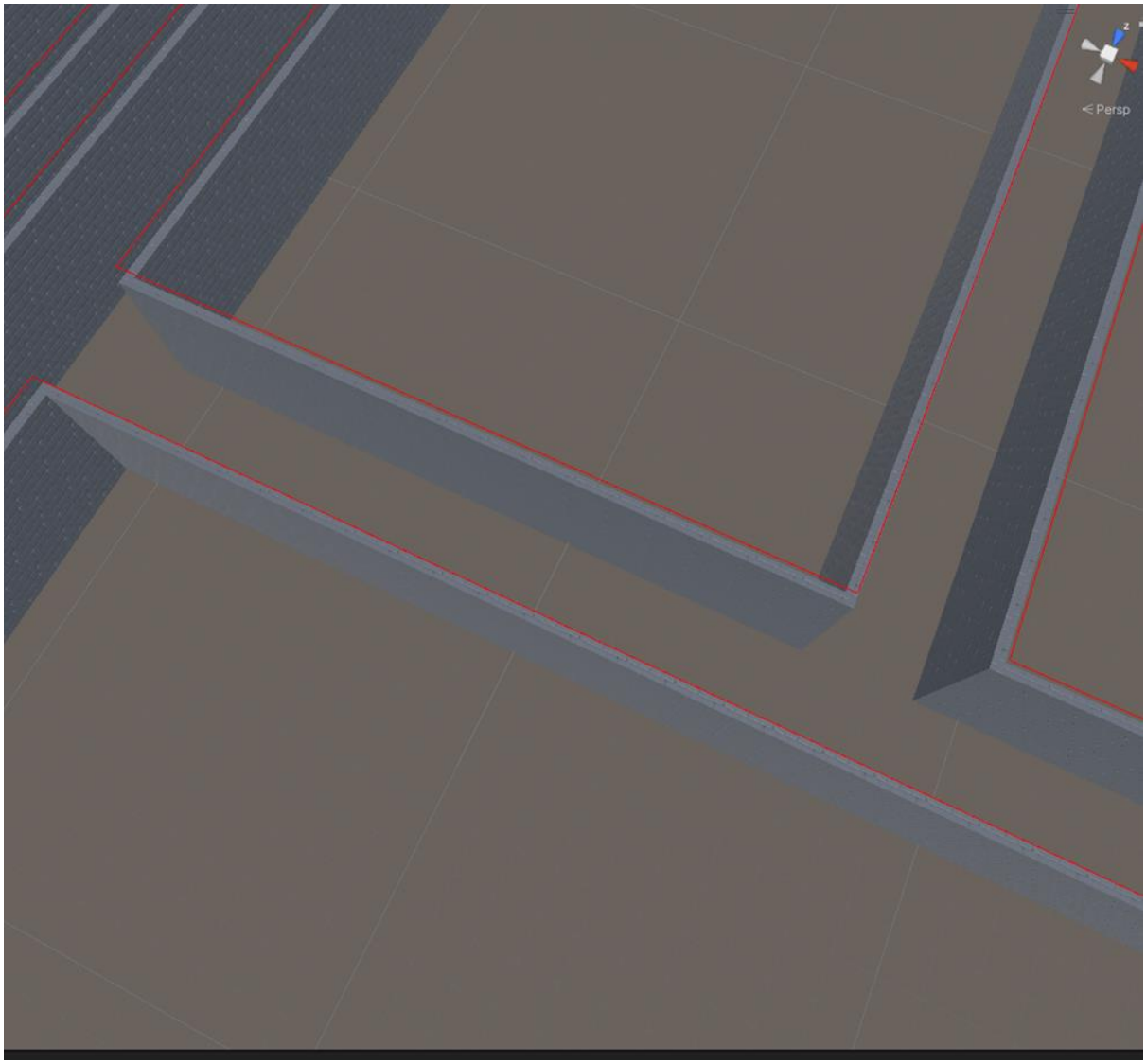
Connecting corridors where the meet square like this should be fairly simple, since no other corridors need to be generated. Thus, I plan this to be my next step.

Another change I will also implement is the corridor starting point when the corridor width is different from the doorway width, as the below image shows:



In such cases, there are potentially multiple valid options for the corridor to be placed (two for the ones shown above). Currently, the generator only opts for the furthest left option, which may be an invalid position for the corridor to go (only the other option is valid). Therefore, I need to add checks and allow for variation.

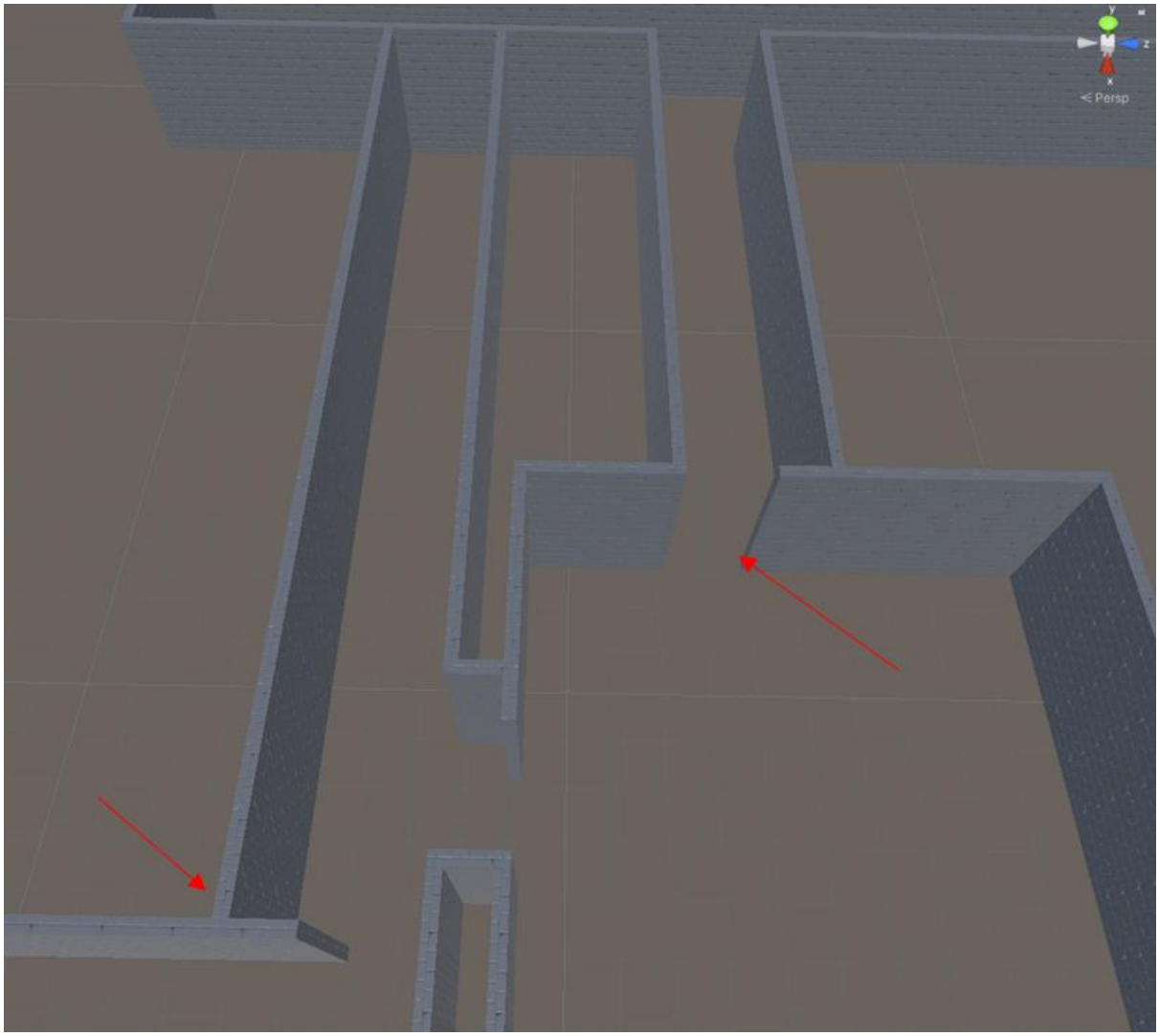
14/03/2023 - Update 2



Corridors that meet square with other corridors now join. This may later be altered to not occur all the time further into development.

16/03/2023

Corridors now begin to generate in any valid spots, at the beginning of a doorway (if the corridor width is greater than the door width, the corridor will no longer always generate in the same place). This took awhile to implement, since there are further checks required to ensure a successful corridor could be built from the chosen position.



The code should work with any sized door and corridor.

18/03/2023

I've spent the last couple days starting the groundwork for the continuation of corridor generation, to handle all possible cases (path forward, left, right, split left and up, split left and right, split right and up, three way split).

I also found a couple bugs, which I fixed:

- Doors could be generated next to each other, creating a double (or greater) width doorway. I have now changed it to ensure there is at least 1 tile between the doors.
- When a corridor runs perpendicularly to a door the program thinks generating an overlapping corridor is valid. If such an option is picked, the following function call (which creates the corridor) fails, since it cannot put a corridor in the chosen position. This is now fixed, but now causes other situations to invalid. I believe that this will be the better way to do it, therefore the change

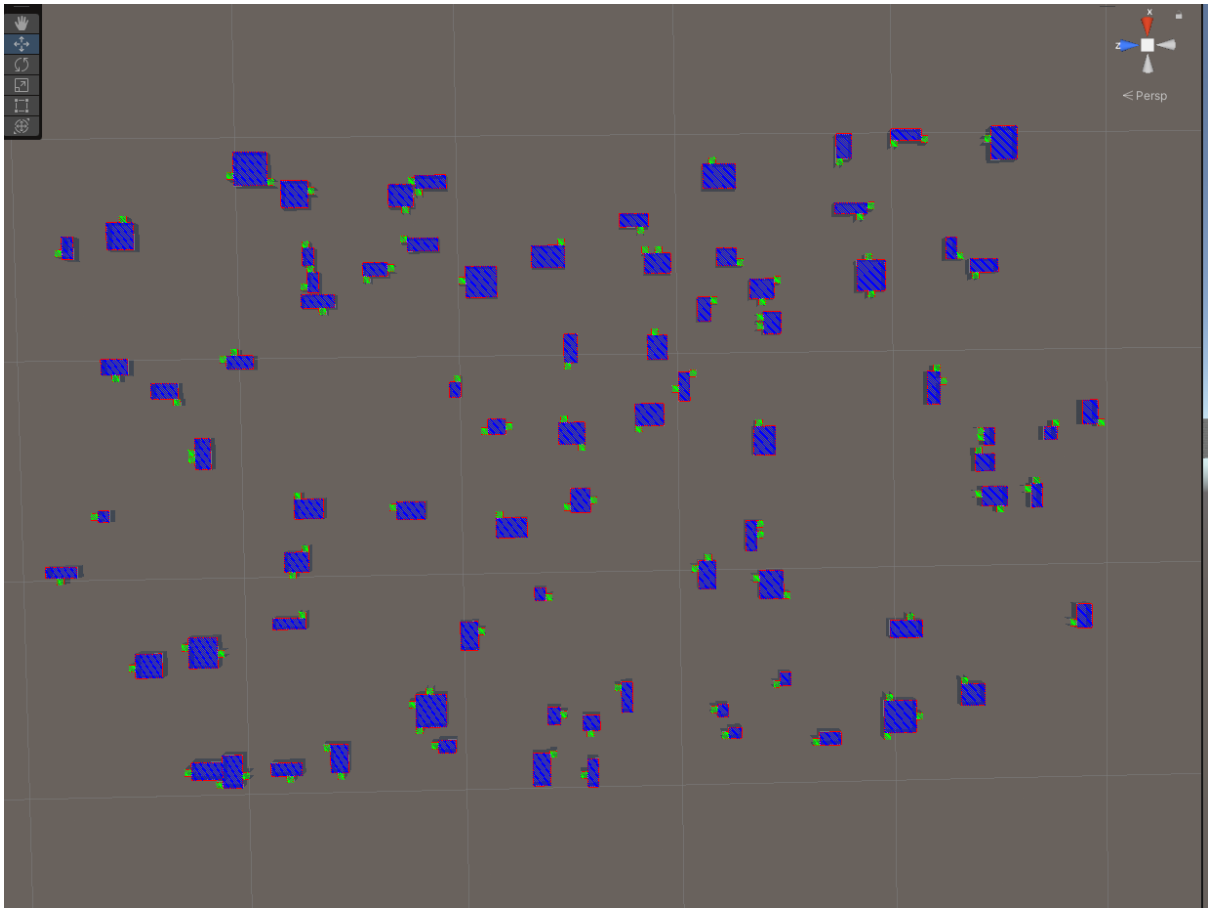
will remain. As for this new bug, a fully working solution cannot be easily made without corners being generated first.

20/03/2023

Over the last few days, I have been working on corridor generation. The code for generating straight corridors was removed, now that I have a better idea of how everything will work. Currently, the code finds a valid position to add a corridor to a door, if it is not otherwise blocked (which will need to be handled elsewhere) or is already connected. It then begins the initial part of the corridor by making corridorWidth x corridorWidth tiles to be type corridor in the required place, and generates walls where required.

The method that will begin the generation of the rest of the corridor is called, checks are made to see if the corridor can be extended forward, to the left and/or to the right, before calling a function to use this information to decide the next part of the corridor.

Currently, the program doesn't act upon this, since its not yet implemented. However, getting to this stage took 1100 lines of code, creating this:

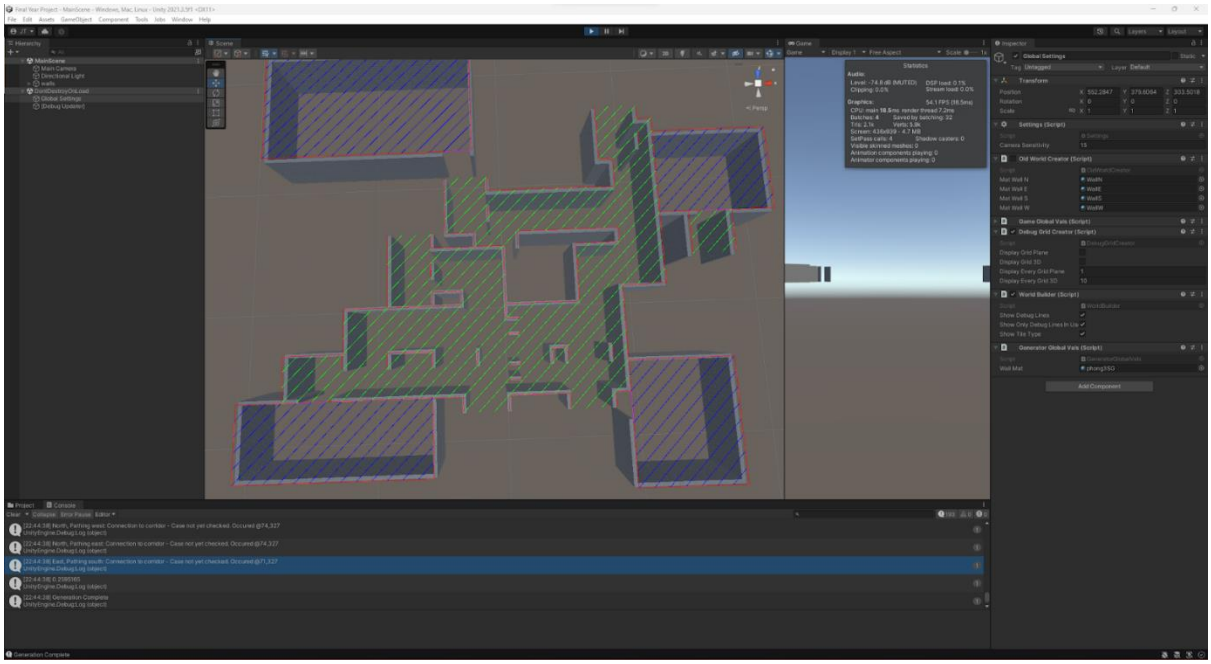


In the image above, the green represents corridors. This is all of the visual feedback I receive from all 1,100 lines of code, without manually altering grid tiles myself in the code. As such, making sure everything is working has been very challenging and has taken far longer than I expected.

Provided that everything is working as intended, I am now ready to begin adding other corridor segments and recursively call the required functions to generate corridors.

21/03/2023

I have coded the generation of corridors in 3 of the 4 possible directions. As one would expect, having 1,100+ lines of code without being able to fully test it leads to a few bugs, resulting in this:

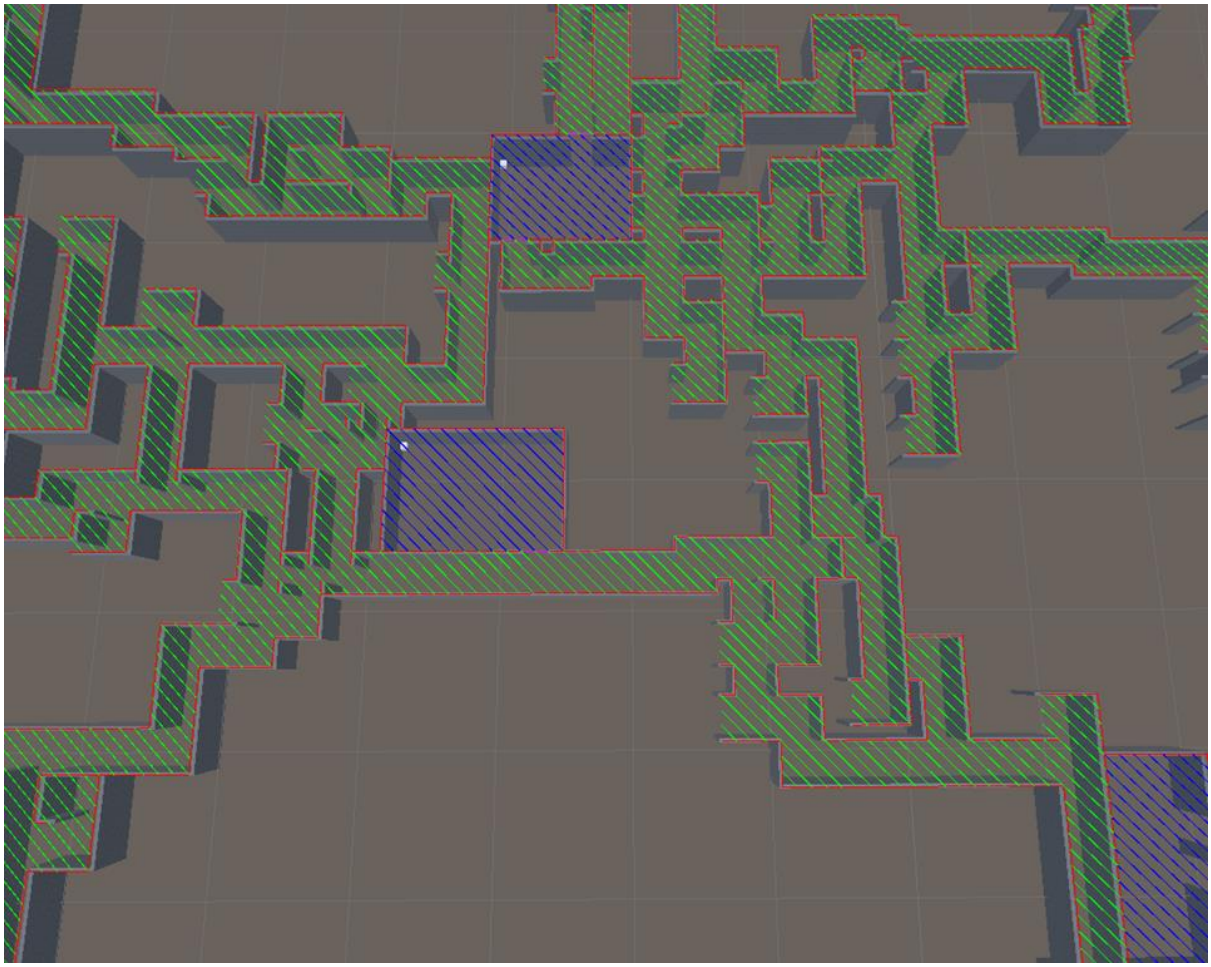


Since everything is procedurally generated, using recursive calls, figuring out what is going on here could prove to be quite the challenge. Somewhat annoyingly, I cannot affect the execution of the code to isolate any issues, since this would result in a completely different corridor being produced. This will make debugging significantly harder.

I have altered the code to write important information to a text document, since this will be the easiest way to figure out what's going on. Unfortunately, this produces 91,000 lines of text.

22/03/2023 - Update 1

I was able to find the cause of a lot of the generation issues: 1 line of code, being passed the wrong position to generate a corridor from.



As the above image shows, the corridors are generating significantly better now, albeit with a few issues that still require fixing.

22/03/2023 - Update 2

All other issues have been fixed - They were due to executing some code under all conditions, where they should've only executed under 1 condition instead.

23/03/2023

Corridor generation is implemented in all directions. However, a pretty major error has occurred. Generating corridors is done recursively, and since corridors can split into multiple directions during generation. Due to this, the generated corridors can be very large, since their only limitation is the available space to generate in. This can (commonly) cause a Stack Overflow error, as it hits Unities max recursion depth.

This is a massive issue, since most map generations now fail. I need to find a way to circumnavigate this problem, how exactly is unknown at this stage.

24/03/2023

A solution has been found for the issue: Rather than calling the required functions recursively, the data to make such a call is saved to an object that is added to a stack. This stack is pushed until empty, executing the functions in the same order as originally.

This approach does exactly what the recursion would have done, but since the previous function ends before the next one is called, rather than in a waiting state, there is minimal recursion depth.

26/03/2023

I've attempted to generate flooring in a way that it covers all of the corridors and rooms, whilst leaving empty tiles empty. The method I created mostly worked, with some cases not yet being accounted for. However, doing this creates a lot of objects, which Unity doesn't handle well, especially since each object would require a collider. The result was a massive performance loss. Since I don't know how else to approach this to reduce the amount of objects, I have opted for a different solution: Create 1 object that covers the entire area of the map for the floor and for the ceiling.

This solution is not compatible for multiple floors, but is significantly more performant. Therefore, given the time I have got to complete this project, this will be the solution I will implement.

28/03/2023

Fixed some issues where corridors were pathing to places where they shouldn't, cases where corridors were claimed as corridors when connecting to rooms, cases where walls were being generated in the wrong place and cases where walls were not being generated when they should.

Fixed an issue whereby walls sometimes had the wrong rotation, sometimes resulting in a complete failure to generate a map: Walls of the same size are cloned, rather than calling ProBuilder, which would be more expensive, and sometimes breaks. The first wall of a given size was used in the grid. If a rotation effect transform was applied to it, all subsequent clones of said wall would also have the transform applied. Instead, only clones of the original are now used.

01/04/2023

Fixed an issue where some cases for corridor generation whereby they incorrectly identified a wall as no longer needed and subsequently deleted it. This was mainly found if the wall belonged to a room.

Fixed an issue where corridor generation didn't work when the door width was equal to the corridor width.

Coded the remaining situations for corridor generation. All cases should now be handled.

03/04/2023

Added some decoration to rooms (chests and torches) to prove it is possible to do so.

I am in the process of adding a Menu and UI, to allow for operations in a game executable format, rather than through the Inspector.

05/04/2023

The required menus and UIs have now been implemented. Development of this project is now complete. This will be the last entry for this Logbook. The only step remaining is to complete the dissertation.

Project Title: Utilizing procedural generation to create a dungeon crawler game

Name: Jack Travis

Supervisors Name: ---

Project aim

The aim of this project is to utilize procedural generation to generate maps for a 3D dungeon crawler game, containing lootable objects and hostile AI. I also plan to include an option whereby the steps taken in map generation are visibly represented.

Minecraft is a popular example where procedural generation is unitized, being used to generate an effectively infinite map based off a random seed. This is one factor that has led to its success, due to the user experience being different each time. Likewise, No Man Sky utilises procedural generation to create entire galaxies, with explorable planted.

Most implementations of this concept are for world generation, with only a few games using it for a different feature. An example of this is the Borderlands series, that use procedural generation to randomly generate weapons.

Target audience and beneficiaries of this project

This project may assist individuals who intend to learn and implement procedural generation in their own system. Showing the process visually in steps may increase their understanding of how procedural generation works and help them figure out how to implement it in their own application.

The product of this project will be a working dungeon crawler game. Therefore, people who are interested in this genre of game may also be interested.

I will also benefit from this project, as I am undertaking something I have never done before. With an increased understanding of how procedural generation works, I will have the groundworks to be able to implement it in other applications, for different purposes and in different programming languages. This could prove to be a very viable skill, depending on what industry I work in, especially the games industry.

This project will also improve various soft skills, as I will be required to find, read, and reference literature, as well as managing my time effectively to complete my project in the given time span. I will also be required to write a professional dissertation. Improving these skills will be very beneficial.

Motivation for this project

The main reason as to why I chose this project is due to my interest in the games industry. I would like to improve my ability in this field, learning skills that would be transferable to a career. Successfully implementing procedural generation to a game can vastly increase its replicability, as your experience will differ each time. Therefore, I would like to learn and understand how it is done.

Depending on my implementation of this feature, I may also require using spanning trees and shortest path algorithms, both of which I have never done before. This will increase my coding ability further, with knowledge that can be transferable to a wide range of other applications, thus increasing employability.

Adding a "learn how it's done" feature may also help members of the community that are also interest in this concept. I would like to create a project that will inspire others to try it out for themselves, be it improving upon my design, or utilising it for different purposes.

I think that this project will be relatively complexed to complete, requiring a large time investment. Therefore, completing this project should also help my increase my organisational and time management skills, which are vital skills in life.

Required research

To complete this project, I will need to investigate methods of procedural generation and choose the option that best suits my requirement. Depending on the method chosen, I may also require investigating methods of path finding, to create corridors between rooms, as well as for the AI.

I will also research the ideal locations for information on the players 'HUD,' to fit with the standard/general consensus of other games.

Objectives

Below are the main objects I have for this project, for it to be successful:

1. To research into procedural generation to find the best approach for my project. Subsequent research into shortest path algorithms, for both procedural generation and the AI, may also be required. Further research in to the 'HUD' layout will also be required.
2. To implement a feature to allow the computer to make rooms, given parameters of minimum and maximum size.
3. To implement a feature to generate multiple rooms and calculate where corridors are going to be added.

4. To further build upon the generated map by adding routes to nowhere (dead ends) and distribute lootable items throughout the map, in rooms and said dead ends.
5. To create and add an AI for the game. They should be assigned the roles, such as guarding a room, or patrolling.
6. To implement a combat system. The AI is to attack the player when conditions are met. Once other conditions are met (when the play escapes), the AI is to return to their room/patrol route.
7. To create a 'Hub,' that the player will be sent to in between dungeon runs. They should be able to upgrade their tools and purchase items that will help them in the game. They should also be able to begin a new level.
8. To add an option to the game that allows the user to see how the levels are created, by dividing what happens into sections, and visually representing the procedure.
9. To create a 'Hud' display, showing relevant information to the user, such as their health. Hud elements are to be placed where research shows to be the optimal position.
10. Write my dissertation, including an evaluation of how successful my project was.

In addition, below are the secondary objectives I have for this project. I plan to add these if I am ahead of schedule:

1. Add multiple floors. This will require the rooms and corridor creation to work in a third dimension.
2. To add light sources to the dungeon, instead of global lighting to illuminate the surroundings, as well as creating darker areas.

Software and Technology used

I will use Unity to develop my game, therefore requiring Visual Studio for coding aspects of this project, and Unity itself. More specifically, I plan on using version 2021.3.5f1 of Unity to keep compatibility with university systems. However, this may change if I deem a different version to be more suitable. This project may also require the use of assets/packages, which I am unable to list at this time. Such assets will most likely be sourced from the Unity Asset Store, or Mixamo (in the case of animations).

For my logbook, I intend to use Google Docs, since it provides all the functionality I require, and has sharing capabilities, allowing my supervisors to access it. For other work, such as the dissertation and Gantt chart, I will opt to use software, which is a part of the Office 365 package, since I am more familiar with these applications. I also plan on using Trello to divide my work into small tasks, and to keep track of where I am up to.

I will primarily be working on my personal home desktop for this project, which has the following specifications:

- Operating System: Windows 11 Professional
- CPU: AMD Ryzen 7 3700X: 8 core, 16 threads, 3.6 GHz base clock speed
- Memory: 24 GB DDR4 RAM
- GPU: NVIDIA GeForce RTX 3080 Ti 24GB
- Storage: 5 drives total, consisting of a hard drive, SATA SSD's and a NVMe m.2, with storage space ranging from 256GB- 2Tb.

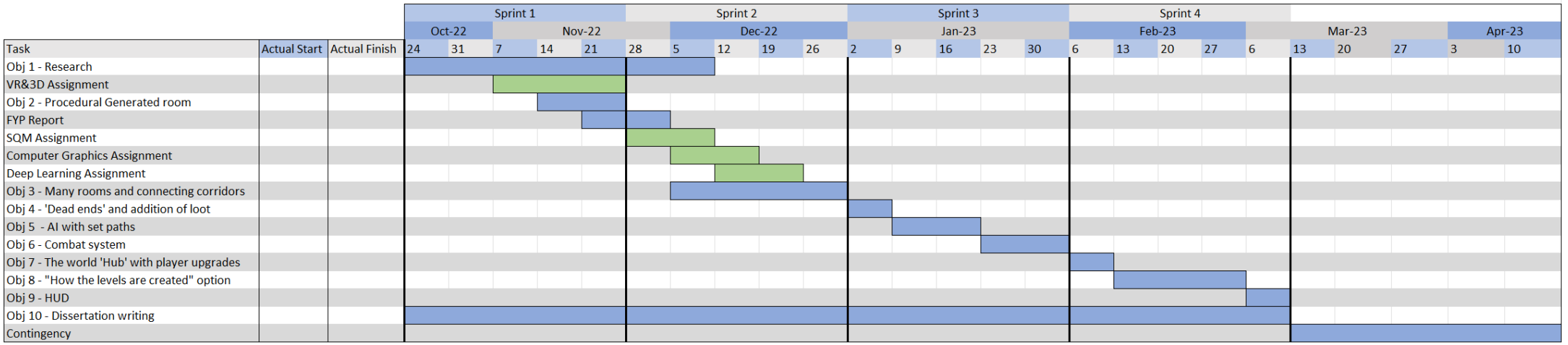
I do not expect my PC's specification to change throughout the course of this project and believe it will be sufficient for my requirements.

Development methodology

To achieve my goals, I believe the appropriate methodology to employ would be Agile. The iterative approach that it provides will be very useful for my project, so that I can implement my game feature by feature. Since I will be working on the project solely, and don't have the oversight of someone who has previously implemented procedural generation into an application, it would not be unreasonable to expect my idea of how to add it will somewhat flawed. By using an agile method, I can respond to necessary changes with relative ease, provided significant less issues than other methodologies, such as Waterfall.

I will use the Agile method, Scrum, for this project. This will allow me to keep track of where I am up to, by splitting the time I've got into Scrums and reviewing my progress at the end of each scrum. This will allow me to easily track if I am on schedule and make changes to my requirements if necessary.

Time Plan


Sprint 1
Sprint 2
Sprint 3
Sprint 4
Oct-22
Nov-22
Dec-22
Jan-23
Feb-23
Mar-23
Apr-23
Task
Actual Start
Actual Finish
24 31 7 14 21 28 5 12 19 26 2 9 16 23 30 6 13 20 27 6 13 20 27 3 10
Obj 1 - Research
VR&3D Assignment
Obj 2 - Procedural Generated room
FYP Report
SQM Assignment
Computer Graphics Assignment
Deep Learning Assignment
Obj 3 - Many rooms and connecting corridors
Obj 4 - 'Dead ends' and addition of loot
Obj 5 - AI with set paths
Obj 6 - Combat system
Obj 7 - The world 'Hub' with player upgrades
Obj 8 - "How the levels are created" option
Obj 9 - HUD
Obj 10 - Dissertation writing
Contingency

Appendix A – Logbook

Logbook

21/09/2022

Today was the first lecture regarding the Final Year Project. As of now, I am unsure as to what I intend to do for this project. My main goal currently is to narrow down on a field/idea for my project.

26/09/2022

I attended the lecture and Team's call regarding FYP. I am still uncertain as to what I want to do, but it will most likely be to do with 3D games. Since this is already a module, I will require to go further than what is already taught. Some ideas of what I can add are realistic lighting/ray tracing, adding animations to actions, creating camera paths and procedural generation.

I am currently unsure as to which path to go down, as I don't want to scope this project under/over. I plan on looking further into potential options, as well as looking at the example dissertations and projects and posters provided.

28/09/2022

I have decided that I want to go down the route of 3D games and have scheduled an email with my top 5 group picks (hopefully I can rely on automation to send the email at the correct time).

03/10/2022

I received confirmation that I am a part of the "Mobile and 3D Games" group and attended the corresponding lecture/teams call. I am leaning toward procedural generation for my project, the nature of the game yet to be decided. Provided there is time, I may also attempt to add ray tracing. However, this is a secondary objective, and only if I have time/if it's worth adding. I plan on speaking to Dr Murray during the 3D lectures to refine my idea into a project.

10/10/2022

I attended this week's lecture and team call. There appears to be a lot of contradiction between provided information, that is becoming confusing and making

me question my ideas. In the lectures, the lecturer is always stating that you must aim to answer some ‘big question,’ such as a comparison between two concepts/software, or something that will add to a field that requires more research. On the other hand, the meetings with my supervisor seem to suggest that making an app with a given purpose is also good. This contradiction is causing lots of confusion. Perhaps the differences in what each expects/thinks is a good idea is biased, based on what they think makes a good Final Year Project.

To alleviate this confusion, I plan to read all the example dissertations in depth, to see if those that made an app/website answered a ‘big question,’ or what that did instead. Following my findings from this, I will commit to what direction I want to take my idea and begin to write my proposal. The deadline for the proposal is 23/10/2022.

13/10/2022

I spoke with my supervisor today about the concern I mentioned above. He doesn’t think that it will be a problem. I will start writing up my proposal tomorrow.

14/10/2022 - 16/10/2022 (5 hours)

During this time period, I began and finished the first draft of my proposal. I sent the finished draft to my supervisor to acquire feedback on what I have done upto now.

18/10/2022

I received feedback from my supervisor today. This suggested feedback included an alteration to some of my objectives, providing an example of a game that uses procedural generation currently, and an adjustment to the layout to make it easier to read.

19/10/2022 (2 hours)

I attended a meeting of sorts with a few colleagues who are also taking this course to discuss the Project proposal. During this period, we discussed the general layout of what the proposal should look like, based on the provided information, and what should go where.

This meeting highlighted a couple alterations that I could make to my proposal to make full use of the maximum word count, to produce a better proposal. I plan on adding these changes tomorrow

20/10/2022 (1 hour)

I asked my supervisor a couple questions I had regarding the proposal. I acted on previously given feedback and notes that I made to make my completed project proposal, ready for submission. From this point on, my focus will change to learning Unity and beginning research regarding my project.