

Computer Science  
Advanced GCE H447  
Unit F453

Name: Karol Jeziorczak  
Candidate Number: 4162  
Centre Name: Rugby High School  
Centre Number: 31255

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Abstract</b>                                   | <b>4</b>  |
| <b>2</b> | <b>Introduction</b>                               | <b>4</b>  |
| <b>3</b> | <b>Analysis</b>                                   | <b>4</b>  |
| 3.1      | Computational Methods . . . . .                   | 4         |
| 3.2      | Researching The Problem . . . . .                 | 5         |
| 3.2.1    | Risk of Rain 2 . . . . .                          | 5         |
| 3.2.2    | Celeste . . . . .                                 | 8         |
| 3.2.3    | Rain World . . . . .                              | 10        |
| 3.3      | Stakeholders . . . . .                            | 12        |
| 3.4      | Interview with Stakeholder . . . . .              | 12        |
| 3.5      | Letter to Stakeholder . . . . .                   | 13        |
| 3.6      | Questionnaire . . . . .                           | 14        |
| 3.7      | Analysing Market . . . . .                        | 17        |
| 3.8      | System Requirements . . . . .                     | 18        |
| 3.9      | Limitations . . . . .                             | 18        |
| 3.10     | Essential Features . . . . .                      | 19        |
| 3.11     | Success Criteria . . . . .                        | 19        |
| <b>4</b> | <b>Designing the Solution</b>                     | <b>20</b> |
| 4.1      | Development Methodology . . . . .                 | 20        |
| 4.2      | Decomposing the problem . . . . .                 | 21        |
| 4.3      | Variable types . . . . .                          | 23        |
| 4.4      | Folder Set up . . . . .                           | 23        |
| 4.5      | Dungeon Generation . . . . .                      | 23        |
| 4.5.1    | Create Rooms . . . . .                            | 23        |
| 4.5.2    | Spread rooms . . . . .                            | 25        |
| 4.5.3    | Delete Rooms . . . . .                            | 29        |
| 4.5.4    | Delaunay Triangulation . . . . .                  | 29        |
| 4.5.5    | Remove random amount of edges . . . . .           | 33        |
| 4.5.6    | Generate spawn and boss room . . . . .            | 35        |
| 4.5.7    | Make sure all rooms are accessible . . . . .      | 38        |
| 4.5.8    | Turn edges into horizontal and vertical . . . . . | 45        |
| 4.5.9    | Generate corridors . . . . .                      | 47        |
| 4.5.10   | Turn into tile map . . . . .                      | 49        |
| 4.5.11   | Spawn player . . . . .                            | 52        |
| 4.6      | Player Movement . . . . .                         | 53        |
| 4.7      | Player Combat . . . . .                           | 53        |
| 4.8      | Enemy Design . . . . .                            | 53        |
| 4.9      | Boss Design . . . . .                             | 53        |
| 4.10     | UI . . . . .                                      | 53        |
| <b>5</b> | <b>Developing a solution</b>                      | <b>53</b> |
| 5.1      | Dungeon Generation . . . . .                      | 53        |
| 5.1.1    | Create rooms . . . . .                            | 54        |
| 5.1.2    | Spread rooms . . . . .                            | 57        |
| 5.1.3    | Delete rooms . . . . .                            | 62        |
| 5.1.4    | Delaunay triangulation . . . . .                  | 63        |
| 5.1.5    | Remove random amount of edges . . . . .           | 75        |
| 5.1.6    | Make sure all rooms are accessible . . . . .      | 77        |
| 5.1.7    | Add spawn and boss room . . . . .                 | 81        |
| 5.1.8    | Turn edges into horizontal and vertical . . . . . | 83        |
| 5.1.9    | Turn into tile map . . . . .                      | 85        |
| 5.1.10   | Spawn player . . . . .                            | 91        |

## List of Figures

|    |  |    |
|----|--|----|
| 1  | [Ror] Risk of Rain 2 Screen shot . . . . .                               | 5  |
| 2  | Reviews for Risk of Rain from [Ror] . . . . .                            | 7  |
| 3  | [Cel] Celeste Screen shot . . . . .                                      | 8  |
| 4  | Reviews for Celeste from [Cel] . . . . .                                 | 9  |
| 5  | [Rws] Rain World Screen shot . . . . .                                   | 10 |
| 6  | Reviews for Rain World from [Rws] . . . . .                              | 11 |
| 7  | Questionnaire Header . . . . .   | 14 |
| 8  | [Shs] Steam OS statistics . . . . .                                      | 17 |
| 9  | [Shs] Steam Hardware Statistics . . . . .                                | 17 |
| 10 | Decomposition Diagram . . . . .  | 22 |
| 11 | Room layout example . . . . .  | 23 |
| 12 | Flowchart for generating rooms . . . . .                                 | 24 |
| 13 | [Dis] Room Movement Direction . . . . .                                  | 25 |
| 14 | Flowchart for spreading rooms . . . . .                                  | 27 |
| 15 | Flowchart for deleting rooms . . . . .                                   | 29 |
| 16 | Bowyer-Watson Classes . . . . .  | 30 |
| 17 | Bowyer-Watson Algorithm Flowchart . . . . .                              | 31 |
| 18 | Flowchart for Edge Deletion Algorithm . . . . .                          | 34 |
| 19 | Flowchart for Generating spawn and boss rooms . . . . .                  | 36 |
| 20 | Flowchart for recursive function to find entire section . . . . .        | 39 |
| 21 | Flowchart for detecting if a new section needs to be made . . . . .      | 41 |
| 22 | Flowchart for detecting sections in the program . . . . .                | 43 |
| 23 | All possibilities for points . . . . .                                   | 45 |
| 24 | Flowchart for turning edges into horizontal and vertical edges . . . . . | 46 |
| 25 | Flowchart for the given pseudocode . . . . .                             | 48 |
| 26 | Flowchart for generating the tilemap . . . . .                           | 50 |
| 27 | Flowchart for spawning the player . . . . .                              | 52 |

# 1 Abstract

I decided to create a game for my coursework. I had researched the market to influence my game to make it more appealing to the target audience. Creating the game was very challenging and challenged my limited understanding of c#, and forced me to learn the language in greater detail, it also challenged the methods that I tend to favour when approaching a challenge forcing me to explore new ways of computational methods. I decided on a 2D rouge-like game with a heavy focus on melee combat and movement. For this I needed to make a character controller, dungeon generation algorithm, enemy AI and balanced items for the player. Each requiring it's own delicate calibration to make the game fun.

## 2 Introduction

For my project I intend to create a 2D rouge like game which is focuses on melee combat. This game would be like Rain World with Celeste movement. I chose this style because there are few games which focus on melee combat and this would help it stand out amongst the other games, the rouge like aspect would allow every run to be unique and distinct. Allowing people to replay the game multiple times without getting bored.

The melee combat will be close range and it will allow people to play in distinctive styles, for example people will be able to change their weapon and ability to suit the play style that they are looking for, making any play style viable.

There will be progress through floors. When the player completes one floor they can move on and enter the next floor. At the end of each floor there will be a boss that the player must defeat to progress.

The loot and map generation will be random meaning that the player cannot memorize the layout and do the same thing every time but need to adjust to the environment, making the game more challenging.

I plan for there to be different enemies that have different attack patterns that will not be entirely random meaning that the player can learn how to effectively defeat the enemy. The enemy difficulty will increase as the player progresses through the floors.

## 3 Analysis

### 3.1 Computational Methods

Computational methods are computer-based methods which are used to solve problems. They are suitable for my project since I want to make an entertaining game. And I can accomplish this by creating features using the following methods.

**Decomposition** – Splitting a large problem into smaller problems which are more manageable. This would help with my project as I am not going to be working in every aspect of the game at once, decomposition allows me to work on one part of the game at a time since debugging one small piece of code with a couple of errors is a lot easier than debugging the entire game code with many, many errors. It also allows me to work on different parts of the game independently from another part, since each problem is broken into it's own self contained module.

**Divide and conquer** – Dividing a problem into smaller problems until they are small enough to be solved directly. This would allow me to develop one aspect of the game at a time and make steady progress on the game. Since each aspect is finished and polished by the time I move on to the next aspect. This can be utilised if decomposition doesn't break the code into small enough segments.

**Abstraction** – Removing the additional detail allowing me to focus on the main points rather than wasting time trying to work on pointless detail that will not be noticed. This will allow me to use my time efficiently as the main features will get the most amount of time making the basics of the game reliable. For example when generating a map for the game I could use a tile map to remove the fine detail of a map to reduce it to squares.

**Modular design** – Subdividing the problem into smaller modules in my case these would be: physics engine, movement, abilities, characters etc. These allow me to priorities the modules which need more attention and keep organized as each module can have its own separate folder for all the things needed for that module.

**Algorithms** – Algorithms allow me to implement features that do not need anything but computer processing to be solved. For example, enemies will use an algorithm to detect and attack the player, the scenery might be made by an algorithm etc. These allow the computer to do specialized processing for the problem that needs to be solved.

**Selection** – Allows for choices to be made in the code. For example, if statement is a type of selection since a condition needs to be met for the code to be executed. This is useful in many scenarios, just to list a couple: if the player is in the air, they should not have the ability to jump. If the players health drops to zero they should be sent to the game over screen and many more.

**Iteration (looping)** - Allows a certain piece of code to be ran multiple times, these can be count controlled or condition controlled. Loops which will continue to cycle the code until it has reached a stopping condition. These are useful since in a 2D game the players input needs to be recorded every frame to minimize latency, the code needs to sort through those inputs every to make the necessary adjustment to the character and environment all in the same frame. These tasks are repetitive therefore the code will be the same every loop and iteration is perfect for that.

**Visualization** – The player will not be able to process the raw data outputted by the computer therefore it needs to be put in a format which is comprehensible for humans. There will be a lot of data processing while the game is running such as player position constantly moving, enemies moving and attacking the player. If that were outputted as a string of numbers to the player, they would have no idea what is going on therefore visualization is used to allow the user to interact with the program.

**Pattern recognition** – It would be useful for the computer to recognize patterns as a certain pattern could be used as a condition for selection this could be useful as when the player attacks the computer could recognize this and react accordingly to make the fight interesting. Or it can be used in fighting games to make a move since in some games you can chain inputs to do a special attack.

These computational methods are suitable for solving my problem because each problem can be broken down into smaller sections that are able to be solved by a machine.

## 3.2 Researching The Problem

### 3.2.1 Risk of Rain 2



Figure 1: [Ror] Risk of Rain 2 Screen shot

Risk of Rain 2 is a rogue like, third person shooter. A rogue like game means that when you die in the game you must restart the entire game, which means that it does not take long to complete and it can be replayed multiple times, since it has a large variety of items which can be combined to make many unique runs. It has a unique concept since as the time goes up so does the difficulty, meaning that the more time you spend looting the more powerful enemies become creating a unique stressful and fast paced shooter.

### Controls

| Key/Button | Action   |
|------------|--|
| WSAD       | Moving Around  |
| Space      | Jump   |
| E          | Interact with the environment (open crates etc.)                             |
| Q          | Activate equipment (item that can be used by every character)                |
| Ctrl       | Toggle sprint (to move faster)   |
| M1         | Primary Skill (unique to character, usually damaging)                        |
| M2         | Secondary Skill (unique to character, usually damaging)                      |
| Shift      | Utility Skill (unique to character, usually movement)                        |
| R          | Special Skill (Unique to character, usually heavy damage, and long cooldown) |
| Tab        | Info Screen (shows the statistics of the current run)                        |
| M3         | Ping (allows players to communicate in game)                                 |

### Characters

Commando and Huntress – Both are beginner friendly characters which have a basic set of skills and utility to ease the player into the game. They are the first characters you unlock and they are both good characters even after the player has learn to play the game.

Every character apart from these two needs some sort of challenge to be completed before the player gains access to them. This allows the player to progress at their own pace.

Acrid, Artificer, Bandit, Captain, Engineer, MUL-T and REX – Once these characters are unlocked, they expand the possible play styles possible to the play allowing the player to play the game in any way they want to for example, Engineer has turrets and a shield if the player wants to bunker down, while Bandit has an invisibility cloak and shotgun allowing for the player to get close and personal.

Loader and Mercenary – Both have the highest skill ceiling (lots of things to master) and both have lots of unique tech (advanced strategies to use utility to achieve a certain result) for example Loader can launch out her pylon and grapple to it immediately after launching the player a great distance

Railgunner and Void fiend – Both DLC characters which is a good example of the developers expanding the play style since none of the other characters had long range and these characters fit that style perfectly since Railgunner has a rail-gun which is like a sniper allowing the player to keep their distance and pick of enemies one by one.

Heretic – A secret character which is unique since they cannot be unlocked, and the character is bound by the run as they are unlocked by picking up a combination of items on that run.

### Aim of the game

The aim of the game is to get loot and activate the teleporter, after this a boss will spawn, which you must kill to progress. You get teleported to a different stage and the process repeats until stage 5 and after that you get teleported to the moon to kill final boss (Mythrix) or obliterate yourself which has a more secret ending which needs the player to enter a portal on stage 8 after they loop (return back to stage 1 and keep all their items), sometimes a large purple portal can appear which teleports the player to a different realm (with some pretty interesting lore) and they have to defeat Voidling which gives the player an alternate ending.

### Target demographic

Teens and older since the game contain blood, drug references and fantasy violence. It is aimed at people with all skill types as the game has different difficulty rating which allows the player to play at the level that they are comfortable with.

### Reviews

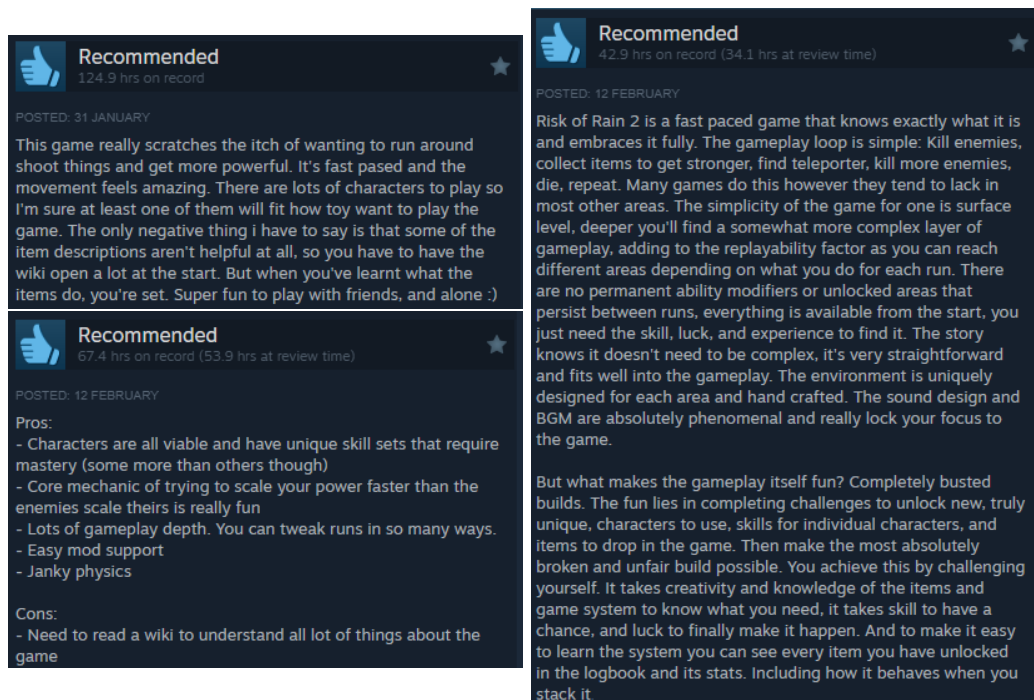


Figure 2: Reviews for Risk of Rain from [Ror]

These reviews show that players enjoy the game for its item variation. This is because the game has 148 unique items that can be combined with each other to make unique builds making the only limitations the player's imagination and luck to gain those items. One of the things that many players complain about is the item descriptions in the game since the in game descriptions often weren't helpful as they would give the general idea but to fully understand it you would have to go to a third party source to look at the accurate description or alternatively in the log book, which is only accessible from the main menu, so it's not useful in game, where it's most commonly needed. However the large amount of items can be very overwhelming to a new player, this is only made worse by the vague description the game provides. The movement, character variations and the balancing of these things is greatly appreciated by the community and therefore it should be something I aim to incorporate.

### Good Qualities

- The game is very beginner friendly because of the difficulty options provided.
- The variety of items allows the player to replay the game with many different combinations and the game does not feel repetitive.
- The different ending makes the player want to experience them all.
- The variety in characters makes any play style suitable.
- The game is never "too easy" since there are eclipse challenges which get progressively harder
- The many secrets in the game allow insentiences the player to explore and solve puzzles to unlock new aspects of the game.
- the game is fast paced and fun

### Bad Qualities

- Once you loop a lot the game gets very chaotic and resource intensive because there are so many enemies with different attributes that can create lots of projectiles that slow down the game a lot.
- Some things can kill you in one hit making death unavoidable in some cases which can feel unfair.
- Sometimes you get runs where you get no good items, and the game becomes a lot more difficult because of bad luck.
- Item description can be vague in game where you need it the most.

### What I would include

- The fast pace achieved by constantly giving the player some enemies to deal with
- The different difficulties allowing the player to play at a level that their comfortable with, this would make the game approachable to anyone with any skill level.
- I would love to include the item variation, however the time limitation wouldn't allow for this as the items would be not balanced and some are made redundant whereas others are too powerful, so I think it would be best to include a few items that have been specially tailored.

#### 3.2.2 Celeste

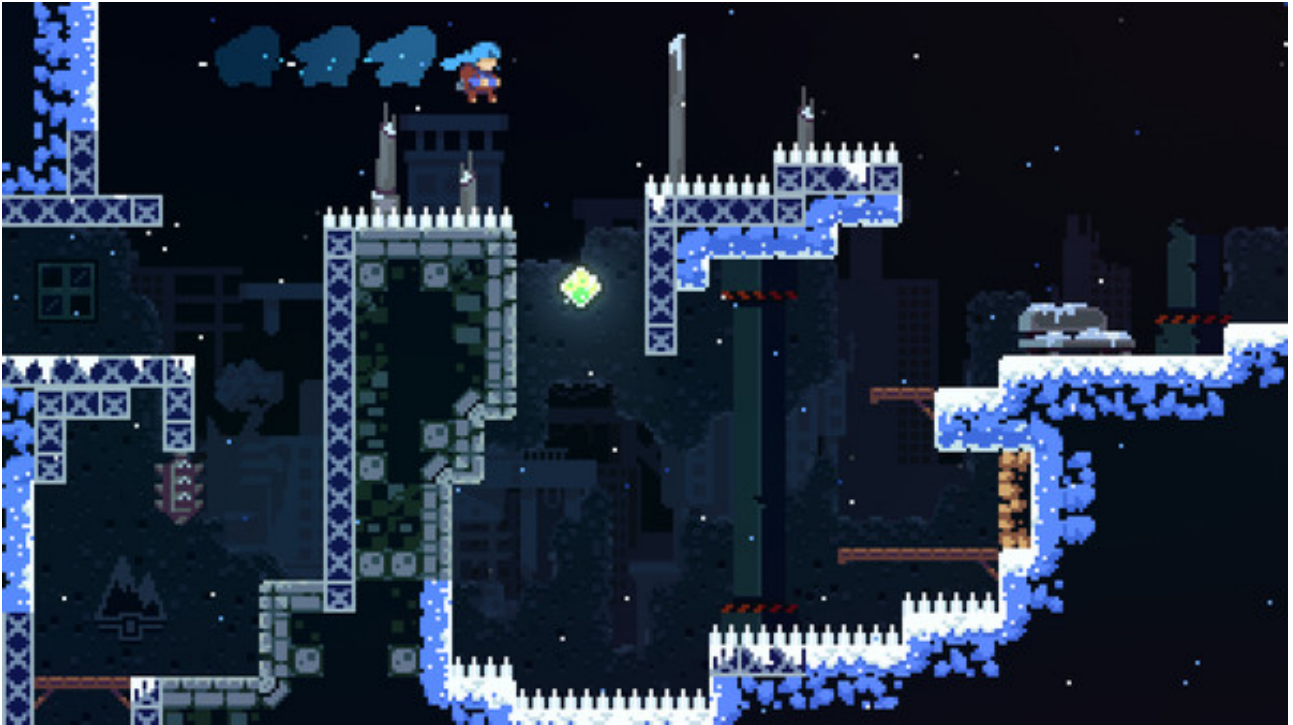


Figure 3: [Cel] Celeste Screen shot

Celeste is a challenging 2D platformer where the aim is to give precise inputs that will clear the level and allow you to progress into the next room. It takes a long time to complete as the game is very difficult and constantly introduces new features that are harder than the last.

#### Controls

| Key/Button | Action        |
|------------|---------------|
| WSAD       | Moving Around |
| Space      | Jump          |
| K          | Dash          |
| L          | Grab          |

#### Characters

Madeline is the protagonist of the game and as the game progresses, as the game progresses they get more and more abilities and the level design reflects this as the game presents rooms that can only be solved with the new skill that the player was shown. The game never gives the character any new skills are just shown to the player and never granted or unlocked meaning that the player can revisit earlier levels and complete them in newer and more efficient ways. The player does get an extra dash in some of the later rooms in the game and this is the only upgrade that the player gets.

#### Aim of the game

The aim is to get to the top of the celeste mountain climbing the mountain one room at a time. Rooms get harder the further you progress in the game, the length of these rooms can vary drastically, the same applies



to the difficulty. The game itself doesn't have that much levels however each level is very difficult which makes completing one very rewarding.

### Target demographic

Anyone as the game doesn't have any violent or difficult topics discussed, however it will cater better to a slightly older audience because of it's difficulty. The game also targets people who are more experience in platformers as the difficulty of the levels scales quickly.

### Reviews

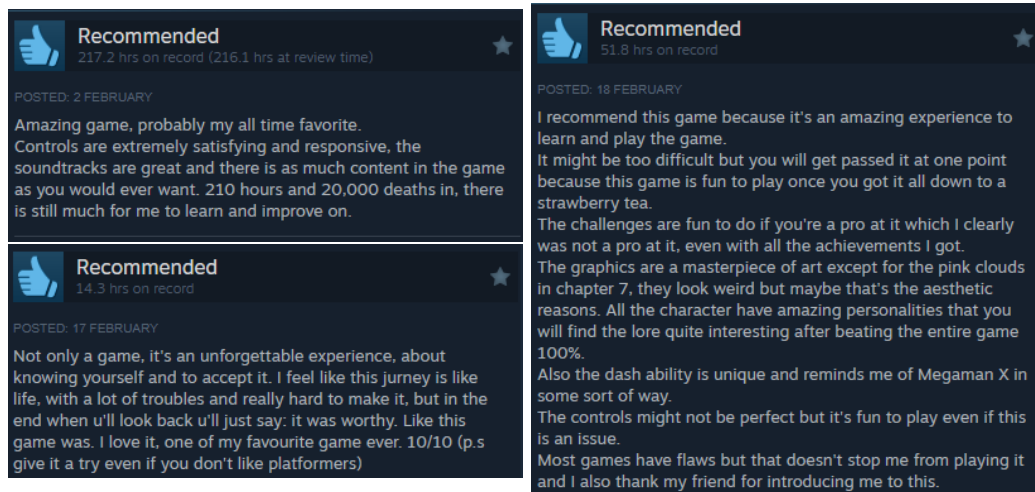


Figure 4: Reviews for Celeste from [Cel]

Many of the players enjoyed the story of the game because of it's beautiful writing. However I am not good at writing therefore my game will not be able to create a compelling story, therefore I would like to replicate the precise and fluid movement of the game. The game is still fun as players are actively failing due to it's difficulty. As seen by the player with 20,000 deaths and still speaks positively about the game. The is because of the game's fair design and the rewarding feeling that comes from beating a level.

### Good Qualities

- Beating a room feels rewarding and it makes the player want to continue playing.
- The movement is very simple which makes it easy for anyone to start.
- The movement feels fun due to the combinations with different movement techniques.
- The movement feels consistent, if you die it feels like it's your fault and not the game being unfair.

### Bad Qualities

- Getting stuck on one room can be very frustrating.
- To progress far into the game you need to dedicate a lot of time and have a lot of skill.

### What I would include

- A similar art style, since the simple pixel art would be easier to implement than hand drawn characters.
- Similar movement system, but something that isn't as complicated so it doesn't overwhelm the player since I also want to add a combat system.

### 3.2.3 Rain World



Figure 5: [Rws] Rain World Screen shot

Rain world is an open world game meaning that the player can go explore anywhere to their heart's content. It's also very difficult, and focuses on treating the player as part of the ecosystem rather than a separate entity, for this reason enemies treat the player as any other rival creature and focuses on their own survival rather than killing the player, which is common in most other games.

#### Controls

| Key/Button | Action        |
|------------|---------------|
| WSAD       | Moving Around |
| Space      | Jump          |
| E          | Grab          |
| Q          | Throw         |
| Z          | Map           |
| Escape     | Pause         |

#### Characters

The player plays as a slugcat which is a small creature with the ability to wield rocks and spears. They can also befriend other wild creatures. They can also interact with scavengers if the player's reputation is good with the scavengers, however this isn't the case for all slugcats as scavengers will become hostile if the player's reputation is low.

Monk - The easy mode of rain world, where enemies are less common and less aggressive. This character is good for people playing the game for the first time as it allows them to experience the game in a less harsh environment than usual.

Survivor - Regular difficulty of the game

Hunter - Hard mode of rain world where some special tougher enemies spawn that don't usually spawn, enemies are also more common and aggressive. There is also a time limit as the hunter has a limited amount of cycles (days) unlike any of the other characters, hunter also has the ability to consume dead animals.

Rain world also has a DLC called downpour which adds 5 new characters, where each character has unique abilities and objectives, the different characters are unique to the base game characters as they have their own

abilities which also come with their downsides. Each character is also set at a different time in the timeline meaning that the layout of the game will be similar however each room will be different for each character and the enemies in that area will also be different. These new characters each require a different approach for the game due to their different downsides, for example gourmand gets tiered after throwing a spear, forcing the player to plan their combat as they will get tiered if they spend too long in combat. Whereas the saint isn't able to use spears because it's a pacifist, but they have a grapple which it can use to traverse the land faster, forcing the player to avoid combat at all costs.

### Aim of the game

The aim for each character in the game varies, however they all need to progress through the environment to reach their goal, often resulting in combat with the wildlife. The player must also gain enough food to be able to sleep in a shelter and avoid the rain that comes at the end of each cycle (game equivalent of a day). Dying results in the player respawning in the shelter they previously slept in and losing a karma point, however for each successful cycle that the player completes they will gain a karma point. Karma is important as it allows the player to traverse between regions, since regions are separated by karma gates which only let you through if your karma is above a certain level.

### Target demographic

Anyone as it doesn't contain any graphic scenes or references to difficult topics. However the game caters more to people who are willing to spend time trying to understand it as it's a very challenging game which is very harsh for a new player, lacking any kind of tutorial or explanation.

### Reviews

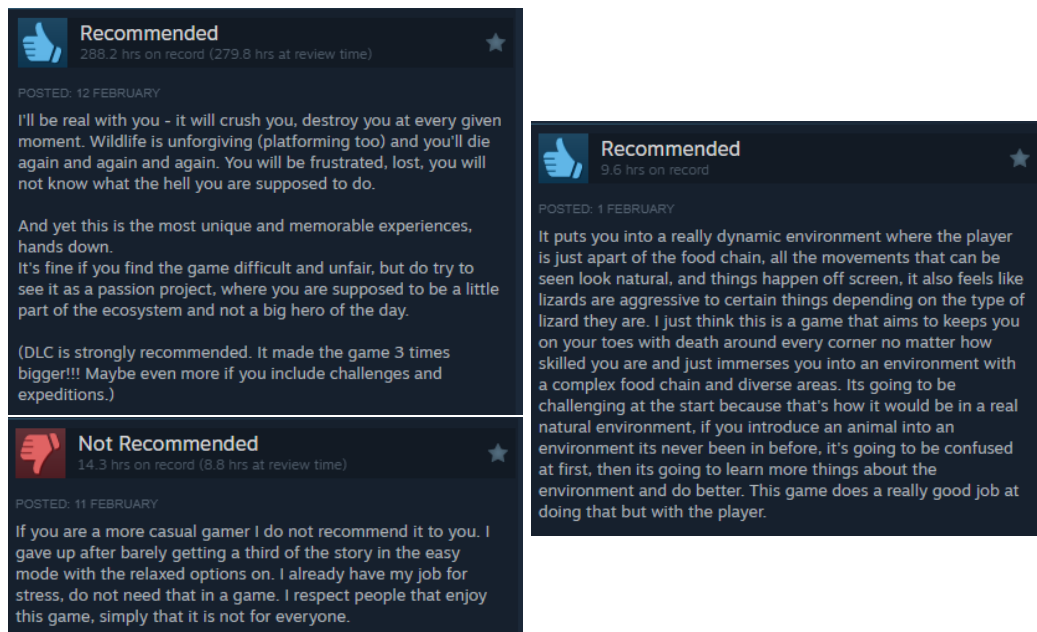


Figure 6: Reviews for Rain World from [Rws]

The community understands that the game is difficult to get into due to its lack of tutorial or any objective or guidance. However this is a positive thing for some people because they enjoyed the lack of guidance, however this isn't something that I am going to try to imitate because it acts as an entry barrier for my game and could deter a significant amount of players as seen with Rain World, since almost every player that got into Rain World enjoyed it, however almost every player has had difficulties with getting into the game. These things can be mitigated by third party sources of information, however this wouldn't be possible with my game so I would like to avoid this type of approach.

### Good Qualities

- Game play is fun and keeps the player engaged due to the many things that the player can do.
- Game has a high skill ceiling meaning there is always a way the player can improve.
- Movement is incredibly fun and feels exceptionally smooth.

- Lack of permanent upgrades makes the game focus on skill rather than items and unlocks.
- Completing a cycle is very rewarding.
- Player is treated as part of ecosystem rather than separate entity, meaning they are treated as every other enemy is by the other enemies which makes it feel more realistic.

### **Bad Qualities**

- Difficult to get into due to no tutorial, there is no way of knowing what to do when you start and the player has to figure it out for themselves, which can be enough to deter some players.
- Game is very unforgiving and a small mistake can result in the player getting killed and losing a lot of progress.
- The map is very large and the in game map system is very unreliable so it's often played with a map open on a separate tab or device.

### **What I would include**

- Variety in enemy types that act differently and need to be tackled differently
- The lack of a permanent upgrade system to keep the focus of the game on skill, due to it being a strong point of the game and it allows me to focus on other aspects

## **3.3 Stakeholders**

My game would be suited for people who have more experience with games, it will be suitable for any age over the age of twelve (might be a too violent for children under the age of twelve). A stakeholder would also be important as they could give me direct feedback for the game to improve it and make it a more enjoyable experience. They would play the game as they often play games as a source of entertainment the survey conducted in a later section shows what type of games people play and therefore would be willing to try.

Therefore, Ethan Armstrong would be a suitable stakeholder. He is 16 years of age therefore he fits the target demographic. He will give feedback through play testing and according to the feedback I will be able to adjust the difficulty and balance the aspects of the game to fit the genre and style I am going for. The feedback given to me by him will allow me to adjust the game to make it more enjoyable.

Daniel Cabrel would also be a stakeholder as they are new to gaming therefore, I will be able to make the game more beginner friendly to expand my target audience and make it more appealing to more people. He will also give feedback through play testing and interviews allowing me to make the game easy to pick up regardless of skill.

I have chosen these people since they are part of my target demographic and they will help me to make the game easy to pick up yet challenging.

## **3.4 Interview with Stakeholder**

I – Interviewer (Karol Jeziorczak)

S – Stakeholder (Ethan Armstrong)

I: Everything you say here will be recorded and used for development of my game; however, it will not be shared with any unauthorized personnel and your data will protected under the GDPR

S: I understand and agree to these terms.

I: Question 1. What is the most important feature to you in a game?

S: Exploration is very important to me; I love finding out hidden mechanics and secrets.

I: Question 2. How important is difficulty scaling with progress to you?

S: It is important to make the game challenging as it will make it more fun to overcome.

I: Question 3. How often should there be boss fights?

S: A bit after a new mechanic gets introduced so I have time to get used to it however the boss fight will be a test to see if the player can use the gimmicks

I Question 4. What type of playstyle do you usually go for?

S: I usually go for a large health build with heavy weaponry

I: Question 5. Are there any specific features you would like to see?

S: I would like to see a power slide because it makes the game seem fast paced.

I: Question 6. What are some qualities of a good combat system?

S: Enemies not having insane amounts of health as it slows down the pace of the game and makes killing them seem like it takes too long. So, with melee weapons there should be more blocking and skill involved rather than just hitting enemies and them dying?

I: Question 7. What should the final boss look like?

S: The final boss should be an enemy foreshadowed by the previous environments that has different mechanics from the different bosses that combine them all into one where they need to be interchanged, with a satisfying dying animation for the final boss.

I: Question 8. How should the inventory management system work?

S: I think that there should be a very limited amount of inventory room so that management and planning become very important. Also, so that the focus of the game isn't shifted too much from the combat, as it is the main focus of the game.

### 3.5 Letter to Stakeholder

31 Bucannon Road  
CV22 6AZ  
20/11/2022

Dear Ethan Armstrong,

My name is Karol Jeziorczak. I am studying computer science (OCR specification) at Rugby High School; the coursework requires me to create a game.

I am asking you to become a game tester for my game so that I can get feedback from you as to how to improve the game. In the future I will ask you to give me feedback on the game to improve the quality and usability of the game, this will be in the form of interviews and game testing. This will hopefully improve the quality of the game and allow me to develop this project further into a fully functional product.

All the data gathered will be kept secure in order with the GDPR (General Data Protection Regulation). This means that your data will be kept secure and will not be shared with any unauthorized personnel. I will do everything that I can to protect your data. However, if the data will get leaked, you will be the first to get notified.

Thank you for your time and co-operation (if you choose to). I am looking forward to working with you.

Sincerely,  
Karol Jeziorczak

# Game Questionnaire

All data collected will be held with regulation to the **GDPR**, the data will be analysed and used to develop my Computer Science Coursework. The results will be shared with examiners from OCR as part of my report but no personal details will be included as you are able to answer anonymously.

This questionnaire regards my Computer Science coursework and it intends to analyse the target audience, to make sure that my product will be suitable for the target audience and meet their needs.

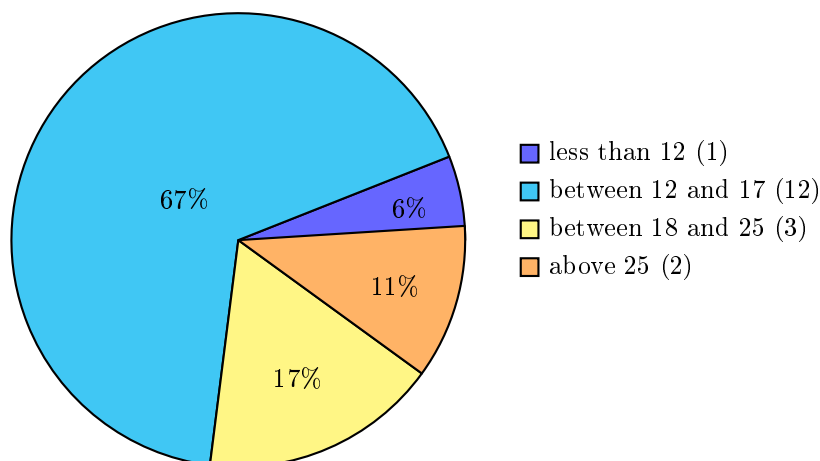
Figure 7: Questionnaire Header

## 3.6 Questionnaire

A questionnaire will allow me to analyse the market and make a game that people want to play. It also allows me to ask the intended audience for any features that they want.

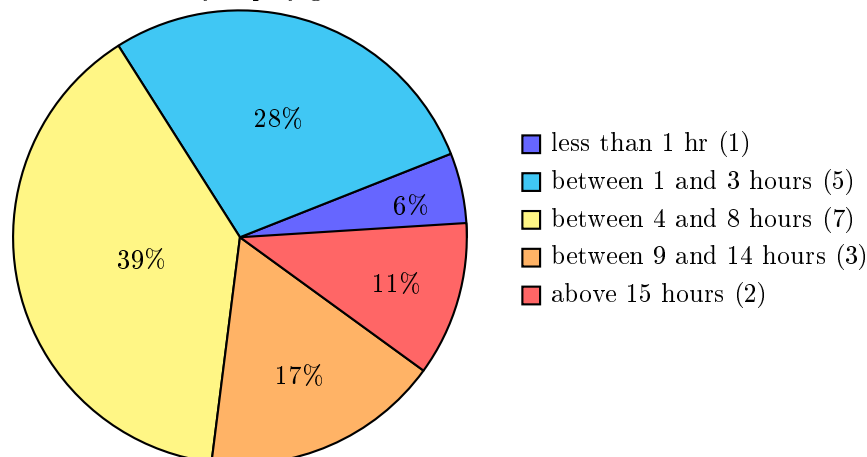
### Analysis

Q1: What is your age?



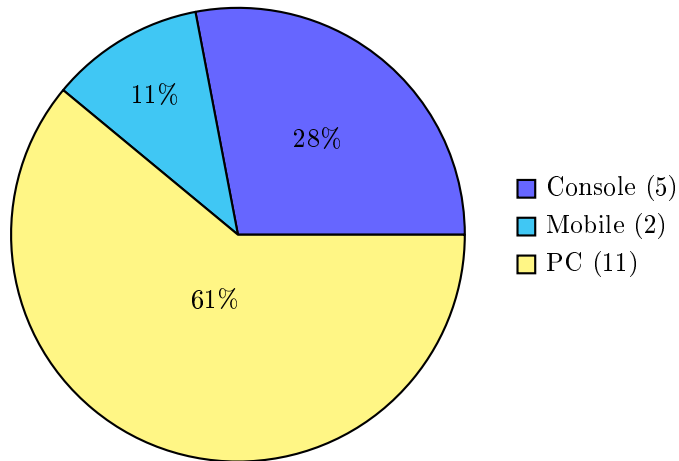
We can see that the target demographic is teens and above meaning that the game can include some violence. It is important to make the game appropriate to the target demographic as they will be the majority of the player base.

Q2: How much do you play games a week?



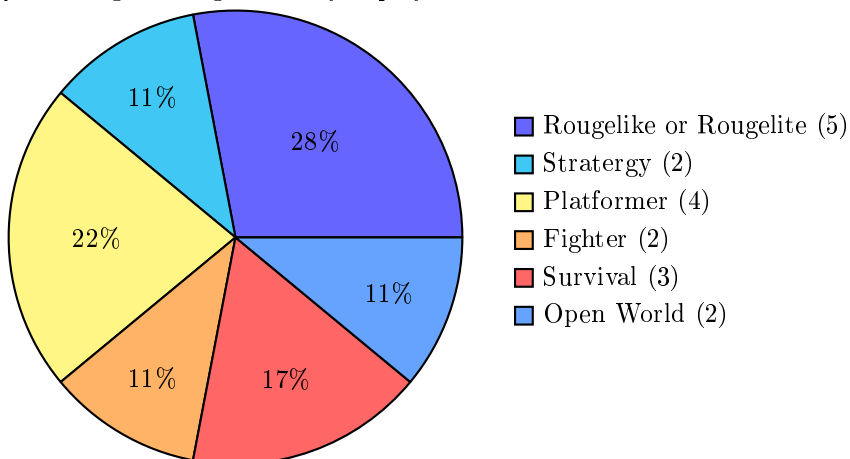
It is important to balance the progress made and the time invested as if people only play 1-2hrs a day as shown by the questionnaire, it is important to make the player feel like they made progress in that time so that they are willing to continue playing. This could be implemented by each run being 30-40 mins allowing the user to have a couple runs in a session.

Q3: What platform do you play games on?



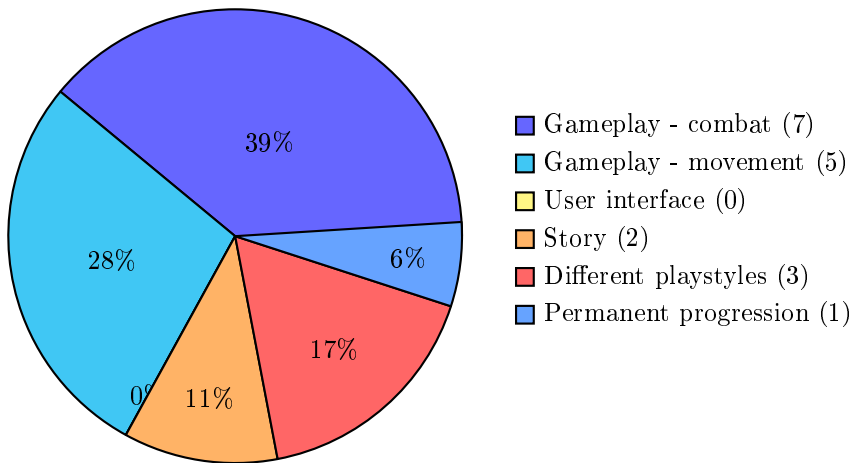
Most people answered replied that they play on pc meaning that it should be ported to pc first. A significant amount of people also play on console so it would be good to port the game to console if possible.

Q4: what genre of games do you play?



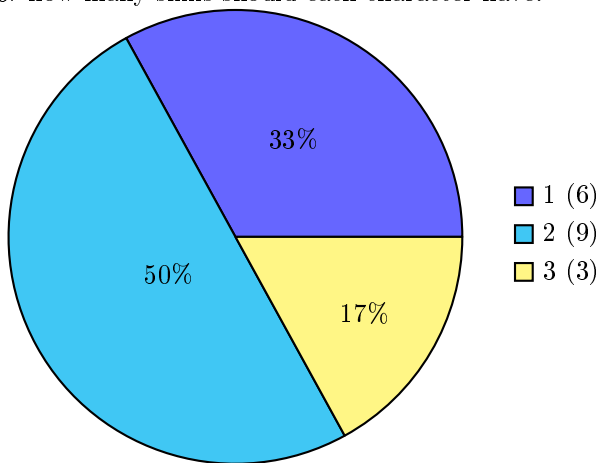
Most people answered Roguelike/Rougelite and platformer meaning I should focus on these two aspects the most when making my game. Survival will be implemented though planning ahead this might be what routes to take and the risk associated with those routes as well as the risk of combat. As well as management of resources to make sure you survive to the next stage.

Q5: What mechanics are important to you in a game?



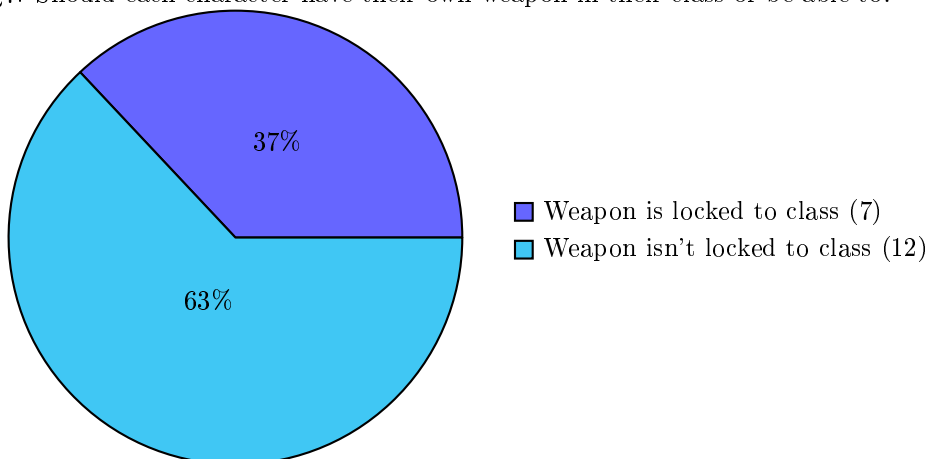
Combat and movement were the most common option so they should be the most polished systems in the game. Though different playstyles was something I should consider. Permanent progression wasn't a popular choice so a roguelike style would be more suitable

Q6: how many skills should each character have?



Most people answered 2 so it would be important to make the skills distinct and unique to allow for different play styles. One ability could be active that the user can activate when needed e.g. throwing grenade and it will have a cool down. The other ability could be a utility skill which can be picked regardless of character and is a limited resource and refreshes once a special item is picked up e.g. throwing knife that does damage, refreshes when more are picked up. There could also be a passive ability that the player doesn't need to do anything to activate it, it will be permanently active in the background (e.g. increased movement speed), this can make every character feel unique.

Q7: Should each character have their own weapon in their class or be able to?





The majority wanted to have weapons that are picked up and not locked to a class therefore I will give each character a starting weapon and give them the ability to change this weapon if they wish to, allowing the player to experiment with different weapons that may suit their play style.

### 3.7 Analysing Market

Steam does a hardware analysis on all of it's users every month so I will be able to make a game that will meet the requirements of the majority of players. So all the data in this section is taken from [Shs]

#### System OS Analysis

| MOST POPULAR              | PERCENTAGE | CHANGE  |
|---------------------------|------------|---------|
| Windows                   | 96.56%     | -0.87%  |
| Windows 10 64 bit         | 53.53%     | -12.05% |
| Windows 11 64 bit         | 42.04%     | +11.51% |
| Windows 7 64 bit          | 0.68%      | -0.38%  |
| Windows 8.1 64 bit        | 0.16%      | +0.03%  |
| Windows 7                 | 0.06%      | +0.06%  |
| OSX                       | 1.53%      | +0.34%  |
| MacOS 14.0.0 64 bit       | 0.34%      | -0.06%  |
| MacOS 14.1.1 64 bit       | 0.20%      | +0.20%  |
| MacOS 14.1.0 64 bit       | 0.18%      | +0.18%  |
| MacOS 13.4.1 64 bit       | 0.05%      | -0.01%  |
| MacOS 13.5.2 64 bit       | 0.05%      | -0.05%  |
| Linux                     | 1.91%      | +0.52%  |
| "Arch Linux" 64 bit       | 0.15%      | +0.05%  |
| Ubuntu 22.04.3 LTS 64 bit | 0.13%      | +0.04%  |
| Linux Mint 21.2 64 bit    | 0.08%      | +0.03%  |
| "Manjaro Linux" 64 bit    | 0.07%      | +0.02%  |

Figure 8: [Shs] Steam OS statistics

Deciding which systems I will be able to support is going to be an important decision I have to make to make sure that the users can play the game us. Figure 8 shows us that the overwhelming majority of players use windows, this tells me that I should prioritise porting the game to windows as it will make it available to the most amount of players as possible. Mac OS (OS x) only account for a small percentage of players. The difficulties that come with porting my game to Mac simply makes it an inefficient use of time. Linux also accounts for a small percentage of players however porting to Linux is very easy in Godot so it may be something to consider in the future if I have time.

#### Hardware Analysis

| ITEM                             | MOST POPULAR            | PERCENTAGE | CHANGE  |
|----------------------------------|-------------------------|------------|---------|
| OS Version                       | Windows 10 64 bit       | 53.53%     | -12.05% |
| System RAM                       | 16 GB                   | 49.88%     | +1.82%  |
| Intel CPU Speeds                 | 2.3 Ghz to 2.69 Ghz     | 21.15%     | -1.40%  |
| Physical CPUs                    | 6 cpus                  | 31.88%     | -7.67%  |
| Video Card Description           | NVIDIA GeForce RTX 3060 | 4.89%      | -4.79%  |
| VRAM                             | 8 GB                    | 31.23%     | -3.63%  |
| Primary Display Resolution       | 1920 x 1080             | 60.09%     | +1.08%  |
| Multi-Monitor Desktop Resolution | 3840 x 1080             | 60.02%     | +0.04%  |
| Language                         | English                 | 38.02%     | +9.62%  |
| Free Hard Drive Space            | 100 GB to 249 GB        | 23.46%     | +4.34%  |
| Total Hard Drive Space           | Above 1 TB              | 51.99%     | -13.16% |
| VR Headsets                      | Oculus Quest 2          | 40.45%     | +0.75%  |
| Other Settings                   | LAHF / SAHF             | 100.00%    | 0.00%   |

Figure 9: [Shs] Steam Hardware Statistics

I intend to make a 2D game. These types of games don't tend to be resource intensive. The most important statistics in figure 9 are system RAM, Intel CPU speeds (More players use Intel CPUs rather than AMD

CPU's so this is a valid statistic to use), Physical CPU's, VRAM, Free Hard Disk Space (the ones with a red mark next to them). None of the statistics should be limiting for my project as they would be more important to consider for a 3D game using a fancy rendering system. However my game is 2D so it should be able to run smoothly on most devices.

### 3.8 System Requirements

All the requirements are taken from [Spe], the official Godot 4 documentation.

#### Minimum Hardware Requirements

- Processor: x86\_32 CPU with SSE2 instructions, or any x86\_64 CPU
- Graphics Card: Integrated graphics with full Vulkan 1.0 support
- Memory: 2GB
- Storage: Fill out when done
- Sound Card: yes

#### Input devices:

- Keyboard  
Allows you to input characters that can be used to control the player
- Mouse  
Inputs 2D vector coordinate for mouse position, isn't needed for the 2D vector aspect but it can be used by the player as an optional input method if they desire to use it

The input devices allow the user to input signals that the computer to process and adjust accordingly based on the inputs provided.

#### Output devices:

- Display – Outputs image
- Speakers – Outputs sound

An example of visualisation since the raw data is presented in a way that the user can comprehend. It also allows them to understand what is happening in the game and feedback on any inputs given.

#### Minimum Software Requirements

- OS: Windows 7+

### 3.9 Limitations

Compared to the large studios that produced the games that I researched I have a significant amount of limitations that could slow down the progress of my game development. Such as manpower as studios usually consist of many specialised and trained people. However I am trying to make a game on my own, this means I will need to do everything on my own, from game assets and art to audio design and code. This could be mitigated slightly by the free assets available online which would save me time as I don't have to create them myself.

Another limitation is my knowledge and experience in developing games as I only have experience developing one small game for a game jam over the course of a couple days. The project was very limited and used GD script, whereas I would like to use c# to develop my game. I could overcome this by following tutorials and doing research to make myself more comfortable with developing my own methodology and ideas.

I am also limited by time since there is a deadline I must reach so I can't continuously develop the game over the course of a couple years. This can be less impactful with good time management and use of abstraction so that I can focus on the main points of the game when developing and spend less time on small features that will go unnoticed for the most part.

### 3.10 Essential Features

One essential feature is the user interface as it allows the user to navigate any menus and displays any important information to the player. The main menu is the first screen that a player would see therefore it is important to give a good first impression and set the tone of the game and allows the user to start a game and customise their settings. The pause menu is also an important feature of the user interface as it allows the player to pause the game and customise their settings. These things are standard in most games so it would be good to implement it in my game. The HUD (Heads Up Display) is the only user interface that the user sees when they are in game, so it's important to keep it clean and out of the way as it will be overlayed on the users screen at all times. It will also be important to have it display the relevant information, such as the player's health, inventory, and possibly a timer. However not all players will want all of this information on screen at all times so it will be important for the HUD to be customizable to suit a player's needs.

Audio is another key feature of games as audio queues can help players fight an enemy, or get feedback since they can hear the sound of their character getting hurt and immediately know that their health has went down without having to look at their health bar. Or when a hit from the player connects with an enemy they know that they damaged the enemy and it also adds to the user experience. This is observed in all the other games I researched, therefore it would be good to include. In game music also sets the tone of the game since if the game changes theme to a combat theme, the player can prepare for combat before the enemies show up.

The type of gameplay that I will focus on can be split into two main sections, player combat and player movement.

Having a good combat system is important as the player will engage with it very often so it's important for it to feel responsive and fun. This could be done by having a variety of weapons that work well together with the movement system.

The movement system will be how the player traverses the level, so having a responsive and fun movement system will be important as the player will always be interacting with it. Allowing for the player to combine movement and combat would create a lot of combinations the player could discover and experiment with.

Level generation will also be important as it creates the environment the player will experience the game in. So it needs to work well with the movement system for the levels to be easy to traverse through. Since if the player is unable to reach an important room that could hinder their progress in the game.

### 3.11 Success Criteria

#### 1. User Interface

##### 1.1. Main Menu

1.1.1. Does it have a new game button?

Allows the player to start the game.

1.1.2. Does it have an options button?

Allows the player to customise the game to suit them

1.1.3. Does it have a tutorial option?

Takes the player to a tutorial level where they can learn the game, important for new players who are playing for the first time.

1.1.4. Does it have a interesting background?

Gives a good first impression to the player when they open the game.

1.1.5. Does it have a quit game button?

Allows the player to close the game.

1.1.5.1. Does it ask you "are you sure"?

Makes sure that the played didn't press the button by accident.

##### 1.2. Pause Menu

1.2.1. Does it pause the game?

Allows player to take a break if they need to.

1.2.2. Is there an options button?

Allows the player to change setting in game in case they need to.

1.2.3. Is there an exit to main menu button?

Allows the player to quit the current run and return to the main menu.

1.2.3.1. Does it tell you "The current run will be ended"?

Makes sure the player didn't press this button by accident and accidentally quits their run

### 1.3. HUD (Heads Up Display)

1.3.1. Is there a health bar?

Allows player to see their health

1.3.2. Is there a current level box?

Allows player to see which stage their on

1.3.3. Is there a timer?

Allows player to see how long the run has lasted

1.3.4. Are there small indicators for status effects?

Allows player to see if they are affected by and de-buffs/buffs

1.3.5. Is the HUD customisable?

Allows player to adjust HUD to their need

### 1.4. Inventory

1.4.1. Can you drop items?

1.4.2. Are there item slots?

1.4.3. Can you rearrange your inventory?

## 2. Audio

### 2.1. Main Menu

2.1.1. Is there some sort of music?

2.1.2. Do the buttons make a sound?

### 2.2. In Game

2.2.1. Is there boss music?

2.2.2. Does the game have hitting sfx?

2.2.3. Does the game have a sound for getting hit?

2.2.4. Does the game have music?

2.2.5. Does the game have a sound for dying?

## 3. Gameplay

### 3.1. Level Generation

3.1.1. Does it create a level in a reasonable amount of time?

3.1.2. Are there objects you can interact with that give items?

3.1.3. Are enemies spawned?

3.1.4. Is there a safe starting room?

3.1.5. Is there a boss room at the end?

3.1.6. Is there variety between levels

### 3.2. Player Combat

3.2.1. Does the player die when they have 0 health?

3.2.2. Can the player melee swing?

3.2.3. Can the player block?

## 4. Visuals

# 4 Designing the Solution

## 4.1 Development Methodology

There are multiple ways of developing a solution so I've compiled a table to summarise all the advantages of disadvantages of each methodology

| Methodology | Description   | Uses  | Advantages  | Disadvantages  |
|-------------|---|---|---|--|
| Rapid       | Iterative methodology where partial functional prototypes are continually built upon.   | <ul style="list-style-type: none"> <li>- Small to medium projects</li> <li>- Low budget</li> <li>- Short timeframe</li> </ul>                               | <ul style="list-style-type: none"> <li>- Flexible to changes</li> <li>- Highly usable product</li> <li>- Focuses on core features</li> </ul>            | <ul style="list-style-type: none"> <li>- Poorer quality documentation</li> <li>- Fast pace may reduce quality</li> </ul>   |
| Spiral      | Goes through four key stages: analysis, risk mitigation, development and evaluation. With each iteration building on the last.        | <ul style="list-style-type: none"> <li>- Large project</li> <li>- Risk intensive projects</li> <li>- High budget</li> </ul>                                 | <ul style="list-style-type: none"> <li>- Thorough risk analysis and mitigation</li> <li>- Flexible to changes</li> <li>- Produces prototypes</li> </ul> | <ul style="list-style-type: none"> <li>- Expensive to analyse risks</li> <li>- Lack of focus on efficiency</li> <li>- High costs due to prototypes</li> </ul>                |
| Agile       | Collection of methodologies which aims to be flexible allowing different sections to be developed in parallel in no particular order. | <ul style="list-style-type: none"> <li>- Small to medium projects</li> <li>- Unclear initial requirements</li> </ul>  | <ul style="list-style-type: none"> <li>- Makes high quality code</li> <li>- Flexible to changes</li> <li>- Regular user input</li> </ul>                | <ul style="list-style-type: none"> <li>- Poorer quality documentation</li> <li>- Requires consistent interaction between user and programmer</li> </ul>                      |
| Waterfall   | Traditional mode where there is a series of stages which are completed in sequence from start to finish.                              | <ul style="list-style-type: none"> <li>- Static, low risk project</li> <li>- Require little user input</li> </ul>   | <ul style="list-style-type: none"> <li>- Easy to manage</li> <li>- Clearly documented</li> </ul>  | <ul style="list-style-type: none"> <li>- Lack of flexibility</li> <li>- No risk analysis</li> <li>- Limited user involvement</li> </ul>                                      |
| Extreme     | Agile methodology where programmers work alongside and end user to cater to their desires.  | <ul style="list-style-type: none"> <li>- Small to medium projects</li> <li>- Unclear initial requirements</li> <li>- Require excellent usability</li> </ul> | <ul style="list-style-type: none"> <li>- Produces high quality code</li> <li>- Highly usable product</li> </ul>   | <ul style="list-style-type: none"> <li>- High cost of two working on project</li> <li>- Teamwork is essential</li> <li>- End user may not be present at all times</li> </ul> |

To create the code for the game I have designed I have decided to use an agile software development methodology as it's best suited for projects with a small to medium scale with unclear initial requirements, which applies perfectly to my project. It involves creating a prototype based on the user feedback/research and collect user feedback to refine the next iteration and continuously repeat this process until a usable product has been created. This methodology also requires a lot of user feedback which makes the final product very tailored to the stakeholders. It also able to be improved upon continuously until the stakeholders needs are satisfied and all the points on the success criteria are met.

## 4.2 Decomposing the problem

This is one of the computational methods that I am going to use for my game. This is important as it allows me to think of how the game will work at a basic level, instead of considering all the components at once.

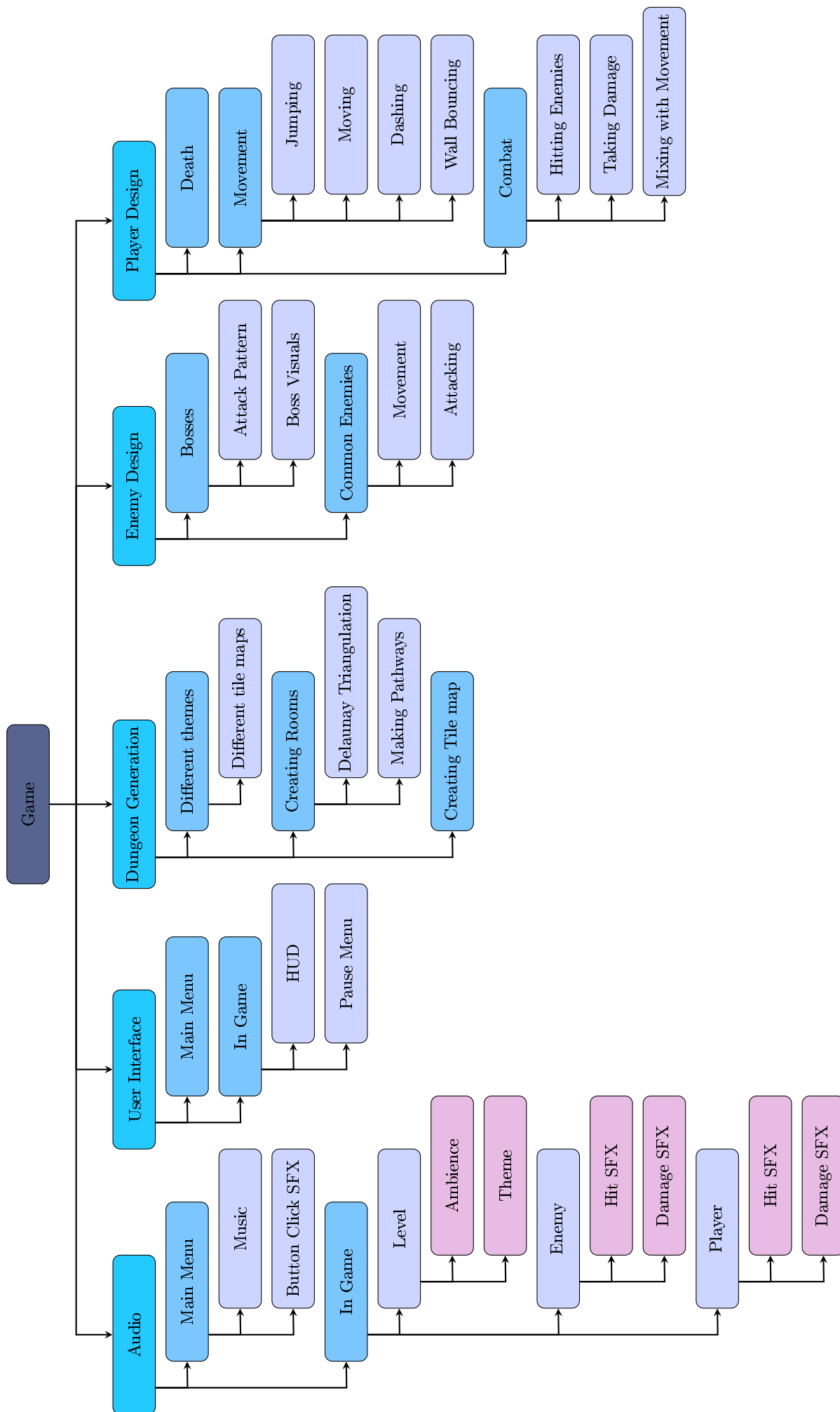


Figure 10: Decomposition Diagram

## 4.3 Variable types

### Variables

stores some sort of data in main memory

### Data Types

- Integer – whole number
- Float – real number
- Character – single letter/symbol
- String – series of characters
- Boolean – true or false

CONSTANT – represented as capitalized with underscores

variable – represented as lowercase with underscores

## 4.4 Folder Set up

## 4.5 Dungeon Generation

The Generation algorithm that I used is a modified version of the algorithm used in [Gen]. The stages I used can be summarised with the key points:

- Create rooms
- Spread rooms
- Delete rooms
- Delaunay triangulation
- Remove random amount of edges
- Add spawn and boss room
- Make sure all rooms are accessible
- Turn edges into horizontal and vertical
- Generate corridors
- Turn into tile map
- Spawn player

Some other miscellaneous algorithms used in generation

- Turning list of edges to graph
- Turning graph to list of edges

### 4.5.1 Create Rooms

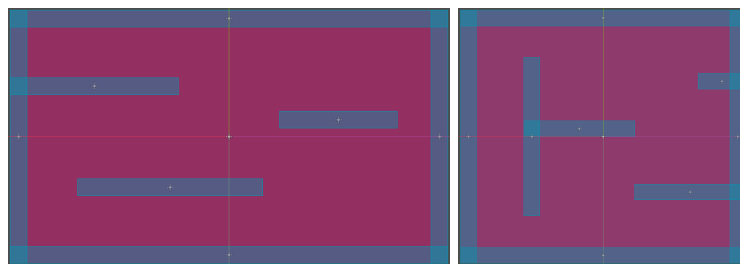


Figure 11: Room layout example

Begin with generating a defined amount of rooms. Rooms are pre-made layouts that a room as seen in figure 11. Originally I generated a cube and gave it a random size, but this would be problematic down the line since when it comes to creating a tile map it would be difficult to generate a room that would be playable. This section only needs to generate the rooms.

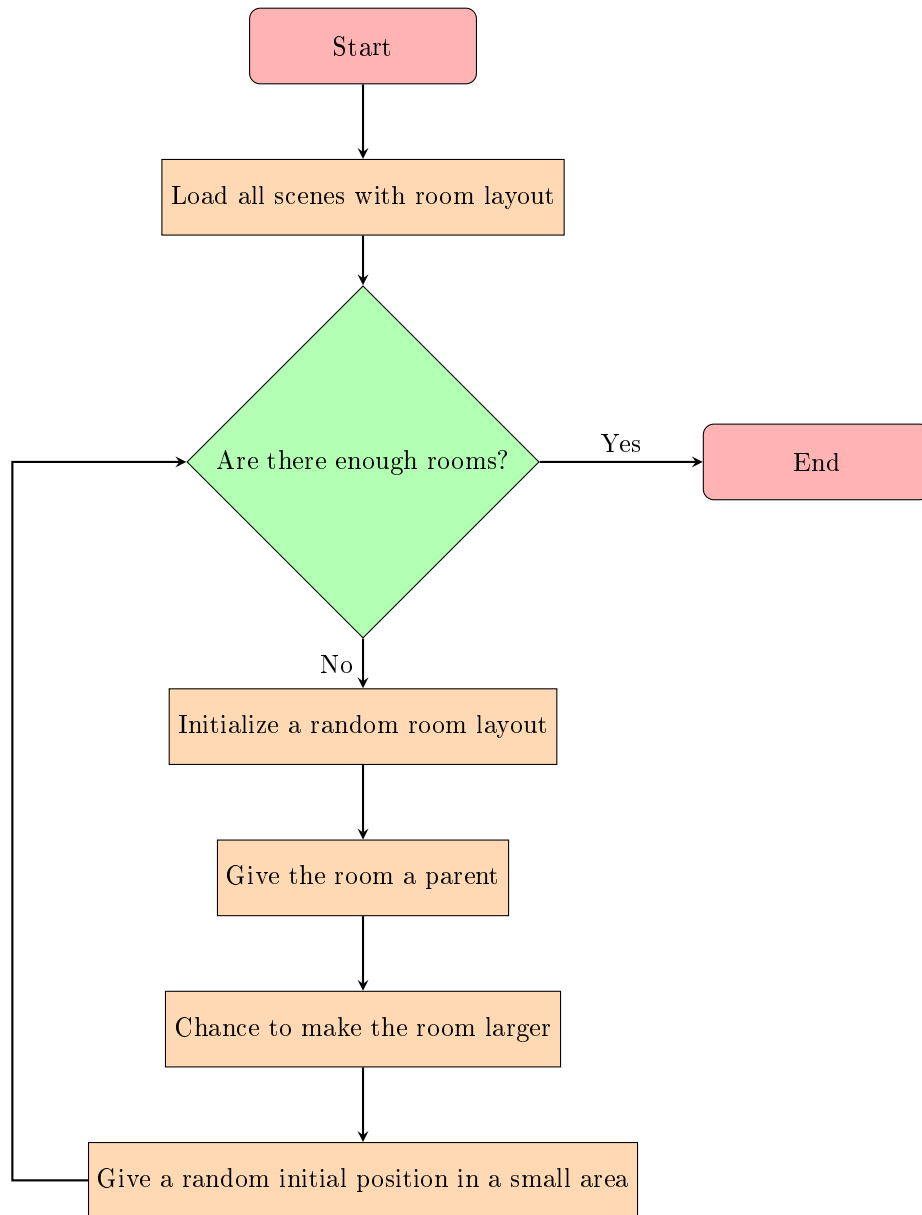


Figure 12: Flowchart for generating rooms



## Pseudocode

```
load all scenes with room layout
for (int i = 0; i < amountOfRooms; i++)
{
    initialise a random room
    give the room a parent
    chance to make the room larger //for level variation
    give a random initial position in small area
}
```

## Variable table

| Variable Name | Variable Type     | Description  |
|---------------|-------------------|--|
| amountOfRooms | Integer           | Define amount of room to be generated                        |
| scaleRange    | Vector2           | Defines the min and max scale of a room                      |
| roomVars      | integer           | Tells the program how many room variations exist             |
| randNum       | Integer           | Random integer used for deciding which room type to generate |
| rooms         | List<PackedScene> | Contains all possible room variations                        |
| instacne      | Node              | An instance of a room  |

## Identifying test data

| Goal Num | Input | Aim                                       | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time   | Annoying for player to wait too long for level to generate                                       |
| 2        | None  | There is correct amount of rooms          | Too much and a level will take too long to generate, too little and the map won't be interesting |
| 3        | None  | There is variation in the rooms generated | Makes it more interesting for the player as there is less repetition                             |

### 4.5.2 Spread rooms

Once all the rooms are all generated in a confined area they need to be spread out until they are no longer overlapping. To separate the rooms I iterated through each room object and check it for overlapping areas. I found the difference in positions to get the vector of the room relative to the original room being considered, using this I can find the direction that the room should be moved in. This idea was taken from [Dis]

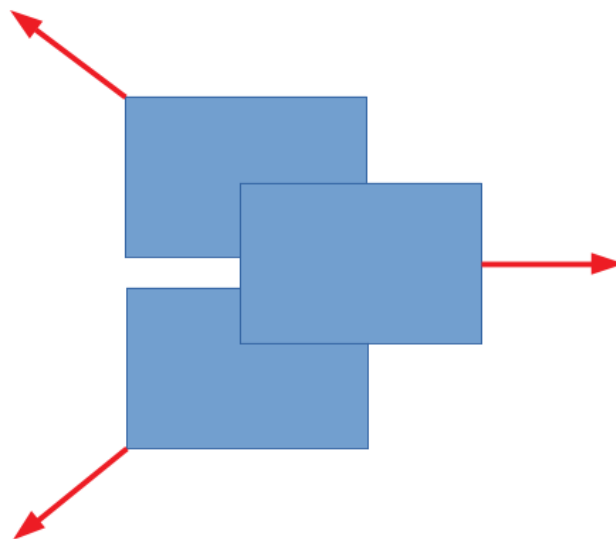


Figure 13: [Dis] Room Movement Direction

The only major difference in the way that I implemented was that I multiplied the vector by the reciprocal of the magnitude, this would make it so that when there is a colliding room closer to the room being considered it's effect on the displacement of the room is much greater than if it was far from the center of the room

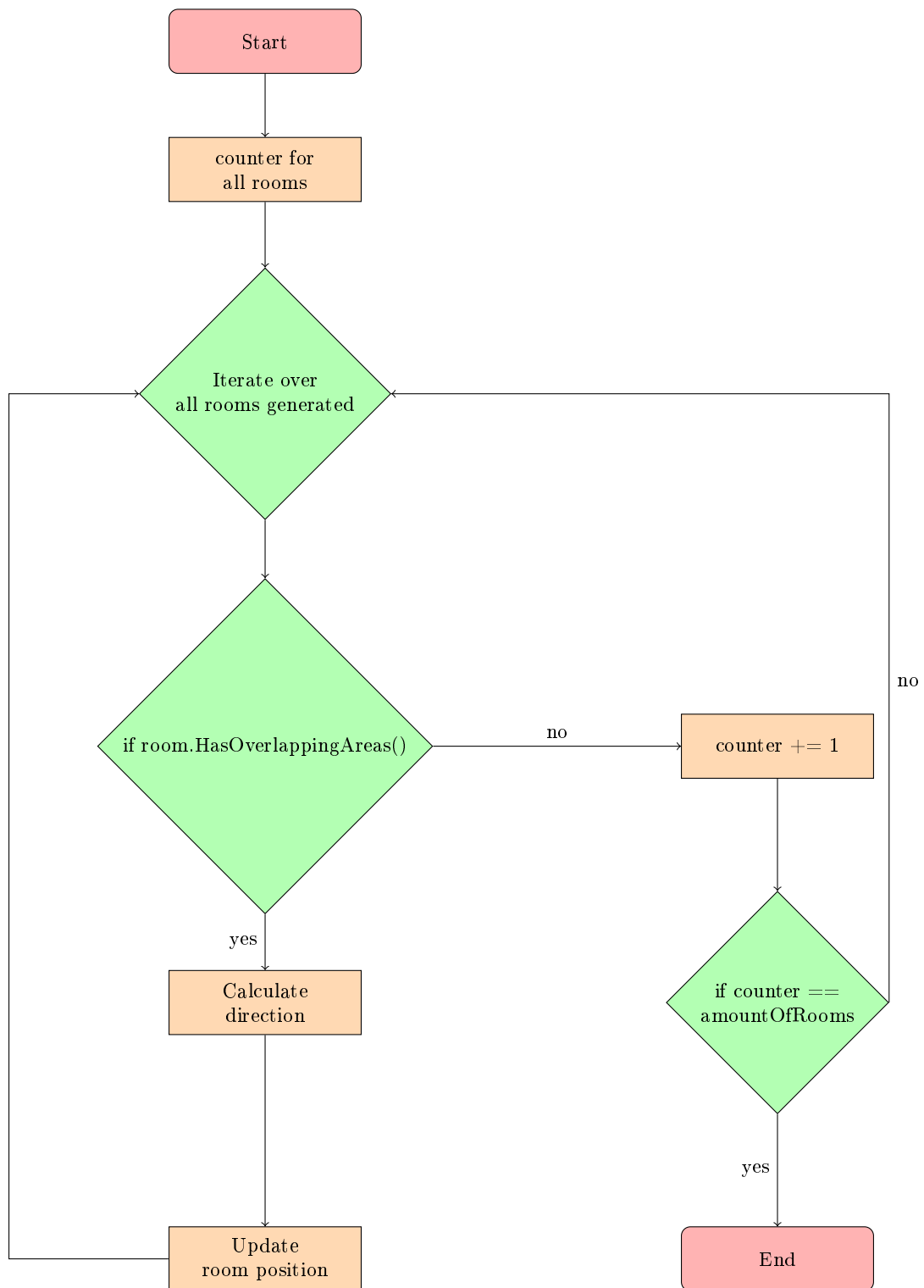


Figure 14: Flowchart for spreading rooms

### Pseudocode for figure 14

```
//this piece of code is called every frame until it develops as solution
checkedRooms = 0
for each room in all_rooms_generated
{
    if room.HasOverlappingAreas()
    {
        direction = Vector2.Zero //sets value to (0,0);
        for each overlappingRoom in room.GetOverlappingAreas()
        {
            displacement = room.Position - overlappingRoom.Position;
            direction += (1/displacement.Length()) *
                displacement.Normalised;
            //takes reciprocal of displacement and adds it to
            direction
        }
        //rounds towards nearest int and multiplies by step, which is a
        variable that scales up with amount of rooms as more rooms
        will need to spread further which optimizes the algorithm
        direction.X = (float)Math.Round(direction.X) * step;
        direction.Y = (float)Math.Round(direction.Y) * step;
        room.Position += direction
    }
    else
    {
        checkedRooms +=1;
    }
}
if (checkedRooms == amountOfRooms)
{
    //each room has been checked and doesn't have overlapping areas so
    algorithm finished
    //move onto next stage of generation
}
```

### Variable table

| Variable Name | Variable Type | Description  |
|---------------|---------------|--|
| count         | Integer       | Counts the rooms that are separated, if it's equal to room count the section is finished |
| displacement  | Vector2       | Average displacement vector of all the overlapping rooms                                 |
| direction     | Vector2       | Manipulated displacement value to suit the program                                       |
| step          | float         | How far the rooms move each iteration  |

### Identifying test data

| Goal Num | Input | Aim                                     | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time | Annoying for player to wait too long for level to generate                     |
| 2        | None  | Room are still close together           | Making the player travel for too long won't be fun                             |
| 3        | None  | Rooms don't overlap                     | Want to rooms to generate correctly, without overlap as it may cause artifacts |

After this process a pre-defined amount of rooms are deleted so that there are gaps in between the rooms, removing 60% of the rooms works well. On top of this the position of each room is multiplied by 2 to spread the rooms further

### 4.5.3 Delete Rooms

The rooms being too close together means that all the corridors will be short and it will be uneventful. Therefore deleting a certain percentage of rooms will make the level more interesting for the player as they will be spread unevenly. To spread the rooms apart further without having to generate more rooms and delete them we can also multiply the position of each room to spread them apart further. This will make the algorithm more efficient because the program doesn't need to generate more rooms to get the same effect.

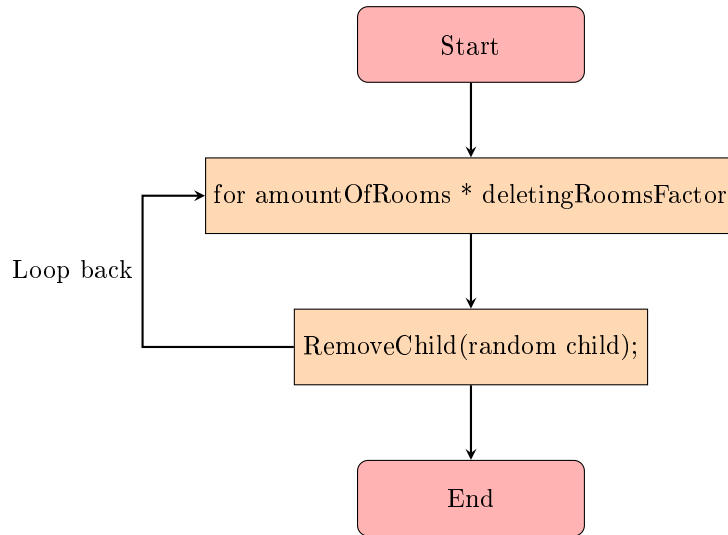


Figure 15: Flowchart for deleting rooms

#### Pseudocode for figure 15

```
for (int)(amountOfRooms * deletingRoomsFactor);  
{  
    RemoveChild(random child);  
}
```

#### Variable table

| Variable Name       | Variable Type | Description  |
|---------------------|---------------|--|
| deletingRoomsFactor | float         | Needs to be between 0-1 (if its 0.8 it will delete 80% of rooms) |

#### Identifying test data

| Goal Num | Input | Aim                                     | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time | Annoying for player to wait too long for level to generate |
| 2        | None  | Distance between rooms varies           | Making the player travel for too long won't be fun         |

### 4.5.4 Delaunay Triangulation

The algorithm used in this section was taken from [Bwa] (Bowyer-Watson algorithm). For the algorithm to function I needed a data structure for points, which link the Area2D, position and a list of points which the point is connected to. Edges are an array of two points which represents a connection between two points. Triangles are an array of three edges and hence three points, from these a circumcircle can be drawn, which is important for the Bowyer-Watson algorithm, this is represented as a vector2 storing the circumcentre and a float for the radius.

| <b>Point</b>  |
|---|
| Public area: Area2D<br>Private position: Vector2<br>Private connectedPoints: List<Point>  |
| Public Constructor(Area2D area)<br>Public Position()<br>returns position<br>Public ConnectedPoint()<br>returns connectedPoints<br>Public AlterPos(Vector2 newPos)<br>updates position<br>Public ConnectPoint(Point newPoint)<br>adds point to connectedPoints |

| <b>Edge</b>  |
|--|
| Private points: Array[2] Point   |
| Public HasPoints(Point point1, Point point2)<br>returns bool weather the edge has those points |

| <b>Triangle</b>   |
|---|
| Private edges: Array[3] Edge<br>Private points: Array[3] Point<br>Private circumcenter: Vector2<br>Private radius: float  |
| Public Constructor(List<Triangle> triangulation,<br>Point point1, Point point2, Point point3)<br>Public IsWithin(Point newPoint)<br>returns a bool if a point lies within a triangles circumcircle<br>Private ContainsEdge(Edge edgeToCompare)<br>returns the edge if it already exists<br>Private FindCircumcentre()<br>updates circumcenter<br>Private FindRadius()<br>updates radius |

Figure 16: Bowyer-Watson Classes

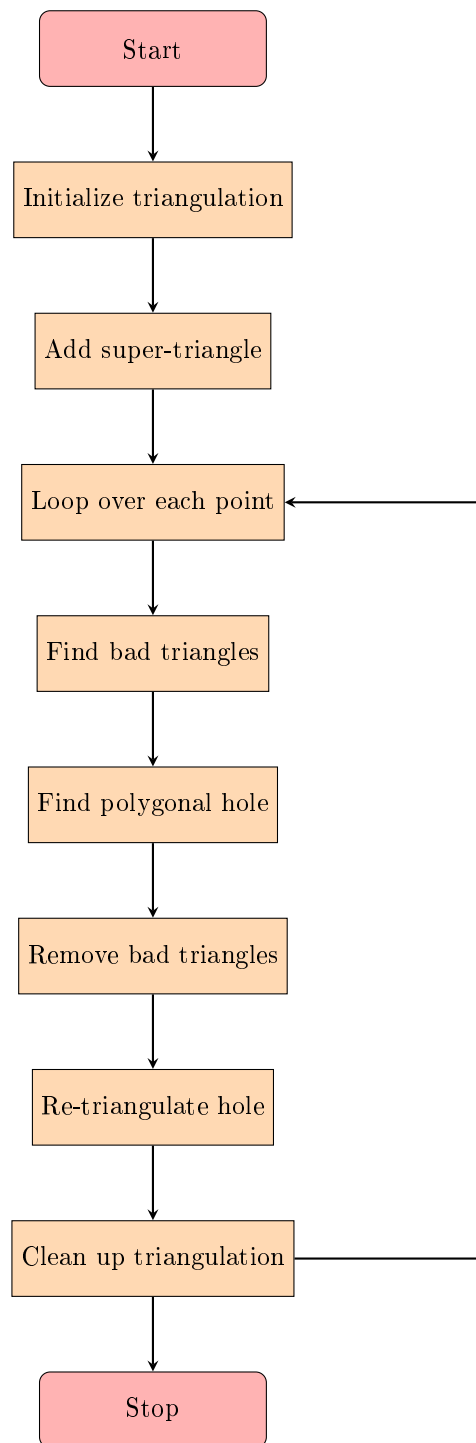


Figure 17: Bowyer-Watson Algorithm Flowchart

### Pseudocode for figure 17

The pseudocode was taken from [Bwa]

```
function BowyerWatson (pointList)
    // pointList is a list of points defining the points to be triangulated
    triangulation := empty list of triangles
    add super-triangle to triangulation // must be large enough to completely
    contain all the points in pointList
    for each point in pointList do // add all the points one at a time to the
    triangulation
        badTriangles := empty list of triangles
        for each triangle in triangulation do // first find all the triangles
        that are no longer valid due to the insertion
            if point is inside circumcircle of triangle
                add triangle to badTriangles
        polygon := list of edges
        for each triangle in badTriangles do // find the boundary of the
        polygonal hole
            for each edge in triangle do
                if edge is not shared by any other triangles in badTriangles
                    add edge to polygon
        for each triangle in badTriangles do // remove them from the data
        structure
            remove triangle from triangulation
        for each edge in polygon do // re-triangulate the polygonal hole
            newTri := form a triangle from edge to the new point
            add newTri to triangulation
    for each triangle in triangulation // done inserting points, now clean up
        if triangle contains a vertex from original super-triangle
            remove triangle from triangulation
    return triangulation
```

### Variable table

| Variable Name   | Variable Type  | Description                                 |
|-----------------|----------------|---|
| triangulation   | List<Triangle> | List of the triangulated mesh               |
| badTriangles    | List<Triangle> | List of triangles to discard                |
| polygon         | List<Edge>     | All edges in end result                     |
| polygonEdgeList | List<Edge>     | List of edges to retriangulate to new point |
| superTriPoint1  | Point          | Point of triangle surrounding all points    |
| superTriPoint2  | Point          | Point of triangle surrounding all points    |
| superTriPoint3  | Point          | Point of triangle surrounding all points    |
| point           | Point          | Point being added to the triangulation      |
| allPoints       | List<Point>    | List of all points to be triangulated       |

### Identifying test data

| Goal Num | Input | Aim   | Justification   |
|----------|-------|---|---|
| 1        | None  | Executes in a reasonable amount of time         | Annoying for player to wait too long for level to generate                          |
| 2        | None  | Every room is included in the triangulation     | This way every room will be included and have a chance of existing in the final map |
| 3        | None  | Triangulated in the most efficient way possible | There is as little as possible overlap when corridors are generated                 |

The pseudo code states that there needs to be triangles, edges and points. These structures can be achieved with object oriented programming as I can make a class with all the relevant attributes and methods for each data structure mentioned.



#### 4.5.5 Remove random amount of edges

Figure ?? shows the result of the algorithm, however this many corridors would be overwhelming for players and would result in each room having 4-5 corridors leading out of it, which would could overwhelm the player. Other rouge-like games usually contain 2-3 corridors leading out of it, this would mean I needed to delete some of the edges but still make sure all the rooms are accessible.

Currently the mesh is stored as a list of edges. So to remove a certain amount I can make a copy of the list and then remove a random amount of edges. However C# passes the list by reference so any alterations on the copy of the list will also alter the original. Therefore I made a new empty list and added a random amount of edges from the original list.

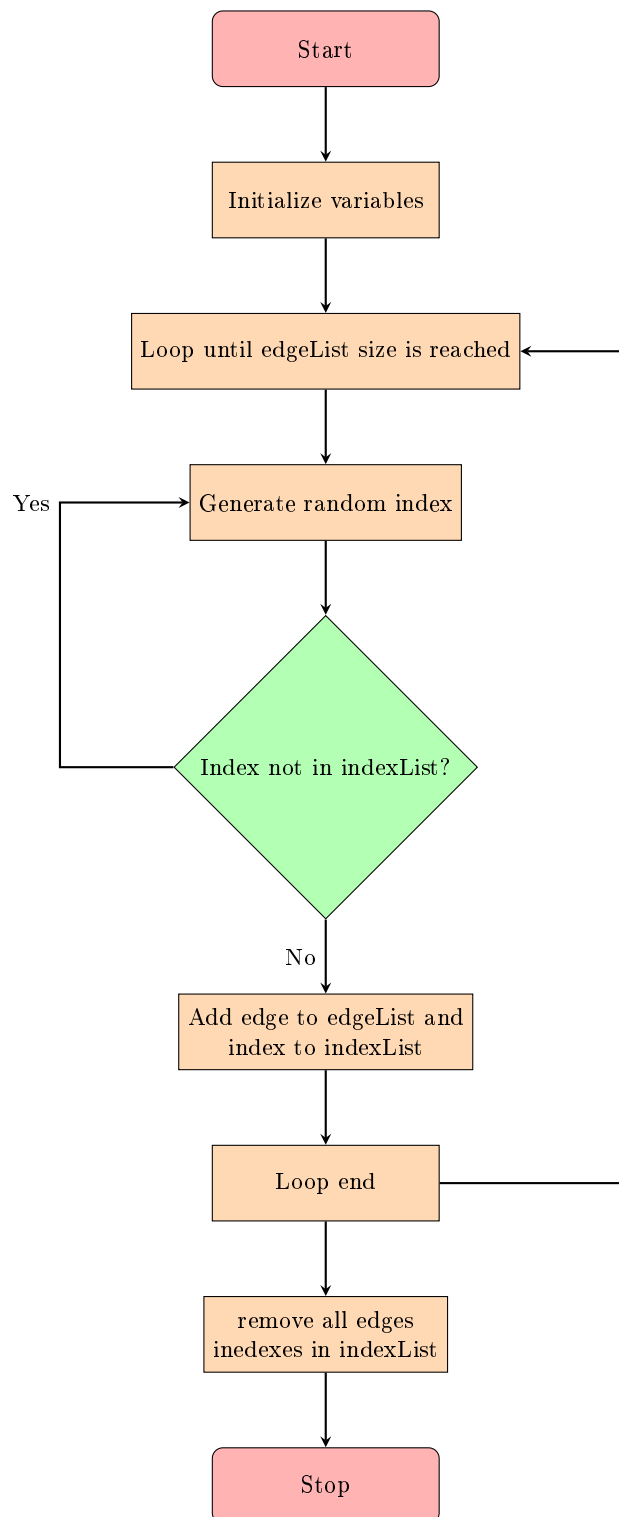


Figure 18: Flowchart for Edge Deletion Algorithm

### Pseudocode for figure 18

```
float edgeDeletingFactor; //number between 0-1 representing how much edges to
    keep
List<Edge> polygon; //List of edges from triangulation stage
List<Edge> edgeList = new List<Edge>();
List<int> indexList = new List<int>();
int newIndex;
while (edgeList.Count() < (int)(polygon.Count()* edgeDeletingFactor))
{
    newIndex = RandomIntBetween(0,polygon.Count())
    //this part ensures there is no duplicates in edgeList
    if (!indexList.Contains(newIndex))
    {
        edgeList.Add(polygon[newIndex]);
        indexList.Add(newIndex);
    }
}
```

### Variable table

| Variable Name      | Variable Type | Description  |
|--------------------|---------------|--|
| edgeDeletingFactor | float         | Needs to be between 0-1 (if its 0.8 it will keep 80% of edges) |
| indexList          | List<int>     | List of indexes to remove from edge list                       |
| newIndex           | int           | New index to be checked if it can be deleted                   |

### Identifying test data

| Goal Num | Input | Aim                                     | Justification   |
|----------|-------|---|---|
| 1        | None  | Executes in a reasonable amount of time | Annoying for player to wait too long for level to generate                      |
| 2        | None  | Random amount of edges are removed      | Every room won't be connected to every other room that's close adding variation |
| 3        | None  | Random edge is selected for removal     | Doesn't bias any specific area when deleting edges                              |

However this creates a problem as now some of the rooms are inaccessible because they have no edges connecting to them or are in another inaccessible section. This is solved in the next section.

#### 4.5.6 Generate spawn and boss room

Before connecting each room to each other we need to have the player a starting position and a goal, this will be done by creating a spawn room where the player starts, this room will always be at the top of the map and force the player to go down. The boss room will always be generated at the bottom of the map since it's the goal for the player to reach. Therefore the player can always go down if they are stuck or unsure as to where to go.

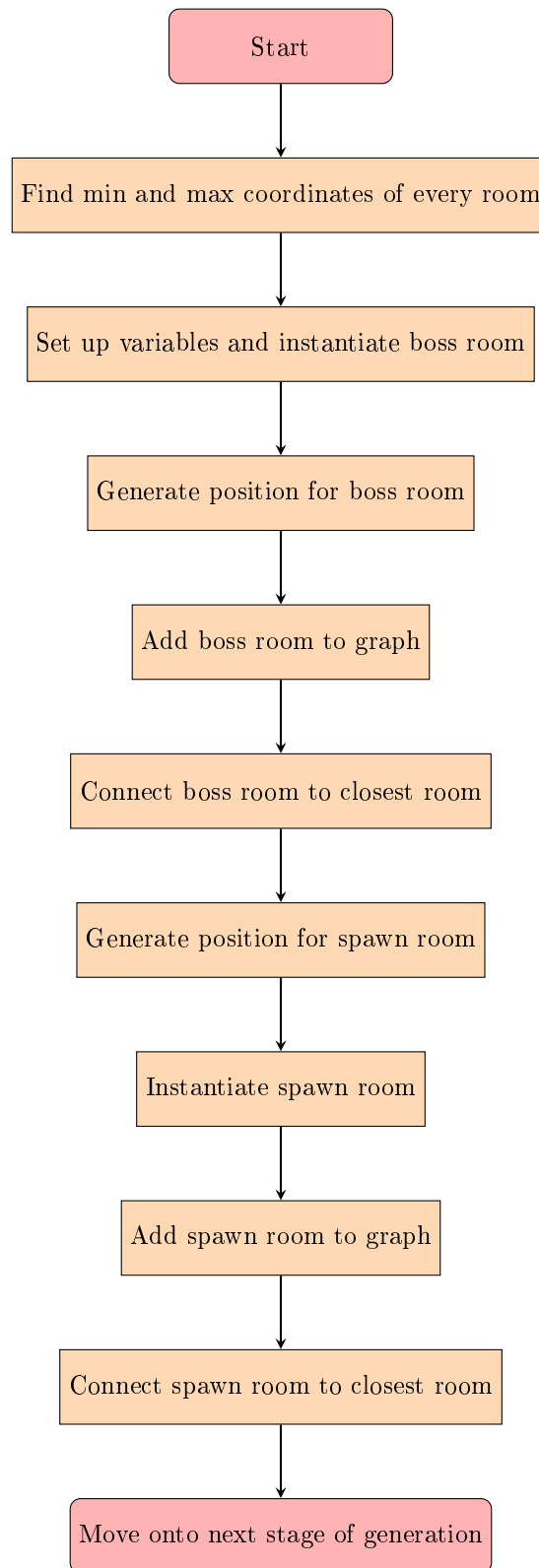


Figure 19: Flowchart for Generating spawn and boss rooms

### Pseudocode for figure 19

```
//polygon is the list of edges where we have to connect new room, its taken
    from the previous section
Find min and max coordinate of every room
//set up variables needed
List<Point> graph = MakeGraph(polygon);
Point closestPoint;
instantiate bossRoom
AddChild(bossRoom);
//label it so that it can be identified
bossRoom.Name = "rmB";
//generate position between min and max X coord but bias for center
//Make Y coord certain amount below other rooms.
bossRoom.Position.X = minCoord.X + rng.RandfRange(0.2f, 0.8f) * (maxCoord.X -
    minCoord.X)
//(0,1) is down in godot so we take maxcoord for y position to place it at the
    bottom
bossRoom.Position.Y = maxCoord.Y + roomDist);
//Make boss room into a point to add it to graph
Point bossPoint = new Point(bossRoom);
graph.Add(bossPoint);
foreach (point in graph)
{
    if (point is closer to bossRoom than current closest known point)
    {
        closestPoint = point;
    }
}
//connect boss to closest room
bossPoint.ConnectPoint(closestPoint);
closestPoint.ConnectPoint(bossPoint);
//Do same thing for spawn room
instantiate spawnRoom
AddChild(spawnRoom);
//label room to make it easy to identify
spawnRoom.Name = "rmS";
//generate position between min and max X coord but bias center more
//Make Y coord certain amount above other rooms.
bossRoom.Position.X = minCoord.X + rng.RandfRange(0.2f, 0.8f) * (maxCoord.X -
    minCoord.X)
//(0,-1) is up in godot so we take mincoord for y position to place it at the
    top
bossRoom.Position.Y = minCoord.Y - roomDist);
//Make spawn room into a point to add it to graph
Point spawnPoint = new Point(spawnRoom);
graph.Add(spawnPoint);
//reset value of closest point
closestPoint = null;
foreach (point in graph)
{
    if (point is closer to spawnRoom than currently closest known point)
    {
        closestPoint = room;
    }
}
//connect spawn to closest room
spawnPoint.ConnectPoint(closestPoint);
closestPoint.ConnectPoint(spawnPoint);

Move onto next stage of generation
```

### Variable table

| Variable Name   | Variable Type | Description   |
|-----------------|---------------|---|
| closestPoint    | Point         | Closest known point to room that is being generated             |
| graph           | List<Point>   | List of all the rooms that have been generated                  |
| currentLength   | float         | distance between closest known point and room being generated   |
| bossRoom        | Node          | The actual boss room  |
| bossPoint       | Point         | bossRoom turned to a point                                      |
| spawnRoom       | Node          | The actual spawn room   |
| spawnPoint      | Point         | spawnRoom turned into a point                                   |
| polygonEdgeList | List<Edge>    | List of all the edges that compose the current state of the map |

### Identifying test data

| Goal Num | Input | Aim  | Justification  |
|----------|-------|--|--|
| 1        | None  | Executes in a reasonable amount of time                            | Annoying for player to wait too long for level to generate                                     |
| 2        | None  | Spawn and room are generated and connected                         | Gives the player a safe room to spawn in   |
| 3        | None  | Spawn is generated above other rooms                               | Player needs to travel down to get to boss room, Gives them a general direction to travel      |
| 4        | None  | Boss room is generated below of other rooms                        | Gives the player a goal and way to progress into the next player                               |
| 5        | None  | Both rooms are generated relatively close to the horizontal center | Having a room generate too far to left or right may limit the rooms that the player can access |

#### 4.5.7 Make sure all rooms are accessible

This section can be decomposed into smaller problems. That being detecting a single section and detecting all sections, then making sure they are connected. First comes detecting sections, this is accomplished more easily when the the graph is represented with list of points along with each point having a list of points it's connected to. Currently the graph is represented as a list of edges. This can be done with a simple algorithm explored further in the "Turning list of edge to graph" section. The detecting sections part is an example of depth first graph traversal since algorithm identifies leaf nodes first.

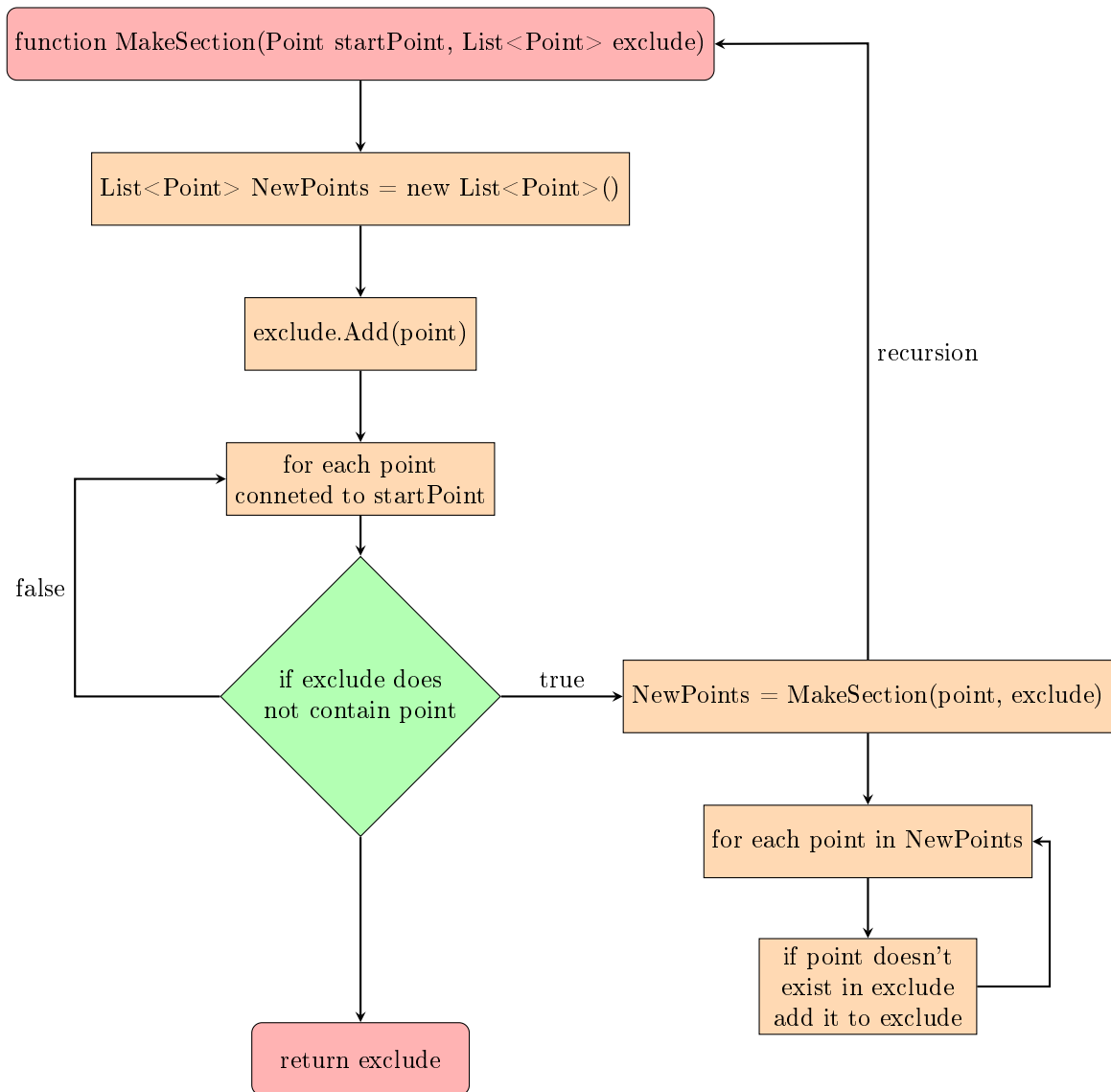


Figure 20: Flowchart for recursive function to find entire section

### Pseudocode for figure 20

*Recursive algorithm for detecting a section, given a starting point*

```
function MakeSection(Point startPoint, List<Point> exclude)
List<Point> NewPoints = new List<Point>()
//add current startPoint to list of points to exclude
exclude.Add(point)
for each point in startPoint.Connectedpoints
{
    //branch out when point isn't already detected and is connected
    if exclude does not contain point
    {
        NewPoints = MakeSection(point, exclude)
        for each newPoint in NewPoints
        {
            if (newPoint doesn't exist in NewPoints)
            {
                exclude.Add(newPoint)
            }
        }
    }
}
return exclude
endfunction
```

### Variable table

| Variable Name | Variable Type | Description  |
|---------------|---------------|--|
| point         | Point         | Starting point for the algorithm to check                |
| exclude       | List<Point>   | List of points to exclude when checking connected points |
| NewPoints     | List<Point>   | Takes result of recursion in the algorithm               |

### Identifying test data

| Goal Num | Input | Aim  | Justification   |
|----------|-------|--|---|
| 1        | None  | Executes in a reasonable amount of time                  | Annoying for player to wait too long for level to generate  |
| 2        | None  | An entire section is detected given a single input point | Can detect all the points that are connected to the start point to make sure that there is one section so that there is one meaning that all rooms are accessible |



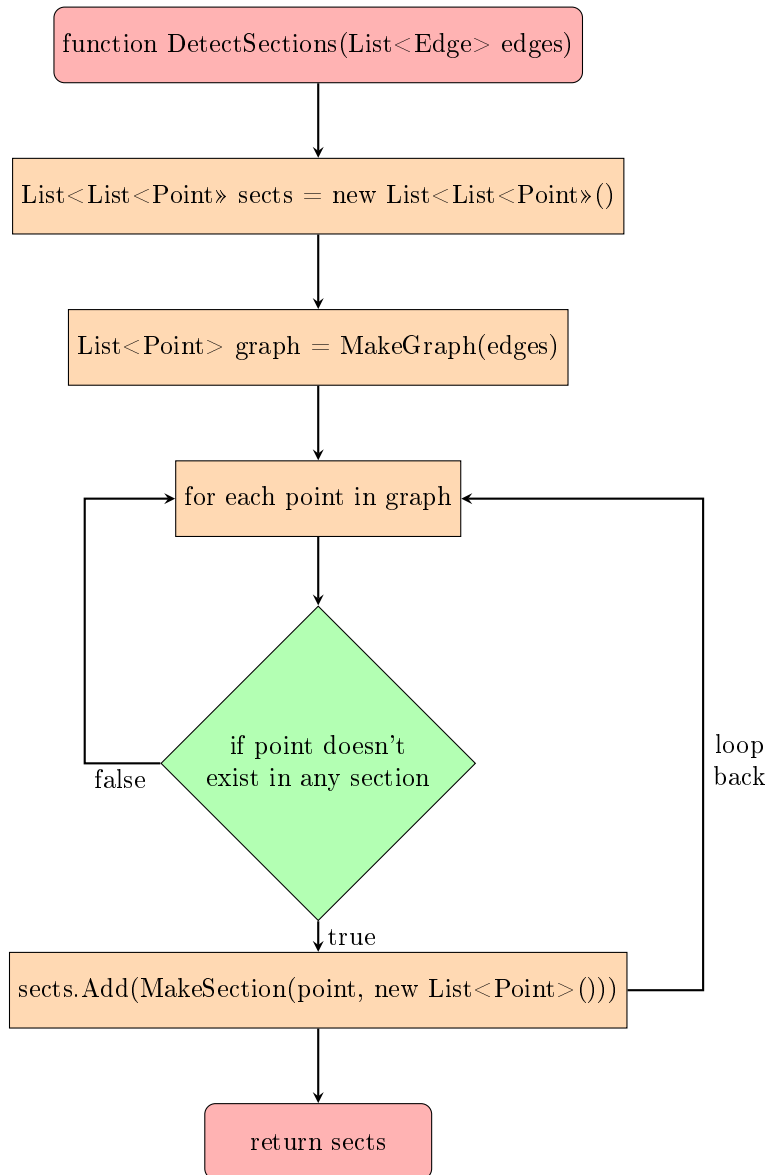


Figure 21: Flowchart for detecting if a new section needs to be made

### Pseudocode for figure 21

*Algorithm for detecting whether a new section needs to be made given a list of edges*

```
function DetectSections(List<Edge> edges)
//list of section, each section is a list of points
List<List<Point>> sects = new List<List<Point>>()
List<Point> graph = MakeGraph(edges)
for each point in graph
{
    check whether point already exists in sects

    if it doesn't exist start a new section with the not existing point as
    starting point
    {
        //add section to list of sections
        sects.Add(MakeSection(point, new List<Point>()))
    }
}
return sects
endfunction
```

### Variable table

| Variable Name | Variable Type     | Description  |
|---------------|-------------------|--|
| sections      | List<List<Point>> | Each List of points is an independent section  |
| graph         | List<Point>       | List of points to be checked   |
| edges         | List<Edge>        | Gets turned into graph but is a parameter of the function since lists of edges are easier to work with |
| exist         | bool              | Acts as a flag set to true if point already exists in a section  |
| NewPoints     | List<Point>       | Takes result of recursion in the algorithm   |

### Identifying test data

| Goal Num | Input | Aim  | Justification  |
|----------|-------|--|--|
| 1        | None  | Executes in a reasonable amount of time          | Annoying for player to wait too long for level to generate |
| 2        | None  | If point has already been checked don't check it | Don't want to detect the wrong amount of sections          |
| 3        | None  | Detect correct amount of sections                | When there is one section all rooms are accessible         |

Now that I am able to detect sections we want to connect them. This is done by saving the list of edges from the triangulation section and finding the difference between the list after removing edges.

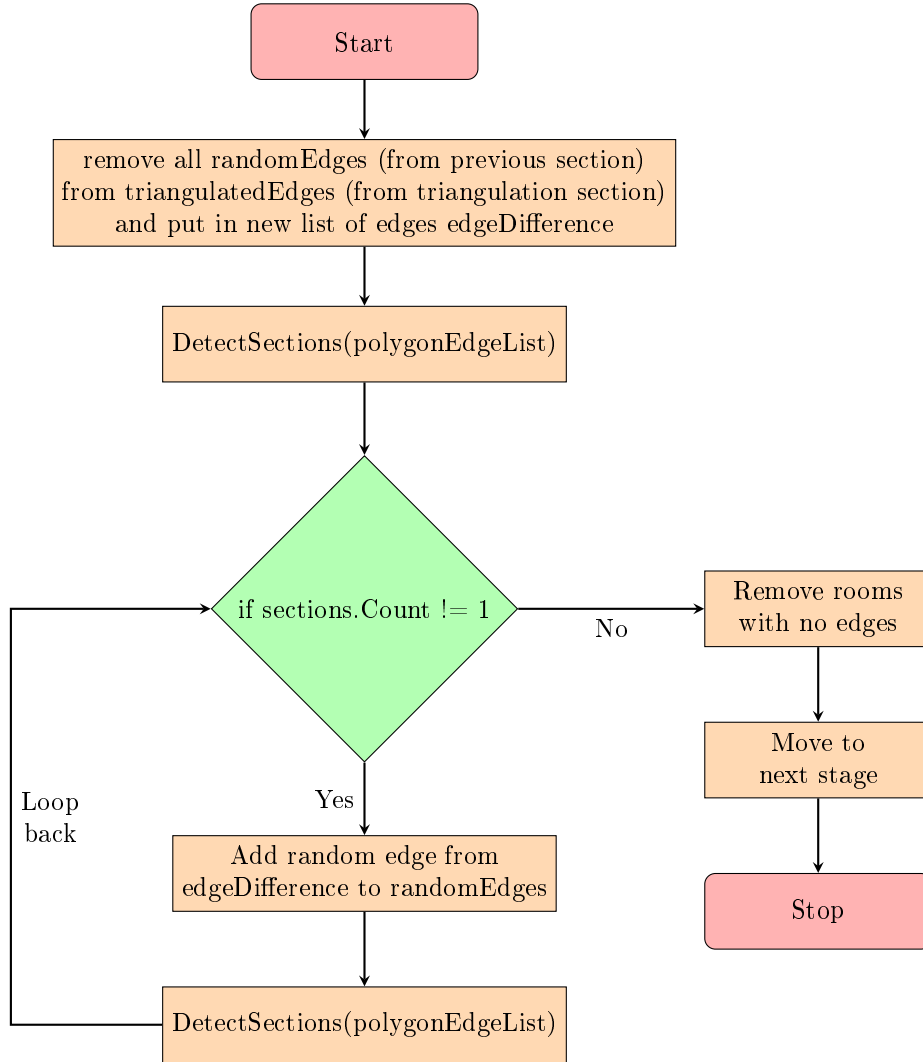


Figure 22: Flowchart for detecting sections in the program

## Pseudocode for figure 22

```
//code called once before main loop is started
//polygon is all the edge after triangulated
//polygonEdgeList are the edges left over after deleting a random amount
//polygonDifference is polygon - polygonEdgeList
List<Edge> polygonDifference(polygon, polygonEdgeList)
List<List<Point>> sections = DetectSections(polygonEdgeList)

//code inside the main loop that runs every frame
if (sections.Count != 1)
{
    //need to add points until there is only one section left
    int index = RandIntInRange(0, polygonDifference.Count - 1)
    //remove random edge from polygonDifference and add it to
    //polygonEdgeList
    polygonEdgeList.Add(polygonDifference[index])
    polygonDifference.RemoveAt(index)
    sections = DetectSections(polygonEdgeList)
}
else
{
    //remove rooms with no edges connected to them
    List<Point> graph = MakeGraph(polygonEdgeList)
    //all rooms are represented with an Area2D data type (custom structure
    //made by godot) and need to check if children(the rooms) of the
    //object are a point in the edgeList
    for each child in GetChildren
    {
        if (a point in graph doesn't have same position as a child)
        {
            //In this case the child isn't considered for in the
            //edgeList
            RemoveChild(child)
        }
    }
    //move onto the next stage of generation
}
```

## Variable table

| Variable Name     | Variable Type | Description   |
|-------------------|---------------|---|
| index             | int           | random number corresponding to index of edge to be added/removed            |
| polygonEdgeList   | List<Edge>    | Edges left over after deleting, taken from previous section                 |
| polygon           | List<Edge>    | All edges after triangulating   |
| polygonDifference | List<Edge>    | Whatever edges are left after polygon - polygonEdgelist                     |
| removeThese       | List<Area2D>  | Collection of all rooms to be deleted since they don't have any connections |

## Identifying test data

| Goal Num | Input | Aim                                     | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time | Annoying for player to wait too long for level to generate |
| 2        | None  | All rooms are connected to each other   | So the player can access all rooms                         |
| 3        | None  | Delete all rooms without connections    | removes all rooms that are inaccessible                    |

#### 4.5.8 Turn edges into horizontal and vertical

The movement system isn't optimised for edges that are angled in such obscure ways as seen in figure ???. Therefore it's important to change the edges into it's vertical and horizontal components to make the level more traversable for the player. Originally I was going to create an advanced network to connect all the points in the most efficient way possible, but looking at how long it took me to implement delaunay triangulation I decided against it as it would take too long. So instead I created an algorithm that generates all possibilities to make sure all rooms are accessible.

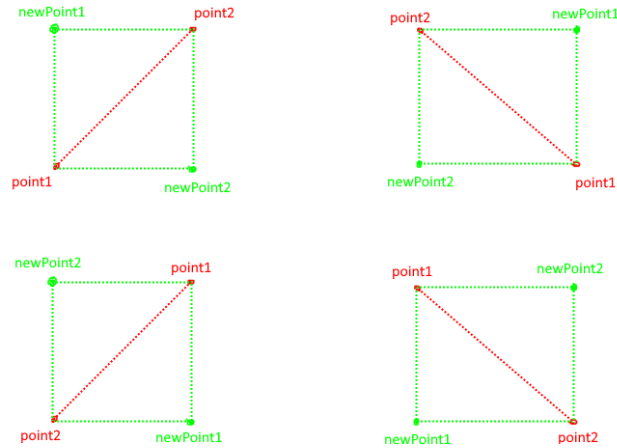


Figure 23: All possibilities for points

The green dotted lines in figure 23 shows where the new edges would be placed when they are replacing the old red edges. This does mean that each edge creates 4 new edges that will likely overlap with other edges, however once this is turned into a tilemap this isn't a problem as all these objects will be deleted.

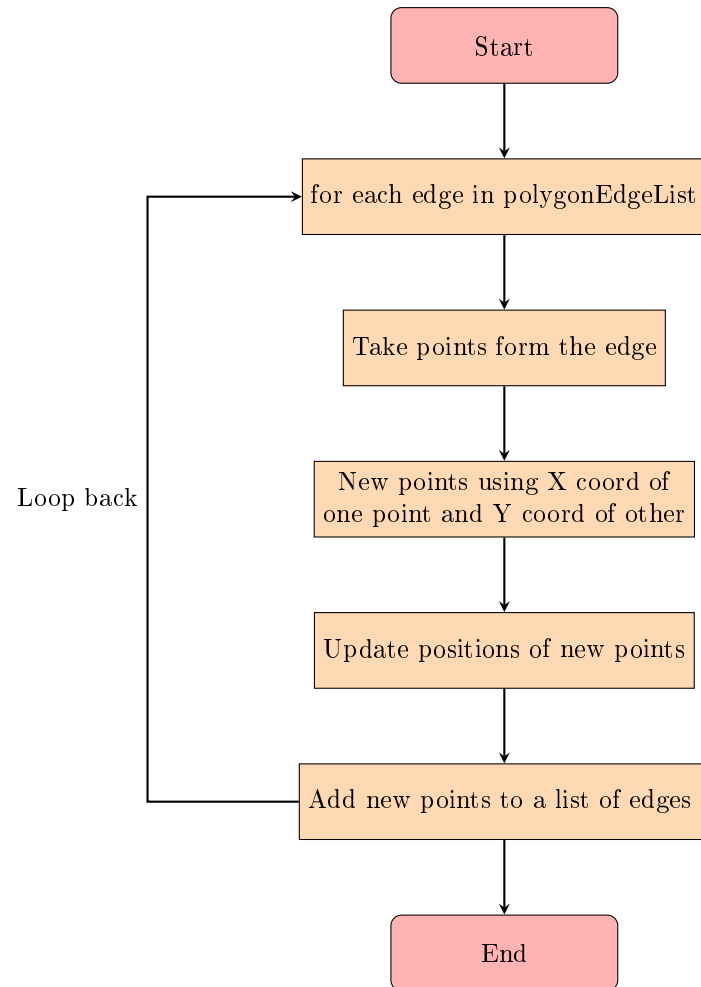


Figure 24: Flowchart for turning edges into horizontal and vertical edges

### Pseudocode for figure 24

```

polygon = new list of edges
//polygonEdgeList is a list of Edges from previous section
for each edge in polygonEdgeList
{
    //take position from points in edge in list
    Point point1 = edge.points[0];
    Point point2 = edge.points[1];
    //new temp point with no values
    Point newPoint1 = new Point(null);
    Point newPoint2 = new Point(null);
    newPoint1.AlterPos(new Vector2(point1.position.X, point2.position.Y));
    newPoint2.AlterPos(new Vector2(point2.position.X, point1.position.Y));
    //all possible paths made and stored in new list of edges (polygon)
    polygon.Add(new Edge(point1, newPoint1));
    polygon.Add(new Edge(point2, newPoint1));
    polygon.Add(new Edge(point1, newPoint2));
    polygon.Add(new Edge(point2, newPoint2));
}

```

### Variable table

| Variable Name   | Variable Type | Description   |
|-----------------|---------------|---|
| polygon         | List<Edge>    | Contains all the new edges generated                        |
| polygonEdgeList | List<Edge>    | List of edges from the previous section                     |
| point1          | Point         | First point of edge being checked                           |
| point2          | Point         | Second point of edges being checked                         |
| newPoint1       | Point         | End point for new edge after being made horizontal/vertical |
| newPoint2       | Point         | End point for new edge after being made horizontal/vertical |

### Identifying test data

| Goal Num | Input | Aim   | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time               | Annoying for player to wait too long for level to generate   |
| 2        | None  | Entire map is made from horizontal and vertical edges | Movement isn't designed for diagonal movement so having only horizontal and vertical movement will make for a better player experience |

In figure ?? the pink lines are the edges before the rooms were generated. The green boxes are the result of this algorithm plus the generate corridors algorithm but it shows the algorithm well so it's included in this section.

#### 4.5.9 Generate corridors

The edges currently have no depth as they are just a connection between two point so we need to convert it into something with an area that the player can interact with. Since we already have the edge we can iterate through the list of edges generated in the previous section and generate a corridor for each edge.

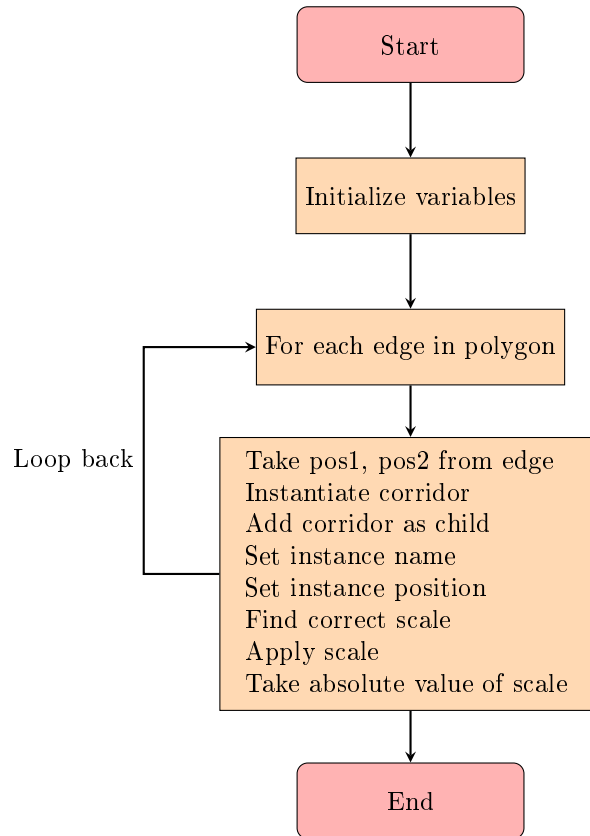


Figure 25: Flowchart for the given pseudocode

### Pseudocode for figure 25

The result of this algorithm can be seen if figure ??

```

// polygon is edge list from previous section
// coridoorRadius is the radius of the coridoor as a vector2
for each edge in polygon
{
    //need to make coridoor between two positions
    Vector2 pos1 = edge.points[0].position;
    Vector2 pos2 = edge.points[1].position;
    Node instance;
    instance = coridoor.Instantiate();
    AddChild(instance);
    //make sure they are tagged so that they can be recognised later
    instance.Name = "cr" + i;
    //place in middle
    instance.Position = (pos1 + pos2) / 2;
    //make sure the radius is added to right part of the room
    //want to add radius to bottom left of one point and top right of other
    Find correct scale and apply to instance
    Take absolute value of x and y of instance scale so it's always positive;
}
  
```

### Variable table

| Variable Name | Variable Type | Description  |
|---------------|---------------|--|
| polygon       | List<Edge>    | list of edges to be turned into corridors              |
| edge          | Edge          | The current edge being turned into a corridor          |
| pos1          | Vector2       | The start position of the corridor                     |
| pos2          | Vector2       | The end position of the corridor                       |
| roomRadius    | Vector2       | How much the corridor corner is offset from the center |
| instance      | Node          | The new corridor that is instantiated                  |



### Identifying test data

| Goal Num | Input | Aim                                       | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time   | Annoying for player to wait too long for level to generate       |
| 2        | None  | Each edge has a corridor generated for it | Corridors allows players to traverse between rooms               |
| 3        | None  | The corridor is an appropriate size       | Needs to be an appropriate size for player to fit into the room. |

However these rooms are still areas that are solid blocks meaning that the player still isn't able to interact with any of the rooms as they are currently all solid blocks. That is why we need to regenerate the entire map to make it playable. A lot of the corridors are overlapping which could be problematic in the future however it wouldn't be worth it to remake that section as it would be very time consuming and wouldn't make any difference to the player since they would never see this section and would hardly effect the following sections.

#### 4.5.10 Turn into tile map

Now that there is a large amount of rooms and corridors there needs to be a way of turning these areas into some kind of environment that the player can interact with. This can be done by turning the map into a tilemap, where the entire map is made of individual square tiles that the player can interact with. This is beneficial as it removes a lot of the detail the areas provide which makes it easier to manage.

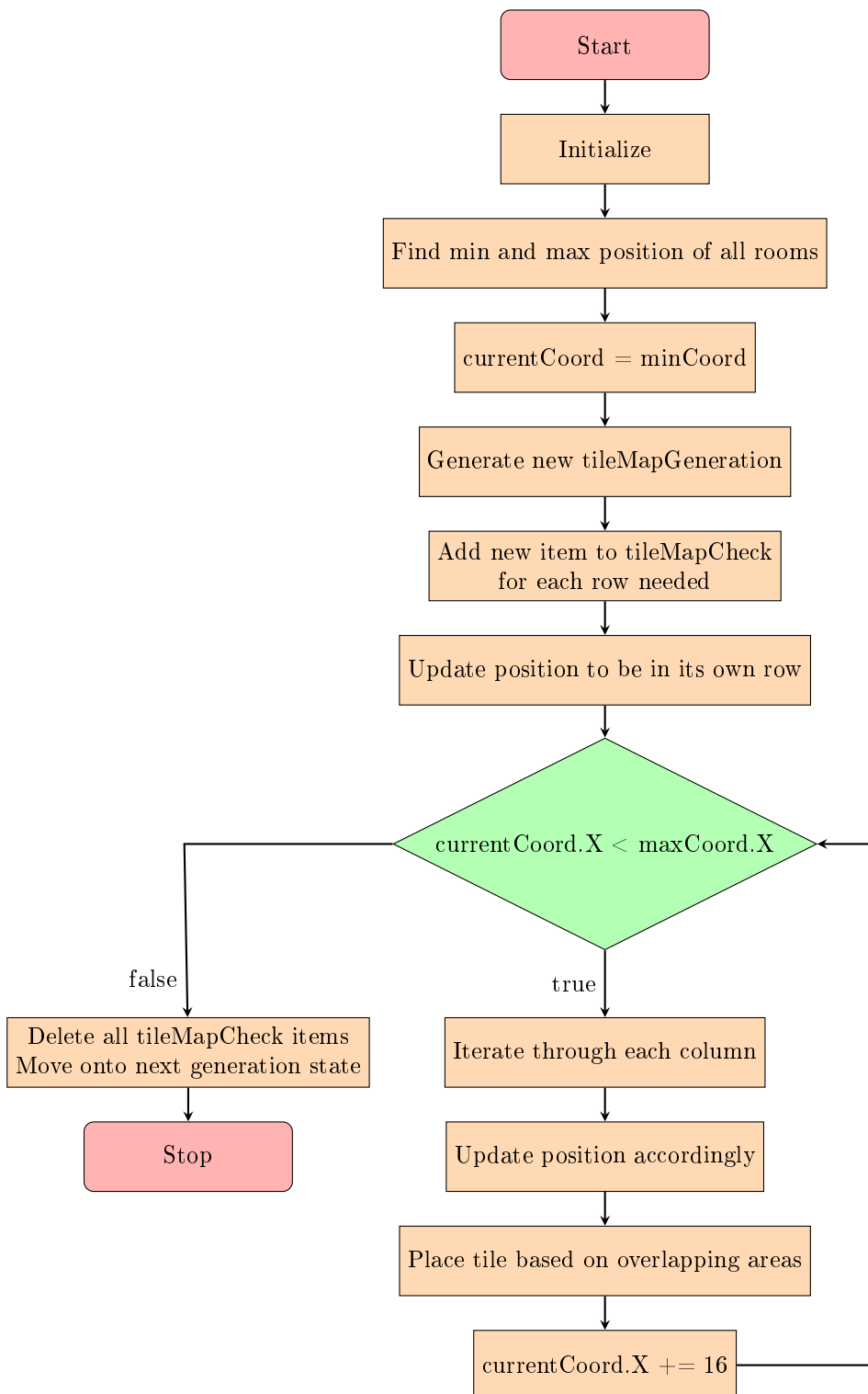


Figure 26: Flowchart for generating the tilemap

## Pseudocode for figure 26

*\*one tile is 16 pixels large*

```
//tileMapGeneration is a class containing properties that help decide what tile
    should be selected
tileMapCheck is a list of tileMapGeneration
Find min and max position of all rooms
currentCoord = minCoord;
//Find how much rows of tilemap are needed to make tilemap
checkAmount = (int)Mathf.Ceil((maxCoord.Y - minCoord.Y) / 16);
Node tempNode;
for (int i = 0; i < checkAmount; i++)
{
    //instantiate new Area2D that will check an area of the map
    tempNode = tileMapCheckScene.Instantiate();
    AddChild(tempNode);
    //update position so it's in it's own row
    tempNode.Position = currentCoord + new Vector2(0f, 16f * i);
    tileMapCheck.Add(tempNode);
}

//this part is ran every frame until next stage is reached
//iterate through each column
if (currentCoord.X < maxCoord.X)
{
    //check each box
    for (int i = 0; i < tileMapCheck.Count(); i++)
    {
        //update position accordingly
        tileMapCheck[i].area.Position = currentCoord + new Vector2(0f, 16f * i);
        place tile based on overlapping areas
    }
    currentCoord.X += 16f;
}
else
{
    delete all tileMapCheck items
    move onto next generation state
}
```

## Variable table

| Variable Name     | Variable Type           | Description   |
|-------------------|-------------------------|---|
| tileMapCheck      | List<TileMapGeneration> | Contains all the areas that check what type of tile belongs in what position          |
| minCoord          | Vector2                 | Minimum coordinate the tilemap will be generated                                      |
| maxCoord          | Vector2                 | Maximum coordinate the tilemap will be generated                                      |
| currentCoord      | Vector2                 | The current coordinate being checked  |
| checkAmount       | integer                 | The amount of tileMapCheck items to be created  |
| tempNode          | Node                    | Instantiates a scene so it can be used in the constructor method of TileMapGeneration |
| tileMapCheckScene | PackedScene             | The scene that is instantiated into tempNode  |
| tilemap           | Tilemap                 | The subject which has tiles generated into it   |

## Identifying test data

| Goal Num | Input | Aim                                     | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time | Annoying for player to wait too long for level to generate                     |
| 2        | None  | Variation in the tiles looks good       | Makes it engaging for the player and more enjoyable                            |
| 3        | None  | Makes interesting map                   | Makes it entertaining for player to explore and actually able to interact with |
| 4        | None  | Makes a traversable map                 | Want the player to be able to move around in the map                           |

### 4.5.11 Spawn player

The only thing left is to let the player explore the map and battle the enemies for themselves. To do this we need to load the player object and place them into the spawn room.

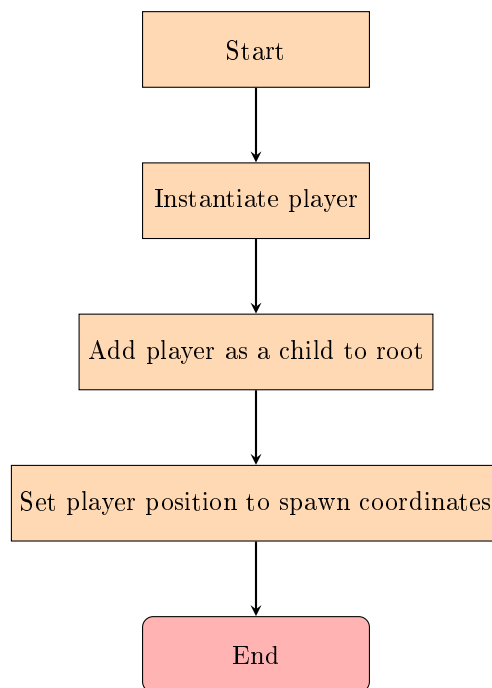


Figure 27: Flowchart for spawning the player

#### Pseudocode for figure 27

```
Instantiate player
//Add child to root rather than temporary node holding all the area rooms
GetNode("/root").AddChild(player);
//Put player in spawn room
player.Position = spawnCoords;
```

#### Variable table

| Variable Name  | Variable Type | Description   |
|----------------|---------------|---|
| characterScene | PackedScene   | The scene which contains the player object                          |
| spawnCoords    | Vector2       | Position where player should be placed, taken from previous section |

## Identifying test data

| Goal Num | Input | Aim                                     | Justification  |
|----------|-------|---|--|
| 1        | None  | Executes in a reasonable amount of time | Annoying for player to wait too long for level to generate |
| 2        | None  | Player is spawned in spawn room         | Player starts in the spawn room                            |
| 3        | None  | Player is able to be controlled         | Player can start the game                                  |

The player is now able to move around and explore the dungeon however we still need to design the player along with all of it's movement.

## 4.6 Player Movement

## 4.7 Player Combat

At this point in design I don't have enough time to create a proper combat system therefore this will be heavily reduced compared to the initial vision for this part.

## 4.8 Enemy Design

## 4.9 Boss Design

## 4.10 UI

# 5 Developing a solution

Since I am using the Godot game engine to create my project I can use many different languages which are supported in Godot such as C, C++, C#, GDScript (made specifically for Godot) and other languages which were implemented by the community, using one of these languages would also mean that there would be less support, since less people would be using it and since it's not official the quality of the documentation may not be of the best quality. Therefore I have decided to use C# to generate my project since I have the most experience with the language due to my experience with unity, however my knowledge is very limited and I would need to adapt my knowledge so I can apply it to Godot.

## 5.1 Dungeon Generation

Each section will need to be completed for the algorithm to be finished, however each section may need multiple frames to be completed so to reliably switch between the generation states, for this to occur I created an enum variable and created an instance of the enum variable which can store one state defined in the enum variable. Below is an example of this in practice.

```
//When initialising variables
private enum generationState
{
    generationState1,
    generationState2,
    generationState3,
    generationState4,
    generationState5,
    finished
}
private generationState currentState = generationState.generationState1;

//Code executing every frame
if (currnetState == generationState.generationState1)
{
    //Code for current section
}
else if (currnetState == generationState.generationState2)
```

```

{
    //Code for current section
}
else if (currnetState == generationState.generationState3)
{
    //Code for current section
}
else if (currnetState == generationState.generationState4)
{
    //Code for current section
}
else if (currnetState == generationState.generationState5)
{
    //Code for current section
}
}

```

This system was implemented later in development but it seems most appropriate in this part as it shows how I am able to switch from one generation state to another. It allows me to switch between states as I please since all I have to do is update the currentState variable. However in my case I just switch from one state to the next since once a section of code is executed it doesn't need to be revisited, if I do need to revisit code I made it into it's own function as it's more convenient.

None of the tests for this section require any input from the user, therefore the input is always none, this is done because if the player had any sort of input they could manipulate the map generation to give themselves some sort of unintended advantage.

#### 5.1.1 Create rooms

Putting the code in the ready function means that it's called one before the game loads, with this all the rooms can be created before the game loads and they will be available to use once the scene is ready. Which is crucial for the program to work.

##### Iteration 1

This iteration achieves goal number 1 and 2, where the algorithm executes within a reasonable amount of time and there is the correct amount of rooms, however there is no variation in the rooms generated so goal number 3 still needs to be developed.

```

// presets for generating rooms
private int amountOfRooms = 100;
private float maxXScale = 20f;
private float minXScale = 8f;
private float maxYScale = 20f;
private float minYScale = 8f;
private int amountOfRooms = 50;

public override void _Ready()
{
    var rng = new RandomNumberGenerator();
    rng.Randomize();
    var room = GD.Load<PackedScene>("res://Scenes/room_generator.tscn");

    for (int i = 0; i < amountOfRooms; i++) //makes a certain amount of "rooms"
    {
        //Make room instance
        var instance = room.Instantiate();
        AddChild(instance);
        //Give random scale and position
        instance.GetNode<CollisionShape2D>("Collision").Scale = new
            Vector2(Map(rng.Randf(),minXScale,maxXScale), Map(rng.Randf(),
                minYScale, maxYScale)); //generates random scale
        instance.GetNode<Area2D>(".").Position = new Vector2(rng.Randf()*10,
            rng.Randf()*10); //generates initial random position
    }
}

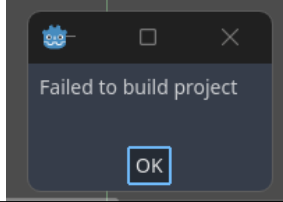
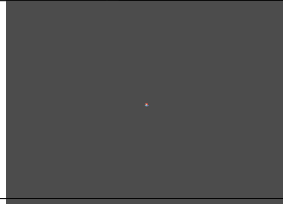
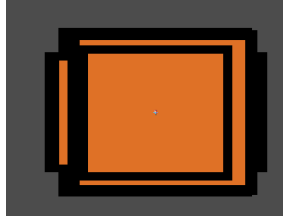
```

```

    }
    SpreadRooms();
}

```

### Iteration 1 Testing

| Test Num | Input | Expected Result   | Actual Result  | Why it happened/What happened   |
|----------|-------|---|--|---|
| 1        | None  | Rooms generated in a dense section around coordinates (0,0) |   | Tried to access Node's position which doesn't exist, need to access Area2D's position |
| 2        | None  | Rooms generated in a dense section around coordinates (0,0) |   | Forgot to assign a parent to the rooms so they didn't generate at all.                |
| 3        | None  | Rooms generated in a dense section around coordinates (0,0) |  | Will be improved upon in the next iteration, but functions currently                  |

### Iteration 2

The last iteration set the position of each room to be a coordinate of two numbers between 0 and 1. This is inefficient because the rooms need to be spread apart so I could just spawn them further apart from each other so they have less distance to travel, this doesn't effect the time to execute for this algorithm but it has a large impact on the time to execute for the next section (Spreading rooms apart).

I also changes the rooms from a cube with random scale to a pre-made room which can be instantiated as needed, however there is only one room variation so there is no progress on goal 3 (creating room variation).

```

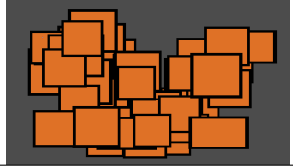
private int amountOfRooms = 25;
public override void _Ready()
{
    room = GD.Load<PackedScene>("res://Scenes/RoomVars/room_var_1.tscn");
    spreadFactor = amountOfRooms * 5f;
    for (int i = 0; i < amountOfRooms; i++) //makes a certain amount of
        "rooms"
    {
        var instance = room.Instantiate();
        AddChild(instance);
        instance.GetNode<Area2D>(".").Position = new Vector2(rng.Randf() *
            spreadFactor, rng.Randf() * spreadFactor); //generates initial
            random position
    }
}

```

### Iteration 2 Testing

### Iteration 3

For this iteration I aimed to add another room variation and because of this I would be able to scale this up for any amount of rooms I plan to add in the future, it also selects a random room variation to instantiate. This progresses goal 3 (room variation), however it still needs to be expanded upon.

| Test Num | Input | Expected Result                      | Actual Result   | Why it happened/What happened            |
|----------|-------|--------------------------------------|---|--|
| 1        | None  | Rooms generated in a less dense area |  | Rooms are generated in a less dense area |

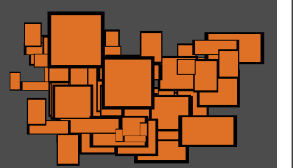
```

private int amountOfRooms = 20;
public override void _Ready()
{
    rng.Seed = 00;
    rng.Randomize();
    room1 = GD.Load<PackedScene>("res://Scenes/RoomVars/room_var_1.tscn");
    room2 = GD.Load<PackedScene>("res://Scenes/RoomVars/room_var_2.tscn");
    spreadFactor = amountOfRooms * 5f;
    int randNum;

    #region make rooms
    // Makes a certain amount of "rooms"
    for (int i = 0; i < amountOfRooms; i++)
    {
        Node instance;
        //pick random room variation
        randNum = rng.RandiRange(0, 1);
        if (randNum == 0)
        {
            instance = room1.Instantiate();
        }
        else if (randNum == 1)
        {
            instance = room2.Instantiate();
        }
        else
        {
            //would never be called but needs to exist because without it
            //there is a syntax error
            instance = room1.Instantiate();
        }
        AddChild(instance);
        // Generates initial random position
        instance.GetNode<Area2D>(".").Position = new
            Vector2((float)Math.Round(rng.Randf() * spreadFactor),
                (float)Math.Round(rng.Randf() * spreadFactor));
    }
    #endregion
}

```

### Iteration 3 Testing

| Test Num | Input | Expected Result               | Actual Result   | Why it happened/What happened   |
|----------|-------|-------------------------------|---|---|
| 1        | None  | Rooms structure has variation |  | Difficult to tell but there is more variation in the structure of the rooms |



#### Iteration 4

For this iteration I aimed to add more room variations and make it even easier to add more variations in the future if needed, since one of the stakeholders has said that they would like there to be a variation in the rooms after showing them the variation of the previous iteration.

I used naming convention to add all the scenes to a list and then instantiate a room from that list. This achieves goal 3 and all the goals are now achieved and the algorithm can be considered complete, however there is also lots of room for expansion in the future if necessary.

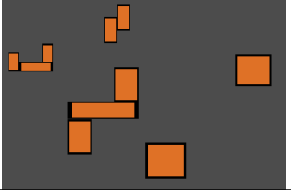
```
private int amountOfRooms = 50;
private Vector2 scaleRange = new Vector2(1f, 1.5f); //min and max of scalse of
rooms
private int roomVars = 10;
public override void _Ready()
{
    rng.Seed = 00;
    rng.Randomize();
    float spreadFactor = amountOfRooms * 10f;
    int randNum;
    float scaleRand;

    #region make rooms
    //Load all the scenes with the room variations
    Node instance;
    List<PackedScene> rooms = new List<PackedScene>();
    for (int i = 0; i < roomVars; i++)
    {
        rooms.Add(GD.Load<PackedScene>("res://Scenes/RoomVars/room_var_" + i +
            ".tscn"));
    }
    //Instantiate a random room variation
    for (int i = 0; i < amountOfRooms; i++) // Makes a certain amount of "rooms"
    {
        randNum = rng.RandiRange(0, roomVars - 1);
        instance = rooms[randNum].Instantiate();
        //label room for identification later
        instance.Name = ("rm" + i);
        AddChild(instance);
        //Add some variation to the size of the rooms
        scaleRand = rng.RandfRange(scaleRange.X, scaleRange.Y);
        //flip room variation so that it appears there are more room
        variations
        if (rng.RandiRange(0, 1) == 0)
        {
            scaleRand *= -1;
        }
        // Generates initial random position and scale
        instance.GetNode<Area2D>(".").Scale = new Vector2(scaleRand, scaleRand);
        instance.GetNode<Area2D>(".").Position = new
            Vector2(Mathf.Round(rng.Randf() * spreadFactor * 3f),
                Mathf.Round(rng.Randf() * spreadFactor));
    }
    #endregion
}
```

#### Iteration 4 Testing

##### 5.1.2 Spread rooms

#### Iteration 1

| Test Num | Input | Expected Result                    | Actual Result   | Why it happened/What happened            |
|----------|-------|------------------------------------|---|--|
| 1        | None  | Rooms structure has more variation |  | Easier to tell with less rooms generated |

C# offers an async function which allows me to run the code while other things are happening, by dedicating a thread to this task. This allows the while loop to work since the code is executed until an end condition is met, this will later be replaced by the generation state system which makes the code neater and easier to manage.

Using an async function would mess up my work flow since I am unable to switch between generation stages since once an async function has began it would be difficult to end the task and since I am using methods native to Godot, inside an async method the refresh rate between the two threads may not be aligned properly so checking overlapping areas would have no result until Godot has updated slowing the program down and it would most likely cause problems when other sections are introduced.

Goals 2 and 3 are complete (rooms are still close together and rooms don't overlap), however the code is very disorganised and goal 1 (executes in a reasonable amount of time), which will need to be developed in the future

```
//spreading rooms apart
private bool spread = false;
private int count = 0;
private Vector2 direction = Vector2.Zero;
private float step = 1f;
//temp statistics stuff for testing
private int loopCount = 0;

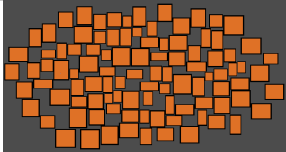
public async void SpreadRooms()
{
    //this only runs once at the start of the game so the inefficiency is
    //loading time
    while (!spread) //if the rooms have overlapping areas
    {
        loopCount += 1;
        step = 2 + (4000/(((0.1f*MathF.Pow(loopCount,2f))+200)));
        await Task.Delay(TimeSpan.FromMilliseconds(1));
        count = 0; //resets count to check overlapping areas of child
        for (int i = 0; i < amountOfRooms; i++)
        {
            if (GetChild<Area2D>(i).HasOverlappingAreas()) //if there are
                overlapping areas on the child
            {
                direction = Vector2.Zero; //resets direction to 0
                for (int j = 0; j <
                    GetChild<Area2D>(i).GetOverlappingAreas().Count; j++)
                //check every overlapping area
                {
                    direction += GetChild<Area2D>(i).Position -
                        GetChild<Area2D>(i).GetOverlappingAreas()[j].Position;
                    //finds difference between original area and overlapping
                    area
                }
                //changes area position
                direction = direction.Normalized();
                GetChild<Area2D>(i).Position += direction * step;
            }
        }
    }
}
```

```

        else
        {
            count += 1; //adds 1 to count if no overlapping areas
        }
    }
    if (count == amountOfRooms)
    {
        spread = true;
        GD.Print("fin");
        GD.Print(loopCount);
    }
}
}

```

### Iteration 1 Testing

| Test Num | Input | Expected Result                            | Actual Result   | Why it happened/What happened  |
|----------|-------|--|---|--|
| 1        | None  | Rooms are spread apart without any overlap |  | Eventually the program separates all rooms, it did take around 40-50 seconds which is too long to be considered successful |

### Iteration 2

I implemented the switching of generation states using the enum variable. That makes the code cleaner and easier to manage.

This iteration moves the room one at a time rather than all of them one bit at a time. However it's inside the update void which updates every frame, which will make it easier for managing sections.

The problem with this is the fact that it uses a while loop, which is bad practice to start every frame. As it sometimes makes the program stop responding when running. Also the room sometimes gets stuck between other rooms as it keeps moving a room in an infinite loop since it's trying to move the rooms one at a time. This iteration still needs to use `thread.sleep` to slow the thread as too much processing happens per frame which causes the program to freeze, this will need to be resolved upon next iteration.

With this iteration none of the goals are achieved so I will have to alter the algorithm to check each room with every iteration rather than considering one room at a time.

```

public override void _Process(double delta)
{
    if (currentState == generationState.spreadRooms)
    {
        for (int i = 0; i < GetChildCount(); i++)
        {
            Thread.Sleep(1);
            while (GetChild<Area2D>(i).HasOverlappingAreas())
            {
                Thread.Sleep(1);
                direction = Vector2.Zero; // Resets direction to 0
                for (int j = 0; j <
                    GetChild<Area2D>(i).GetOverlappingAreas().Count; j++)
                {
                    // Check every overlapping area
                    displacement = GetChild<Area2D>(i).Position -
                        GetChild<Area2D>(i).GetOverlappingAreas()[j].Position;
                    direction += (10 / displacement.Length()) *
                        displacement.Normalized();
                }
            }
        }
    }
}

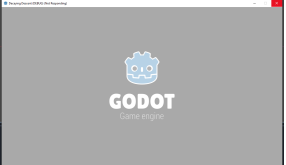
```

```

        // Finds difference between original area and overlapping
        area
    }
    //update position of room
    direction = direction.Normalized();
    GetChild<Area2D>(i).Position += direction * step;
}
}
currentState = generationState.deleteRooms;
}
// Code for other sections (when it exists)
}

```

### Iteration 2 Testing

| Test Num | Input | Expected Result                                   | Actual Result  | What happened/Why it happened   |
|----------|-------|---|--|---|
| 1        | None  | Rooms spread apart faster than previous iteration |  | Rooms get stuck on eachother while trying to spread apart and program can't end while loop causing it to freeze |

### Iteration 3

This iteration removes the need for while loops making it easier to debug as the program doesn't crash if there is an error.

With this goal 1,2 and 3 are all achieved in this iteration (executes in a reasonable amount of time, rooms are still together and none of the rooms overlap)

```

public override void _Process(double delta)
{
    if (currentState == generationState.spreadRooms)
    {
        #region spreading rooms
        count = 0; //resets count to check overlapping areas of child
        for (int i = 0; i < amountOfRooms; i++)
        {
            if (GetChild<Area2D>(i).HasOverlappingAreas()) //if there are
                overlapping areas on the child
            {
                direction = Vector2.Zero; //resets direction to 0
                for (int j = 0; j <
                    GetChild<Area2D>(i).GetOverlappingAreas().Count; j++)
                //check every overlapping area
                {
                    displacement = GetChild<Area2D>(i).Position -
                        GetChild<Area2D>(i).GetOverlappingAreas()[j].Position;
                    direction += (10 / displacement.Length()) *
                        displacement.Normalized(); //finds difference
                        between original area and overlapping area
                }
                // Rounds the result to make the numbers easier to work with
                direction = direction.Normalized() * step;
                direction.X = Mathf.Round(direction.X) * step;
                direction.Y = Mathf.Round(direction.Y) * step;
                GetChild<Area2D>(i).Position += direction;
            }
        }
    }
    else
    {

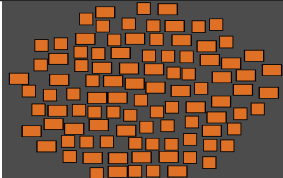
```

```

        count += 1; //adds 1 to count if no overlapping areas
    }
}
if (count == amountOfRooms)
{
    currentState = generationState.deleteRooms;
    return;
    //GD.Print("fin");
    //GD.Print(loopCount);
}
}
// Code for other sections (when it exists)
}

```

### Iteration 3 Testing

| Test Num | Input | Expected Result                              | Actual Result  | What happened/Why it happened  |
|----------|-------|--|--|--|
| 1        | None  | Room spread actually works and doesn't crash |  | After running the program 5 times it didn't crash, previous iteration would freeze every time it's ran so it's more robust |

### Iteration 4

This iteration takes the reciprocal of the difference between the two rooms so that closer rooms have a larger impact on the overall displacement of the room, hopefully speeding up the algorithm and further achieving goal 1 of executing in a reasonable amount of time.

Now all the goals are reached and expanded upon the algorithm can be considered complete. Messing with some of the parameters of the algorithm could also result in speeding up the algorithm.

```

public override void _Process(double delta)
{
    if (currentState == generationState.spreadRooms)
    {
        #region spreading rooms
        count = 0;
        // Resets count to check overlapping areas of child
        for (int i = 0; i < amountOfRooms; i++)
        {
            // If there are overlapping areas on the child
            if (GetChild<Area2D>(i).HasOverlappingAreas())
            {
                direction = Vector2.Zero; // Resets direction to 0
                // Check every overlapping area
                for (int j = 0; j <
                    GetChild<Area2D>(i).GetOverlappingAreas().Count; j++)
                {
                    // Finds difference between original area and overlapping
                    area
                    displacement = GetChild<Area2D>(i).Position -
                        GetChild<Area2D>(i).GetOverlappingAreas()[j].Position;
                    // Take reciprocal so closer rooms have more of an effect
                    direction += (10 / displacement.Length()) *
                        displacement.Normalized();
                }
                direction = direction.Normalized();
                GetChild<Area2D>(i).Position += direction * step * rng.Randf();
            }
        }
    }
}

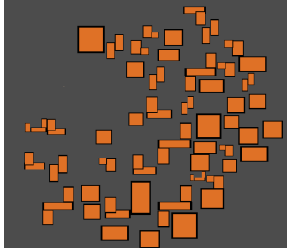
```

```

    }
    else
    {
        count += 1; // Adds 1 to count if no overlapping areas
    }
}
if (count == amountOfRooms)
{
    // Move onto the next stage of generation
    currentState = generationState.deleteRooms;
    return;
}
#endregion
}
// Code for other sections
}

```

#### Iteration 4 Testing

| Test Num | Input | Expected Result           | Actual Result  | What happened/Why it happened  |
|----------|-------|---------------------------|--|--|
| 1        | None  | Rooms spread apart faster |  | Difficult to tell it is actually faster due to random nature however, room variations were implemented and it still functions. |

The algorithm in it's current state allows for development in the future (if needed) since it's well commented, it also achieves it's purpose therefore it's complete.

#### 5.1.3 Delete rooms

##### Iteration 1

Algorithm crashes from time to time which is a critical error since it completely halts the generation process and breaks the program. Therefore none of the goals are achieved

```

//Needs to be between 0-1 (if its 0.8 it will delete 80% of rooms)
private float deletingRoomsFactor = 0.5f;
public override void _Process(double delta)
{
    // Code for other sections

    if (currentState == generationState.deleteRooms)
    {
        #region delete rooms
        for (int i = 0; i < (int)(amountOfRooms * deletingRoomsFactor); i++)
        {
            //generate index of room to delete
            count = rng.RandiRange(0, GetChildCount());
            RemoveChild(GetChild<Area2D>(count));
        }
        currentState = generationState.triangulation;
        #endregion
    }

    // Code for other sections
}

```

## Iteration 1 Testing

| Test Num | Input | Expected Result                   | Actual Result   | What happened/Why it happened   |
|----------|-------|-----------------------------------|---|---|
| 1        | None  | Random amount of rooms is deleted |  | Index out of range error when count = GetChildCount() since there is nothing at that index as the last item of the list is GetChildCount()-1. |

## Iteration 2

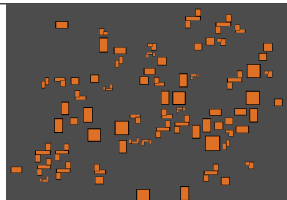
Simple fix to improve last iteration so that now the algorithms achieves all the goals (executing in a reasonable amount of time and distance between rooms varying).

```
//Needs to be between 0-1 (if its 0.8 it will delete 80% of rooms)
private float deletingRoomsFactor = 0.5f;
public override void _Process(double delta)
{
    // Code for other sections

    if (currentState == generationState.deleteRooms)
    {
        #region delete rooms
        for (int i = 0; i < (int)(amountOfRooms * deletingRoomsFactor); i++)
        {
            //generate index of room to delete
            count = rng.RandiRange(0, GetChildCount() - 1);
            RemoveChild(GetChild<Area2D>(count));
        }
        currentState = generationState.triangulation;
        #endregion
    }

    // Code for other sections
}
```

## Iteration 2 Testing

| Test Num | Input | Expected Result                   | Actual Result   | What happened/Why it happened  |
|----------|-------|-----------------------------------|---|--|
| 1        | None  | Random amount of rooms is deleted |  | Rooms are spread appart further making their placement appear more random. |

### 5.1.4 Delaunay triangulation

The classes don't achieve any of the goals single-handedly however they are a requirement for the Boyer Watson algorithm [Bwa] to function.

Testing is not available for any of the classes as their only use is in delaunay triangulation so testing the Bowyer Watson algorithm will reveal any errors in the code of the classes. More code needs to be written to test the classes if I intended to do so, however since all of these were developed in parallel with the Bowyer Watson algorithm they were tested accordingly.

### Point Class iteration 1

The attributes also aren't declared as public or private, by default C# assumes that they are private so all the values aren't able to be updated since there are no methods to access them.

```

class Point
{
    Area2D area;
    Vector2 position;

    public Point(Area2D area)
    {
        //Constructor method for point
        this.area = area;
        //Position is for convenience
        position = area.Position;
    }
}

```

### Point Class iteration 2

Changing the attributes to public allows me to access them from any other class in C#, which may be bad practice as they may be accidentally altered but it performs the function that it needs to do. The AlterPos

method was added so that the position of the point can be altered without actually changing the position of the area that the point represents. This is used for the superTri points in the delaunay triangulation function as they don't need to have an area, so the value is set to null, but their position values are necessary for the algorithm to function.

```

public class Point
{
    public Area2D area;
    public Vector2 position;

    public Point(Area2D area)
    {
        this.area = area;
        position = area.Position;
    }

    public void AlterPos(Vector2 newPos)
    {
        //allows to update position without actually changing position
        //of area
        position = newPos;
    }
}

```

### Point Class iteration 3

To represent the points as a graph I would need to know which other points they are connected to so for this iteration I added a list of points of which the point is connected to, allowing me to represent rooms layout as a directed graph, once the triangulation has completed.

This iteration also adds input validation to the values passed into the instantiation class, so that when an null value is passed instead of a Area2D there is still an position value generated.

```

public class Point
{
    // Point in a graph also represents a room in the code
    public Area2D area;
    public Vector2 position;
    public List<Point> connectedPoints = new List<Point>();

    public Point(Area2D Area)
    {
        area = Area;
    }
}

```



```

        //For super triangle the points don't have area so null is
        passed in
    if (area == null)
    {
        position = new Vector2();
    }
    else
    {
        position = Area.Position;
    }
}

public void AlterPos(Vector2 newPos)
{
    position = newPos;
}

//Want to do input validation on connected points so this method exists
public void ConnectPoint(Point newPoint)
{
    if (newPoint == null || newPoint.position == position ||
        connectedPoints.Any(x => x.position == newPoint.position))
    {
        //If newPoint doesn't exist or is the current point or is
        already connected.
        //Want to know if this is called since if it gets called
        printed something went wrong
        GD.Print("Invalid point");
        return;
    }

    connectedPoints.Add(newPoint);
}
}

```

### Edge Class iteration 1

The edge aims to represent the connection between two points. The midpoint and perpendicular methods are used to help the triangle class to find the circumcentre and circumcircle of the triangle.

```

class Edge
{
    public Point[] points = new Point[2]; // 2 points make an edge

    public Edge(Point point1, Point point2)
    {
        points[0] = point1;
        points[1] = point2;
        // Set the value of the two points
    }

    public Vector2 Midpoint()
    {
        //Finds the midpoint of the two points
        Vector2 mid;
        mid.X = 0.5f * (points[0].position.X + points[1].position.X);
        mid.Y = 0.5f * (points[0].position.Y + points[1].position.Y);
        return mid;
    }

    public float Perpendicular()
    {

```

```

        //finds perpendicular of gradient of line that links the two
        points
        float gradient = (points[0].position.X - points[1].position.X) /
            (points[0].position.Y - points[1].position.Y);
        return (-1f / gradient);
    }
}

```

### Edge Class iteration 2

Checking whether two edges are composed of the same points is problematic since C# checks whether they are the same so comparing two edges may not work for all cases. This iteration adds the HasPoints method which allows me to check if two edges are the same or if an edge is composed of two specific points. This is useful as when comparing two edges, they could be made up of the same points however since they are in different instances they wouldn't be considered as the same edge, for my use case I want to consider them as the same edge, this method allows me to accomplish this

```

public class Edge
{
    public Point[] points = new Point[2]; // 2 points make an edge

    public Edge(Point point1, Point point2)
    {
        points[0] = point1; points[1] = point2; // Set value of the two points
    }

    public Vector2 midpoint() // Finds midpoint of edges
    {
        Vector2 mid;
        mid.X = 0.5f * (points[0].position.X + points[1].position.X);
        mid.Y = 0.5f * (points[0].position.Y + points[1].position.Y);
        return mid;
    }

    public float perpendicular() // Finds perpendicular gradient of the line
        that links the two points
    {
        float gradient = (points[0].position.X - points[1].position.X) /
            (points[0].position.Y - points[1].position.Y);
        return (-1f / gradient);
    }

    public bool HasPoints(Point point1, Point point2) // Checks whether the
        edge is composed of specified points
    {
        if ((points[0] == point1 && points[1] == point2) || (points[0] ==
            point2 && points[1] == point1))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

### Edge Class Iteration 3

Some of the now redundant code is removed, as it was needed for a previous iteration of the triangle class, but no longer has a use.

```

public class Edge

```

```

{
    public Point[] points = new Point[2]; //2 points make a edge
    public Edge (Point point1,Point point2)
    {
        points[0] = point1; points[1] = point2; //set value of the two
        points
    }
    public bool HasPoints(Point point1, Point point2)//checks weather the
    edge is composed of specified points
    {
        if ((points[0] == point1 && points[1] == point2) || (points[0]
        == point2 && points[1] == point1))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

### Triangle Class iteration 1

This class aims to bring together 3 edges and 3 points, so that it can be used in the Bowyer Watson algorithm.

```

class Triangle
{
    public Edge[] edges = new Edge[3]; // 3 edges make a triangle
    public Vector2 circumcentre;
    public float radius;

    public Triangle(Edge edge1, Edge edge2, Edge edge3)
    {
        edges[0] = edge1; edges[1] = edge2; edges[2] = edge3; // Set value of
        all the edges
        FindCircumcentre();
        FindRadius();
    }

    public bool IsWithin(Point newNode)
    {
        // Check whether the new node lies within the circumcircle
        // Doesn't exist yet
        return false;
    }

    private void FindCircumcentre()
    {
        // Find circumcentre of the circumcircle
        // Finds x coordinate of circumcentre
        circumcentre.X = ((edges[1].Midpoint().X * edges[1].Perpendicular()) -
        (edges[2].Midpoint().X * edges[2].Perpendicular()) +
        (edges[2].Midpoint().Y - edges[1].Midpoint().Y)) /
        (edges[1].Perpendicular() - edges[2].Perpendicular());

        // Finds y coordinate of circumcentre
        circumcentre.Y = (edges[2].Perpendicular() * (circumcentre.X -
        edges[2].Midpoint().X)) + edges[2].Midpoint().Y;
        //This doesn't work yet
    }
}

```

```

private void FindRadius()
{
    // Find radius of circumcircle that the edges lie on
    // Pythagorean theorem to find distance between points and centre
    radius = MathF.Sqrt(MathF.Pow((circumcentre.X -
        edges[0].points[0].position.X), 2f) + MathF.Pow((circumcentre.Y -
        edges[0].points[0].position.Y), 2f));
}
}

```

### Triangle Class iteration 2

Equation used in the findCircumcentre method is taken from [Equ]. The method actually works and returns the correct result. Triangulation is also added as a parameter for the constructor method, this aims to reduce the amount of new instances of edges created, which will save memory when running the program as well as reducing the amount of edges I need to work with.

```

public class Triangle
{
    public Edge[] edges = new Edge[3]; // 3 edges make a triangle
    public Point[] points = new Point[3];
    public Vector2 circumcentre;
    public float radius;

    public Triangle(List<Triangle> triangulation, Point point1, Point point2,
        Point point3)
    {
        points[0] = point1; points[1] = point2; points[2] = point3;

        #region edge check
        // Checks whether the edge already exists and if not, it creates a new
        // one
        int edge1index = -1;
        int edge2index = -1;
        int edge3index = -1;

        for (int i = 0; i < triangulation.Count; i++)
        {
            if (triangulation[i].ContainsEdge(new Edge(point1, point2)) != null
                && edge1index == -1)
            {
                edge1index = i;
            }
            if (triangulation[i].ContainsEdge(new Edge(point2, point3)) != null
                && edge2index == -1)
            {
                edge2index = i;
            }
            if (triangulation[i].ContainsEdge(new Edge(point1, point3)) != null)
            {
                edge3index = i;
            }
        }

        if (edge1index != -1)
        {
            edges[0] = triangulation[edge1index].ContainsEdge(new Edge(point1,
                point2));
        }
        else
        {
            edges[0] = new Edge(point1, point2);
        }
    }
}

```

```

    }

    if (edge2index != -1)
    {
        edges[1] = triangulation[edge2index].ContainsEdge(new Edge(point2,
            point3));
    }
    else
    {
        edges[1] = new Edge(point2, point3);
    }

    if (edge3index != -1)
    {
        edges[2] = triangulation[edge3index].ContainsEdge(new Edge(point1,
            point3));
    }
    else
    {
        edges[2] = new Edge(point1, point3);
    }
    #endregion

    findCircumcentre();
    findRadius();
}

public Edge ContainsEdge(Edge edgeToCompare)
{
    Point point1 = edgeToCompare.points[0];
    Point point2 = edgeToCompare.points[1];

    for (int i = 0; i < 3; i++)
    {
        if (edges[i].HasPoints(point1, point2))
        {
            return edges[i];
        }
    }

    return null;
}

public bool isWithin(Point newNode)
{
    // Check whether the new node lies within the circumcircle
    // Pythagorean theorem to check if it lies within
    if (Mathf.Pow(newNode.position.X - circumcentre.X, 2f) +
        Mathf.Pow(newNode.position.Y - circumcentre.Y, 2f) <
        MathF.Pow(radius, 2f))
    {
        return true;
    }
    else
    {
        return false;
    }
}

private void findCircumcentre()
{

```

```

// Find circumcentre of the circumcircle
// Equation from
// https://www.omnicalculator.com/math/circumcenter-of-a-triangle
float t = Mathf.Pow(points[0].position.X, 2f) +
    Mathf.Pow(points[0].position.Y, 2f) -
    Mathf.Pow(points[1].position.X, 2f) -
    Mathf.Pow(points[1].position.Y, 2f);
float u = Mathf.Pow(points[0].position.X, 2f) +
    Mathf.Pow(points[0].position.Y, 2f) -
    Mathf.Pow(points[2].position.X, 2f) -
    Mathf.Pow(points[2].position.Y, 2f);
float J = ((points[0].position.X - points[1].position.X) *
    (points[0].position.Y - points[2].position.Y)) -
    ((points[0].position.X - points[2].position.X) *
    (points[0].position.Y - points[1].position.Y));

circumcentre.X = ((t * (points[0].position.Y - points[2].position.Y)) -
    (u * (points[0].position.Y - points[1].position.Y))) / (2 * J);
circumcentre.Y = ((u * (points[0].position.X - points[1].position.X)) -
    (t * (points[0].position.X - points[2].position.X))) / (2 * J);
}

private void findRadius()
{
    // Find radius of circumcircle that the edges lie on
    // Pythagorean theorem to find distance between points and centre
    radius = Mathf.Sqrt(Mathf.Pow((circumcentre.X -
        edges[0].points[0].position.X), 2f) + Mathf.Pow((circumcentre.Y -
        edges[0].points[0].position.Y), 2f));
}
}

```

### Bowyer Watson Algorithm Iteration 1

This iteration didn't make use of the switching between states, which makes it disorganised. It also contains print statements for debugging so I can identify where errors arise. As the classes were still in development this allowed me to do some sort of testing on them.

```

#region Delaunay Triangulation
// Creates supertriangle that should be large enough to hold all points
GD.Print("Make super triangle");
Point superTriPoint1 = new Point(new Area2D());
superTriPoint1.AlterPos(new Vector2(-999999f, -999999f));
Point superTriPoint2 = new Point(new Area2D());
superTriPoint2.AlterPos(new Vector2(-999999f, 999999f));
Point superTriPoint3 = new Point(new Area2D());
superTriPoint3.AlterPos(new Vector2(999999f, 0f));
Triangle superTri = new Triangle(triangulation, superTriPoint1, superTriPoint2,
    superTriPoint3);
triangulation.Add(superTri);
GD.Print("Start big loop");
for (int i = 0; i < GetChildCount(); i++)
{
    GD.Print("Iteration " + i);
    // Resets bad triangles
    badTriangles = new List<Triangle>();
    GD.Print("Bad triangles list initialized");
    // Makes new point
    Point newPoint = new Point(GetChild<Area2D>(i));
    GD.Print("New point created");
    // Checks which triangles are invalid because of the new point
    GD.Print(triangulation.Count);
}

```

```

for (int y = 0; y < triangulation.Count; y++)
{
    if (triangulation[y].IsWithin(newPoint))
    {
        badTriangles.Add(triangulation[y]);
    }
}
GD.Print("Passed adding bad triangles");
// For remeshing when a new point is added
polygon = new List<Edge>();
polygonEdgeList = new List<Edge>();
for (int y = 0; y < badTriangles.Count; y++)
{
    triangulation.Remove(badTriangles[y]);
    for (int x = 0; x < 3; x++)
    {
        polygonEdgeList.Add(badTriangles[y].edges[x]);
    }
}
for (int y = 0; y < polygonEdgeList.Count; y++) // Checks if there is one
or more items in the list
{
    int count = 0;
    for (int x = 0; x < polygonEdgeList.Count; x++) // Checks if values are
equal
    {
        if (polygonEdgeList[y] == polygonEdgeList[x] && x != y)
        {
            count += 1;
        }
    }
    if (count == 1) // If there is one of an item type, add it to the
polygon
    {
        polygon.Add(polygonEdgeList[y]);
    }
}
// Retriangulate mesh from polygon
for (int y = 0; y < polygon.Count; y++)
{
    triangulation.Add(new Triangle(triangulation, newPoint,
        polygon[y].points[0], polygon[y].points[1])); // Add new triangle to
triangulation
}
}
// Clean up points from superTri
badTriangles = new List<Triangle>();
for (int i = 0; i < triangulation.Count; i++)
{
    for (int y = 0; y < 3; y++)
    {
        if (triangulation[i].points[y] == superTriPoint1 ||
            triangulation[i].points[y] == superTriPoint2 ||
            triangulation[i].points[y] == superTriPoint3)
        {
            badTriangles.Add(triangulation[i]);
        }
    }
}
for (int i = 0; i < badTriangles.Count; i++)
{

```


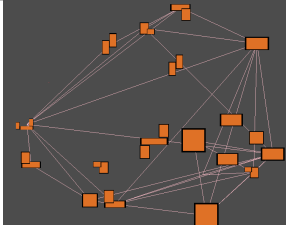
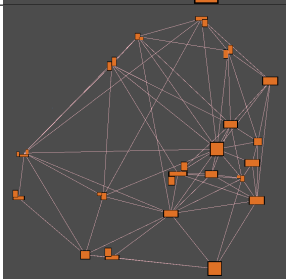
```

        triangulation.Remove(badTriangles[i]);
    }
#endregion

#region Draw Result
GD.Print("Finished");
QueueRedraw();
#endregion

```

### Bowyer Watson Algorithm Iteration 1 Testing

| Test Num | Input | Expected Result                  | Actual Result  | Why it happened/What happened   |
|----------|-------|----------------------------------|--|---|
| 1        | None  | Triangulation between all points |   | Stops responding because a loop isn't closed  |
| 2        | None  | Triangulation between all points |   | Circumcentre wasn't calculated correctly so generation messed up. Will need to alter the find circumcentre method in the triangle class |
| 3        | None  | Triangulation between all points |  | Some of the edges that were meant to be removed when adding a new point aren't removed  |

### Bowyer Watson Algorithm Iteration 2

This iteration adds it to the better way of switching between the generation stages, making the code neater and easier to maintain. It also converts the result from a list of triangles to a list of edges which is easier to work with.

This algorithm accomplishes all the goals defined in the design stage, those being executing within a reasonable amount of time, every room is included in the triangulation and triangulating the rooms in the most efficient way possible.

```

#region Delaunay Triangulation
if (loops == -1)
{
    // Creates supertriangle that should be large enough to hold all points
    superTriPoint1 = new Point(new Area2D());
    superTriPoint1.AlterPos(new Vector2(-999999f, -999999f));
    superTriPoint2 = new Point(new Area2D());
    superTriPoint2.AlterPos(new Vector2(-999999f, 999999f));
    superTriPoint3 = new Point(new Area2D());
    superTriPoint3.AlterPos(new Vector2(999999f, 0f));
    Triangle superTri = new Triangle(triangulation, superTriPoint1,
        superTriPoint2, superTriPoint3);
    triangulation.Add(superTri);
    loops = 0;
}
if (loops < GetChildCount() && loops != -1)

```



```

{
    // Resets bad triangles
    badTriangles = new List<Triangle>();
    // Makes new point
    Point newPoint = new Point(GetChild<Area2D>(loops));
    // Checks which triangles are invalid because of the new point
    for (int y = 0; y < triangulation.Count; y++)
    {
        if (triangulation[y].IsWithin(newPoint))
        {
            badTriangles.Add(triangulation[y]);
        }
    }
    // For remeshing when a new point is added
    polygon = new List<Edge>();
    polygonEdgeList = new List<Edge>();
    for (int y = 0; y < badTriangles.Count; y++)
    {
        triangulation.Remove(badTriangles[y]); // Remove bad triangles from
        triangulation
        for (int x = 0; x < 3; x++)
        {
            polygonEdgeList.Add(badTriangles[y].edges[x]);
        }
    }
    for (int y = 0; y < polygonEdgeList.Count; y++) // Checks if there is one
    or more item in the list
    {
        count = 0;
        for (int x = 0; x < polygonEdgeList.Count; x++) // Checks if values are
        equal
        {
            if (AreTwoEdgesTheSame(polygonEdgeList[x], polygonEdgeList[y]) && x
            != y)
            {
                count += 1;
            }
        }
        if (count == 0) // If there is one of an item type add it to polygon
        {
            polygon.Add(polygonEdgeList[y]);
        }
    }
    // Re-triangulate mesh from polygon
    for (int y = 0; y < polygon.Count; y++)
    {
        triangulation.Add(new Triangle(triangulation, newPoint,
        polygon[y].points[0], polygon[y].points[1])); // Add new triangle to
        triangulation
    }
}
loops += 1;
if (loops == GetChildCount())
{
    // Clean up points from superTri
    badTriangles = new List<Triangle>();
    for (int i = 0; i < triangulation.Count; i++)
    {
        for (int y = 0; y < 3; y++)
        {
            if (triangulation[i].points[y] == superTriPoint1 ||

```


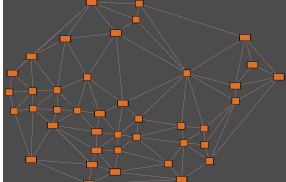

```

        triangulation[i].points[y] == superTriPoint2 ||
        triangulation[i].points[y] == superTriPoint3)
    {
        badTriangles.Add(triangulation[i]);
    }
}
for (int i = 0; i < badTriangles.Count; i++)
{
    triangulation.Remove(badTriangles[i]);
}
// Dumps all edges into polygon for use
polygon = new List<Edge>();

for (int triangle = 0; triangle < triangulation.Count; triangle++)
{
    for (int edges = 0; edges < 3; edges++)
    {
        count = 0;
        for (int poly = 0; poly < polygon.Count; poly++)
        {
            if (AreTwoEdgesTheSame(triangulation[triangle].edges[edges],
                polygon[poly])) // Check if item is already in list
            {
                count += 1;
            }
        }
        if (count == 0) // If item isn't already in list, add it to list
        {
            polygon.Add(triangulation[triangle].edges[edges]);
        }
    }
}
currentState = generationState.draw;
}
#endregion

```

### Bowyer Watson Algorithm Iteration 2 Testing

| Test Num | Input | Expected Result                  | Actual Result   | Why it happened/What happened  |
|----------|-------|----------------------------------|---|--|
| 1        | None  | Triangulation between all points |  | Doesn't remove triangles containing super tri points at the end                              |
| 2        | None  | Triangulation between all points |  | Triangulates all points successfully   |
| 3        | None  | Triangulation between all points |  | Triangulated all points successfully even when there was an abnormally large amount of rooms |

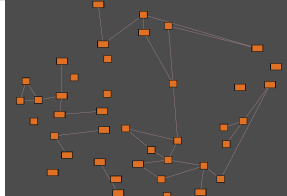
### 5.1.5 Remove random amount of edges

#### Iteration 1

Removes random amount of edges at random indexes. However it permanently removes the edges which means if a rooms is inaccessible there is no way of connecting it back to other rooms. So even though it achieves all the goals set out, it jeopardises the following section.

```
// Removes a certain amount of edges
int polyCount = polygon.Count;
for (int i = 0; i < (int)(edgeDeleteFactor * polyCount); i++)
{
    polygon.RemoveAt(rng.RandiRange(0, polygon.Count - 1));
}
//Stuff to prepare for next section
currentState = generationState.connectSections;
GD.Print("polygon count: " + polygon.Count);
sections = DetectSections(polygon);
```

#### Iteration 1 Testing

| Test Num | Input | Expected Result                   | Actual Result  | What happened/Why it happened   |
|----------|-------|-----------------------------------|--|---|
| 1        | None  | Random amount of edges is removed |  | Random amount of rooms is deleted however it deletes all references of an edge rather than just the instance in the list breaking the rest of the generation. |

#### Iteration 2

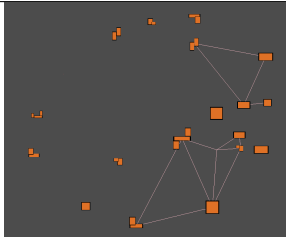
The problem with the previous iteration is that when it removes an edge it removes all instances of that even if they are in other lists of edges, because all copies of that edge are fed by reference, meaning if one reference of that edge are removed all references are removed. So instead of removing a certain amount of edges I will add a certain amount of edges to a new list.

With this iteration goal 1 and 2 are complete so it executes within a reasonable amount of time and removes a random amount of edges, however this still leaves goal 3 where a random edge is selected for removal and currently biases some edges as it takes them in order.

```
// Removes a certain amount of edges
// GD.Print("Edges Removed: "+ (int)(polygon.Count*edgeDeleteFactor));
for (int i = 0; polygonEdgeList.Count < (int)(polygon.Count *
    edgeDeleteFactor); i++)
{
    // Make it so that it takes a random edge and not in order as it has too
    // much structure
    polygonEdgeList.Add(polygon[i]);
    // GD.Print("loop: "+ polygonEdgeList.Count);
}

// Does stuff for the next section
polygonDifference = EdgeDiff(polygon, polygonEdgeList);
sections = DetectSections(polygonEdgeList);
```

#### Iteration 2 Testing

| Test Num | Input | Expected Result                   | Actual Result   | What happened/Why it happened  |
|----------|-------|-----------------------------------|---|--|
| 1        | None  | Random amount of edges is removed |  | The selection of edges isn't random and some sections are biased over others |

### Iteration 3

Now it picks a random edge to add to the list of indexes, before removing it at the end making sure that there are no duplicates removed and a random edge is selected. Meaning that all goals are achieved as the algorithm executes in a reasonable amount of time (goal 1), random amount of edges are removed (goal 2) and a random edge is selected for removal (goal 3).

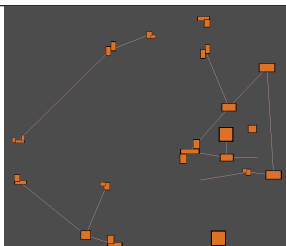
```
// Removes a certain amount of edges
// GD.Print("Edges Removed: "+ (int)(polygon.Count*edgeDeleteFactor));
polygonEdgeList.Clear();
List<int> indexList = new List<int>();
int newIndex;

for (int i = 0; polygonEdgeList.Count() < (int)(polygon.Count *
edgeDeleteFactor); i++)
{
    // takes a random edge and not in order as it has too much structure
    rng.Seed = (ulong)(i);
    newIndex = rng.RandiRange(0, polygon.Count() - 1);

    if (!indexList.Contains(newIndex))
    {
        // GD.Print(newIndex);
        // If the edge doesn't already exist
        polygonEdgeList.Add(polygon[newIndex]);
        indexList.Add(newIndex);
    }
}

currentState = generationState.connectSections;
// Does stuff for the next section
polygonDifference = EdgeDiff(polygon, polygonEdgeList);
sections = DetectSections(polygonEdgeList);
// Polygon is the entire triangulated thing
// PolygonEdgeList is edges left over after deleting
// polygonDifference = polygon - PolygonEdgeList
```

### Iteration 3 Testing

| Test Num | Input | Expected Result                   | Actual Result   | What happened/Why it happened  |
|----------|-------|-----------------------------------|---|--|
| 1        | None  | Random amount of edges is removed |  | Random edge is selected and kept, but it has same effect as deleting a random edge |

### 5.1.6 Make sure all rooms are accessible

This section is broken down to 3 sections like in the design section, branching algorithm from a singular point, detecting a new sections and connecting sections. The algorithm that will run will be the connecting sections algorithm, but it makes use of all the other algorithms to function, therefore testing for these needs to be conducted together, so all testing will be done on the connecting sections algorithm since it will be the easiest to test.

#### Branching algorithm Iteration 1

Currently doesn't work and infinitely loops, then crashes the program with a stack overflow, which doesn't accomplish any of the goals set out.

```
public List<Point> MakeSection(Point point, List<Point> exclude)
{
    List<Point> pts = new List<Point>();
    List<Point> allPoints = new List<Point>();
    bool visited = false;

    //GD.Print("connected points: " + point.connectedPoints.Count);
    //Don't want to make new point if it's already been checked
    for (int i = 0; i < point.connectedPoints.Count; i++)
    {
        exclude.Add(point.connectedPoints[i]);
    }

    for (int i = 0; i < point.connectedPoints.Count; i++)
    {
        //check every connected point
        visited = false;

        for (int j = 0; j < exclude.Count; j++)
        {
            //if it exists in exclude set visited to true
            if (exclude[j].position == point.connectedPoints[i].position)
            {
                visited = true;
                break;
            }
        }

        //if it doesn't exist in exist branch out from that point
        if (!visited)
        {
            pts = MakeSection(point.connectedPoints[i], exclude);
            //put all results of pts into allPoints
            for (int k = 0; k < pts.Count; k++)
            {
                allPoints.Add(pts[k]);
            }
        }
    }
    return allPoints;
}
```

#### Branching algorithm Iteration 2

Don't need all the other variables at the start since the exclude points will be the same as the points visited, so they can be returned, It also accomplishes both of the goals defined being executing within a reasonable amount of time (goal 1) and detecting an entire section given a single input point (goal 2).

```
public List<Point> MakeSection(Point point, List<Point> exclude)
{
    List<Point> newPoints = new List<Point>();
```

```

//add current point to exclude list as it doesn't need to be checked anymore
exclude.Add(point);

for (int i = 0; i < point.connectedPoints.Count; i++)
{
    if (exclude.Where(x => x.position ==
        point.connectedPoints[i].position).Count() == 0)
    {
        // check if connected point already exists in exclude
        // if it doesn't exist then add it to exclude and branch off at new
        point
        newPoints = MakeSection(point.connectedPoints[i], exclude);

        for (int j = 0; j < newPoints.Count; j++)
        {
            if (exclude.Where(x => x.position ==
                newPoints[j].position).Count() == 0)
            {
                //if new point doesn't already exist in exclude add it
                to exclude
                //don't want duplicates
                exclude.Add(newPoints[j]);
            }
        }
    }
}

return exclude;
}

```

### New section detection Iteration 1

Algorithm to detect all sections given a list of edges. Doesn't achieve any of the goals as it doesn't work.

```

public List<List<Point>> DetectSections(List<Edge> edges)
{
    List<List<Point>> sects = new List<List<Point>>(); // We already have a
        variable named 'section', so we shorten it to this
    List<Point> newSection = new List<Point>();

    // Each new item in 'sections' is a new section (stray edges not connected
        to each other)
    GD.Print("Edge count: " + edges.Count);
    List<Point> graph = MakeGraph(edges);
    bool exists = false;

    GD.Print("Graph count: " + graph.Count);
    for (int i = 0; i < graph.Count; i++)
    {
        // Checks whether point exists in 'sects'
        exists = false;
        for (int j = 0; j < sects.Count; j++)
        {
            for (int k = 0; k < sects[j].Count; k++)
            {
                if (graph[i] == sects[j][k])
                {
                    exists = true;
                    break;
                }
            }
        }
        if (exists)
    }
}

```

```

        {
            break;
        }
    }

    // If it doesn't already exist in 'sects'
    if (!exists)
    {
        // Make a section with all the new connected edges
        sects.Add(MakeSection(graph[i], new List<Point>()));
    }
}

return sects;
}

```

### New section detection Iteration 2

Previous iteration compared the actual point to check if it was the same point, however when a point is copied it's sometimes made into a new instance, so I need to compare the positions since they will always be the same if two points are the same, also all the debugging print statements were removed.

This algorithm now accomplishes all of the goals set out, those being executing within a reasonable time frame (goal 1), if a point is already checked don't check it to improve efficiency and function properly (goal 2) and detecting the correct amount of sections (goal 3).

```

public List<List<Point>> DetectSections(List<Edge> edges)
{
    List<List<Point>> sects = new List<List<Point>>(); // We already have a
        variable named 'section', so we shorten it to this
    List<Point> newSection = new List<Point>();

    // Each new item in 'sections' is a new section (stray edges not connected
        to each other)
    // GD.Print("Edge count: " + edges.Count);
    List<Point> graph = MakeGraph(edges);
    bool exists = false;
    // GD.Print("Graph count: " + graph.Count);
    for (int i = 0; i < graph.Count; i++)
    {
        // Checks whether point exists in sects
        exists = false;
        for (int j = 0; j < sects.Count; j++)
        {
            for (int k = 0; k < sects[j].Count; k++)
            {
                if (graph[i].position == sects[j][k].position)
                {
                    // GD.Print("exists");
                    exists = true;
                    break;
                }
            }
            if (exists)
            {
                break;
            }
        }
        // If it doesn't already exist in sects
        if (!exists)
        {
            // Make a section with all the new connected edges

```

```

        sects.Add(MakeSection(graph[i], new List<Point>()));
    }
}
return sects;
}

```

### Connecting sections Iteration 1

Just planned what to do in this iteration so none of the goals are achieved.

```

if (sections.Count != 1)
{
    // add some edges

    // Detect sections
    // sections = DetectSections(polygon);
    // add some edges to connect sections
    GD.Print("sections: " + sections.Count);
}
else
{
    //Move onto next stage
    currentState = generationState.draw;
}

```

There is no testing as no functional code has been produced.

### Connecting sections Iteration 2

Algorithm will add a random edge each time it loops until there is only one section. This means that it accomplishes goal 2, which was making sure all edges are accessible. Goal 1 and 3 are also achieved since the algorithm executes in a reasonable amount of time and rooms without connections are deleted.

```

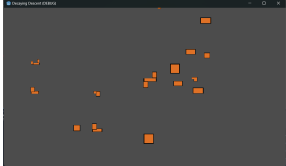
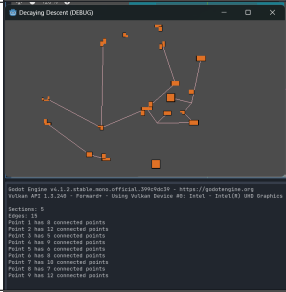
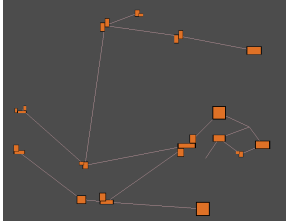
if (sections.Count != 1)
{
    // add some edges
    int index = rng.RandiRange(0, polygonDifference.Count - 1);
    polygonEdgeList.Add(polygonDifference[index]);
    polygonDifference.RemoveAt(index);
    sections = DetectSections(polygonEdgeList);
    // GD.Print("sections: "+sections.Count);
}
else
{
    // there is one sections so can move onto next stage
    // remove rooms with no edges connecting to them
    List<Point> graph = MakeGraph(polygonEdgeList);
    List<Area2D> removeThese = new List<Area2D>();

    // check all children to see if they are a stray point
    for (int i = 0; i < GetChildCount(); i++)
    {
        if (!graph.Where(x => x.position == GetChild<Area2D>(i).Position).Any())
        {
            // add to list of items to remove to not mess up the loop
            // remove children with no points connecting to them
            removeThese.Add(GetChild<Area2D>(i));
        }
    }

    // remove items detected in loop
    for (int i = 0; i < removeThese.Count; i++)
    {
        RemoveChild(removeThese[i]);
    }
}

```



| Test Num | Input | Expected Result                            | Actual Result   | What happened/Why it happened  |
|----------|-------|--|---|--|
| 1        | None  | Adds edges until there is only one section |  | Recursive algorithm end condition didn't work properly so the function kept calling itself until the program crashed.  |
| 2        | None  | Adds edges until there is only one section |  | When turning list of edges into graph the connected points aren't cleared and are added on top of the old ones so their connected to points they aren't meant to be connected to |
| 4        | None  | Adds edges until there is only one section |  | Algorithm adds edges until there is only one section left as intended  |

```

    }
    //move onto next stage
    currentState = generationState.makeCoridoors;
}

```

## Connecting sections Iteration 2 Testing

### 5.1.7 Add spawn and boss room

#### Iteration 1

The first iteration works well and achieves all the goals listed in design process. Those being: executing in a reasonable amount of time (goal 1), spawn room is generated and connected to the rest of the rooms (goal 2), spawn room is generated above all the other rooms (goal 3), boss room is generated below of other rooms (goal 4) and both rooms are generated relatively close to the horizontal center (goal 5)

```

// PolygonEdgeList is where we have to connect new room
FindMinMax(polygonEdgeList);

// Set up variables needed
Point closestPoint = new Point(null);
List<Point> graph = MakeGraph(polygonEdgeList);
float currentLength = 999999f;

// Boss Room Setup
Node bossRoom =
    GD.Load<PackedScene>("res://Scenes/RoomVars/boss.tscn").Instantiate();
AddChild(bossRoom);

// Generate position between min and max X coord but bias center more
// Make Y coord a certain amount below other rooms.
bossRoom.GetNode<Area2D>(".").Position = new Vector2(minCoord.X +
    rng.RandfRange(0.2f, 0.8f) * (maxCoord.X - minCoord.X), maxCoord.Y +
    roomDist);
Vector2 bossRoomPos = bossRoom.GetNode<Area2D>(".").Position;

// Make boss room into a point to add it to the graph

```

```

Point bossPoint = new Point(bossRoom.GetNode<Area2D>("."));
graph.Add(bossPoint);

foreach (Point room in graph)
{
    if (currentLength > (room.position - bossRoomPos).Length() && room !=
        bossPoint)
    {
        // If the room is closer to the currently closest known room
        closestPoint = room;
        currentLength = (room.position - bossRoomPos).Length();
    }
}

GD.Print("Boss distance between: " + currentLength);

// Connect boss to the closest room
bossPoint.ConnectPoint(closestPoint);
closestPoint.ConnectPoint(bossPoint);

// Reset for the next section
currentLength = 999999f;
rng.Randomize();

// Spawn Room Setup
Node spawnRoom =
    GD.Load<PackedScene>("res://Scenes/RoomVars/spawn.tscn").Instantiate();
AddChild(spawnRoom);

// Generate position between min and max X coord but bias center more
// Make Y coord a certain amount above other rooms.
spawnRoom.GetNode<Area2D>(".").Position = new Vector2(minCoord.X +
    rng.RandfRange(0.2f, 0.8f) * (maxCoord.X - minCoord.X), minCoord.Y -
    roomDist);
Vector2 spawnRoomPos = spawnRoom.GetNode<Area2D>(".").Position;

// Make spawn room into a point to add it to the graph
Point spawnPoint = new Point(spawnRoom.GetNode<Area2D>("."));
graph.Add(spawnPoint);

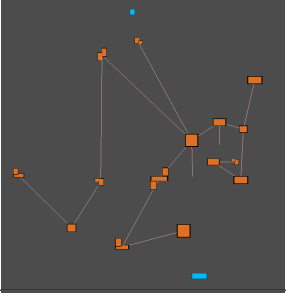
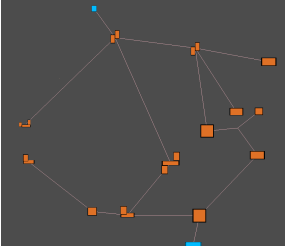
foreach (Point room in graph)
{
    if (currentLength > (room.position - spawnRoomPos).Length() && room !=
        spawnPoint)
    {
        // If the room is closer to the currently closest known room
        closestPoint = room;
        currentLength = (room.position - spawnRoomPos).Length();
    }
}

GD.Print("Spawn distance between: " + currentLength);

// Connect spawn to the closest room
spawnPoint.ConnectPoint(closestPoint);
closestPoint.ConnectPoint(spawnPoint);

// Prepare for the next section
spawnCoords = spawnRoomPos;
polygonEdgeList = MakeEdges(graph);
currentState = generationState.makeCoridoors;

```

| Test Num | Input | Expected Result   | Actual Result   | What happened/Why it happened  |
|----------|-------|---|---|--|
| 1        | None  | Spawn room at top<br>boss room at bottom<br>(rooms represented in blue) |  | Didn't check if the closest point is itself, so it connects itself to itself |
| 3        | None  | Spawn room at top<br>boss room at bottom<br>(rooms represented in blue) |  | Correctly generate above and below respectively                              |

## Iteration 1 Testing

### 5.1.8 Turn edges into horizontal and vertical

#### Iteration 1

Still doesn't generate new objects for the corridor just converts the list of edges into vertical and horizontal. However it doesn't generate an edge if there is an overlap due to a system I tried to implement, this makes some rooms inaccessible so I will need to remake that system or settle with the overlapping edges.

Goal 1 is achieved since it executes in a reasonable amount of time however goal 2 isn't because some parts are being rejected meaning the entire map isn't converted.

```
// Make the corridors horizontal and vertical
polygon.Clear();
List<Point> graph = MakeGraph(polygonEdgeList);

for (int i = 0; i < polygonEdgeList.Count; i++)
{
    // Need to check whether the edges with new points intersect with existing
    // rooms
    Point point1 = polygonEdgeList[i].points[0];
    Point point2 = polygonEdgeList[i].points[1];
    Point newPoint1 = new Point(null);
    Point newPoint2 = new Point(null);
    newPoint1.AlterPos(new Vector2(point1.position.X, point2.position.Y));
    newPoint2.AlterPos(new Vector2(point2.position.X, point1.position.Y));

    // If there is a room in the way, don't make the edge
    // First check where the points are relative to each other and adjust
    // accordingly
    if (point1.position.X <= point2.position.X)
    {
        radius.X = MathF.Abs(radius.X);
    }
    else
    {
        radius.X = -Mathf.Abs(radius.X);
    }

    if (point1.position.Y <= point2.position.Y)
    {

```

```

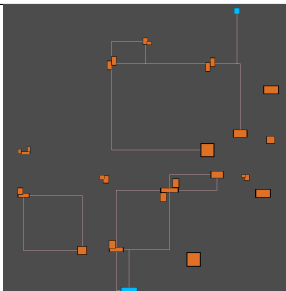
        radius.Y = MathF.Abs(radius.Y);
    }
    else
    {
        radius.Y = -MathF.Abs(radius.Y);
    }

    if ((graph.Where(x => point1.position - radius <= x.position && x.position
        <= newPoint2.position + radius).Count() == 1) ||
        (graph.Where(x => newPoint2.position - radius <= x.position &&
            x.position <= point2.position + radius).Count() == 1))
    {
        //Can generate horizontal first then vertical, this generates
        //vertical first
        //if there isn't a point inside the area where the corridor
        //will exist
        polygon.Add(new Edge(point1, newPoint2));
        polygon.Add(new Edge(point2, newPoint2));
    }

    if (graph.Where(x => point1.position - radius <= x.position && x.position
        <= newPoint1.position + radius).Count() == 1 ||
        graph.Where(x => newPoint1.position - radius <= x.position &&
            x.position <= point2.position + radius).Count() == 1)
    {
        //Can generate horizontal first then vertical, this generates
        //horizontal first
        //if there isn't a point inside the area where the corridor
        polygon.Add(new Edge(point1, newPoint1));
        polygon.Add(new Edge(point2, newPoint1));
    }
}
//Move onto next stage of generation
currentState = generationState.draw;

```

### Iteration 1 Testing

| Test Num | Input | Expected Result  | Actual Result   | Why it happened/What happened |
|----------|-------|--|---|-------------------------------|
| 1        | None  | Horizontal and vertical edges are connecting all the rooms |  | Some rooms are unreachable    |

### Iteration 2

Settled with overlapping corridors and generated all possibilities to ensure that all rooms are accessible. With this the generate corridors section has also been accomplished as both goals are achieved since the algorithm executes in a reasonable amount of time (goal 1) and the entire map is now composed of vertical and horizontal edges. It also generates the corridor object.

```

// Make the corridors horizontal and vertical
polygon.Clear();
coridoorGraph = MakeGraph(polygonEdgeList);

for (int i = 0; i < polygonEdgeList.Count; i++)

```

```

{
    //Create new temporary points for corner of corridor
    Point point1 = polygonEdgeList[i].points[0];
    Point point2 = polygonEdgeList[i].points[1];
    Point newPoint1 = new Point(null);
    Point newPoint2 = new Point(null);
    newPoint1.AlterPos(new Vector2(point1.position.X, point2.position.Y));
    newPoint2.AlterPos(new Vector2(point2.position.X, point1.position.Y));

    // All possible paths made
    polygon.Add(new Edge(point1, newPoint1));
    polygon.Add(new Edge(point2, newPoint1));
    polygon.Add(new Edge(point1, newPoint2));
    polygon.Add(new Edge(point2, newPoint2));
}

PackedScene coridoor =
    GD.Load<PackedScene>("res://Scenes/RoomVars/coridoor.tscn");

for (int i = 0; i < polygon.Count; i++)
{
    // Make area over all the edges
    // Make sure they are tagged so that they can be recognized later
    Vector2 pos1 = polygon[i].points[0].position;
    Vector2 pos2 = polygon[i].points[1].position;
    Node instance;
    instance = coridoor.Instantiate();
    AddChild(instance);
    //tag the corridor so it can be identified later
    instance.Name = "cr" + i;
    //Place corridor at centre
    instance.GetNode<Area2D>(".").Position = (pos1 + pos2) / 2;

    //Scale corridor correctly
    if (pos1 < pos2)
    {
        instance.GetNode<Area2D>(".").Scale = ((pos1 - coridoorRadius) - (pos2
            + coridoorRadius)) / 20;
    }
    else
    {
        instance.GetNode<Area2D>(".").Scale = ((pos1 + coridoorRadius) - (pos2
            - coridoorRadius)) / 20;
    }

    coridoorList.Add(new CoridoorEntry(instance.GetNode<Area2D>("."),
        polygon[i]));
}
//Move onto next stage of generation
currentState = generationState.draw;

```

#### 5.1.9 Turn into tile map

Checks each tile one at a time, is very slow (goal 1 isn't achieved) and doesn't have much variation in tiles placed (goal 2 isn't achieved), this causes the map to be boring so goal 3 is also not met. There is also no way for the player to traverse the vertical corridor so it's not manoeuvrable for the player (goal 4).

#### Iteration 1

```

//preparing for section in previous section
//one tile is 20 units so need to int division it so it aligns with tilemap

```

```

minCoord = new Vector2(minCoord.X - (minCoord.X % 20) - 500, minCoord.Y -
    (minCoord.Y % 20) - 500);
maxCoord = new Vector2(maxCoord.X - (maxCoord.X % 20) + 500, maxCoord.Y -
    (maxCoord.Y % 20) + 500);

Node tempNode = tileMapCheckScene.Instantiate();
AddChild(tempNode);
tileMapCheck = tempNode.GetNode<Area2D>(".");

currentCoord = minCoord;
tileMapCheck.Position = minCoord;

GD.Print("minCoord: " + minCoord);
GD.Print("maxCoord: " + maxCoord);

//Code inside the section that runs every frame
if (currentCoord.X < maxCoord.X)
{
    // GD.Print(currentCoord);
    tileMapCheck.Position = currentCoord;

    if (!tileMapCheck.HasOverlappingAreas())
    {
        //If there is no overlapping areas, essentially background tiles.
        tile.SetCell(0, ToTileCoords(currentCoord), 0, new Vector2I(1, 1));
    }

    // GD.Print("Iterate Coord: " + currentCoord);
    //Move checker one tile across to check next tile
    currentCoord.X += 16;
}
else
{
    //Update position to check layer of tiles
    // GD.Print("Y coord Iterated: " + currentCoord);
    currentCoord.X = minCoord.X;
    currentCoord.Y += 16;
}

if (currentCoord.Y > maxCoord.Y)
{
    //Move onto next stage of generation
    currentState = generationState.draw;
}

```

## Iteration 2

This iteration is a lot faster as there is a box for every row of the tilemap so it iterates through each column and is therefore faster (goal 1 is achieved). Still lacks variation in tiles (goal 2 and 3 aren't achieved). There is still no way for the player to traverse vertical corridors so goal 4 isn't met.

```

// Preparing for section in previous section
//find bottom left and top right of all rooms to find limits of tilemap
FindMinMax();
// One tile is 16 units, so need to int division it so it aligns with tilemap,
// also add some leeway
minCoord = new Vector2(minCoord.X - (minCoord.X % 16) - 500, minCoord.Y -
    (minCoord.Y % 16) - 500);
maxCoord = new Vector2(maxCoord.X - (maxCoord.X % 16) + 500, maxCoord.Y -
    (maxCoord.Y % 16) + 500);
currentCoord = minCoord;
checkAmount = (int)Mathf.Ceil((maxCoord.Y - minCoord.Y) / 16);

```

```

Node tempNode;

for (int i = 0; i < checkAmount; i++)
{
    //make new checking box for every row in the tilemap
    tempNode = tileMapCheckScene.Instantiate();
    AddChild(tempNode);
    tileMapCheck.Add(tempNode.GetNode<Area2D>("."));
    tileMapCheck[i].Position = currentCoord + new Vector2(0f, 16f * i);
}

// Code inside the section that runs every frame
if (currentCoord.X < maxCoord.X)
{
    // Check each box
    for (int i = 0; i < tileMapCheck.Count(); i++)
    {
        // Update position of checking box
        tileMapCheck[i].Position = currentCoord + new Vector2(0f, 16f * i);

        if (!tileMapCheck[i].HasOverlappingAreas())
        {
            // Checks whether tile should exist or not
            tile.SetCell(0, ToTileCoords(tileMapCheck[i].Position), 0, new
                Vector2I(1, 1));
        }
    }
    //movex box to next tile, scans from left to right,
    currentCoord.X += 16f;
}
else
{
    // Remove all the children, as they are no longer needed.
    // move onto next stage of generation
    currentState = generationState.done;

    foreach (Area2D child in tileMapCheck)
    {
        RemoveChild(child);
    }
}
}

```

### Iteration 3

This iteration aims to add more variations to the tiles generated (goal 2 partially achieved), there are also 10 room variations instead of the two that were in the previous iteration. This iteration also adds platforms in rooms that are in predefined locations, they can be different tile types to make the room more interesting to look at (goal 3 is achieved). Also added platforms in vertical corridors to allow the player to traverse upwards in them. This is done with a class to allow me to store attributes that can be used to check previous tiles to decide weather a hole needs to be generated for a platform. Problematic since it generates a hole to the left side of each corridor that the player can fall down (goal 4 is partially achieved).

All of this means that most of the goals are achieved however they are just barely achieved and could still be improved upon to make the game better.

```

// Class to allow for platform generation
public class TileMapGeneration
{
    public Area2D area;
    public int platSize = 0;
    public int holeSize = 0;
}

```

```

public int maxHoleSize;
public int maxPlatSize;

public TileMapGeneration(Area2D newArea)
{
    RandomNumberGenerator rng = new RandomNumberGenerator();
    area = newArea;
    maxHoleSize = rng.RandiRange(2, 3);
    maxPlatSize = rng.RandiRange(4, 6);
}

public void checkTile(Vector2 newCoords)
{
    // Update position of checking box
}

// Ran before the code in section loop from the previous section
FindMinMax(polygon);
currentCoord = minCoord;
checkAmount = (int)Mathf.Ceil((maxCoord.Y - minCoord.Y) / 16);
Node tempNode;

for (int i = 0; i < checkAmount; i++)
{
    tempNode = tileMapCheckScene.Instantiate();
    AddChild(tempNode);
    tempNode.GetNode<Area2D>(".").Position = currentCoord + new Vector2(0f, 16f
        * i);
    tileMapCheck.Add(new TileMapGeneration(tempNode.GetNode<Area2D>(".")));
}

// Inside section loop
// If it hasn't reached the side yet
if (currentCoord.X < maxCoord.X)
{
    // Check each box
    for (int i = 0; i < tileMapCheck.Count(); i++)
    {
        // Room tiles
        tileMapCheck[i].area.Position = currentCoord + new Vector2(0f, 16f * i);

        if (tileMapCheck[i].area.GetOverlappingAreas().Where(x =>
            x.Name.ToString() == "bd").Any() &&
            !tileMapCheck[i].area.GetOverlappingAreas().Where(x =>
                x.Name.ToString().Substring(0, 2) == "cr").Any())
        {
            // If not, then an empty space because corridor leads into it.
            tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
                Vector2I(1, 1));
        }
        else if (tileMapCheck[i].area.GetOverlappingAreas().Where(x =>
            x.Name.ToString() == "pl").Any())
        {
            // Checks whether tile should be a platform
            tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
                Vector2I(4, 4));
        }
        else if (tileMapCheck[i].area.GetOverlappingAreas().Where(x =>
            x.Name.ToString() == "rm").Any())
        {

```



```

        // Do nothing, just need to not do other stuff
        tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
            Vector2I(-1, -1));
    }

    // Corridors
    if (tileMapCheck[i].area.GetOverlappingAreas().Where(x =>
        x.Name.ToString().Substring(0, 2) == "cr").Where(x => x.Scale.X <
        x.Scale.Y).Any() &&
        !tileMapCheck[i].area.GetOverlappingAreas().Where(x =>
            x.Name.ToString().Substring(0, 2) == "rm").Any() &&
        (((tileMapCheck[i].area.Position.Y - 8) / 16) % 5) == 0f)
    {
        // Make platform in vertical corridors
        if (tileMapCheck[i].platSize < tileMapCheck[i].maxPlatSize)
        {
            // Fill with tile
            tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0,
                new Vector2I(1, 1));
            tileMapCheck[i].platSize += 1;
            tileMapCheck[i].holeSize = 99999;
        }
        else if (tileMapCheck[i].holeSize < tileMapCheck[i].maxHoleSize)
        {
            // Make hole
            tileMapCheck[i].platSize = 0;
            tileMapCheck[i].holeSize += 1;
        }
        else
        {
            tileMapCheck[i].holeSize = 0;
        }
    }
    else
    {
        tileMapCheck[i].platSize += 1;
        tileMapCheck[i].holeSize = 0;
    }

    // Checks for empty space
    if (!tileMapCheck[i].area.HasOverlappingAreas())
    {
        // Checks whether tile should exist or not
        tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
            Vector2I(4, 7));
    }
}

currentCoord.X += 16f;
}
else
{
    // Remove all the children
    foreach (TileMapGeneration child in tileMapCheck)
    {
        RemoveChild(child.area);
    }
    currentState = generationState.spawnPlayer;
}

```

#### Iteration 4

Fixed the problem where there is a hole to the left side of every vertical corridor so goal 4 is now achieved there is also some progress on goal 3 since the platforms made in vertical corridors are varying in sizes which was a change made by a stakeholder who pointed out that they didn't like the regularity of the platforms generated previously.

```
// If it hasn't reached the side yet
if (currentCoord.X < maxCoord.X)
{
    // GD.Print(currentCoord);
    // Check each box
    for (int i = 0; i < tileMapCheck.Count(); i++)
    {
        // Room tiles
        tileMapCheck[i].area.Position = currentCoord + new Vector2(0f, 16f * i);
        if (tileMapCheck[i].GOAWhere("bd").Any() &&
            !tileMapCheck[i].GOAWhere("cr").Any())
        {
            // Border of rooms
            tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
                Vector2I(1, 1));
            tileMapCheck[i].lastTile = "room border";
            // If not, then an empty space because corridor leads into it.
        }
        else if (tileMapCheck[i].GOAWhere("pl").Any())
        {
            // Checks whether tile should be a platform
            tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
                Vector2I(4, 4));
            tileMapCheck[i].lastTile = "room platform";
        }
        else if (tileMapCheck[i].GOAWhere("rm").Any())
        {
            // Do nothing, just need to not do other stuff
            tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
                Vector2I(-1, -1));
            tileMapCheck[i].lastTile = "empty";
        }

        // Corridors
        if (tileMapCheck[i].GOAWhere("cr").Where(x => x.Scale.X <
            x.Scale.Y).Any() &&
            !tileMapCheck[i].GOAWhere("rm").Any() &&
            (((tileMapCheck[i].area.Position.Y - 8) / 16) % 5) == 0f)
        {
            // Make platform in vertical corridors
            if (tileMapCheck[i].currentSize > tileMapCheck[i].maxHoleSize &&
                !tileMapCheck[i].genPlat)
            {
                // Switch state to generate platforms and change the size of
                // the platform
                tileMapCheck[i].genPlat = true;
                tileMapCheck[i].maxPlatSize = rng.RandiRange(2, 8);
                tileMapCheck[i].currentSize = 0;
            }
            if (tileMapCheck[i].currentSize > tileMapCheck[i].maxPlatSize &&
                tileMapCheck[i].genPlat)
            {
                // Switch state to generate holes and change the size of the
                // hole
                tileMapCheck[i].maxHoleSize = rng.RandiRange(2, 4);
            }
        }
    }
}
```

```

        tileMapCheck[i].genPlat = false;
        tileMapCheck[i].currentSize = 0;
    }
    if (tileMapCheck[i].currentSize <= tileMapCheck[i].maxPlatSize &&
        tileMapCheck[i].genPlat)
    {
        // Fill with tile
        tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0,
            new Vector2I(1, 1));
        tileMapCheck[i].currentSize += 1;
        tileMapCheck[i].lastTile = "corridor platform";
    }
    if (tileMapCheck[i].currentSize <= tileMapCheck[i].maxHoleSize &&
        !tileMapCheck[i].genPlat)
    {
        // Make hole
        tileMapCheck[i].currentSize += 1;
        tileMapCheck[i].lastTile = "empty";
    }
}
else
{
    // When not making a platform, use a large size so it will generate
    // a platform next time
    tileMapCheck[i].genPlat = false;
    tileMapCheck[i].currentSize = 9999;
}

// Checks for empty space
if (!tileMapCheck[i].area.HasOverlappingAreas())
{
    // Checks whether tile should exist or not
    tile.SetCell(0, ToTileCoords(tileMapCheck[i].area.Position), 0, new
        Vector2I(4, 7));
    tileMapCheck[i].lastTile = "filled";
}
}
currentCoord.X += 16f;
}

```

#### 5.1.10 Spawn player

##### Iteration 1

This iteration spawns the player in a random position (failing goal 2 of spawning player in spawn room) on the map, this is bad because they can spawn in the middle of enemies or close to the boss room, allowing them to essentially skip the level, or immediately get killed by enemies. The camera also doesn't change to the players camera so they aren't able to see the player character.

```

//Random empty tile is selected as spawnCoords from previous tilemap generation
iteration
PackedScene character = GD.Load<PackedScene>("res://Scenes/character.tscn");
Node player;
player = character.Instantiate();
GetNode("/root").AddChild(player);
GD.Print("Spawn Coords: " + spawnCoords);
//move to spawn
player.GetNode<CharacterBody2D>(".").Position = spawnCoords;
currentState = generationState.done;

```

##### Iteration 2

This iteration spawns the player in the spawn room giving them a safe and predictable start achieving goal 2 and the camera switches to the player and the character is no able to be controlled. The algorithm also executes in a reasonable amount of time so goal 1 is achieved.

```
//spawnCoords is position of spawn room
PackedScene character = GD.Load<PackedScene>("res://Scenes/character.tscn");
Node player;
player = character.Instantiate();
GetNode("/root").AddChild(player);

GD.Print("Spawn Coords: " + spawnCoords);
//move to spawn
player.GetNode<CharacterBody2D>(".").Position = spawnCoords;
//Change camera to player's camera
player.GetNode<Camera2D>("./Camera2D").MakeCurrent();
//Finished so can move t
currentState = generationState.done;
```

## References

- [Bwa] *Bowyer–Watson algorithm*. Accessed: 07/12/23. URL: [https://en.wikipedia.org/wiki/Bowyer%E2%80A3Watson\\_algorithm](https://en.wikipedia.org/wiki/Bowyer%E2%80A3Watson_algorithm).
- [Cel] *Celeste on Steam*. Accessed: 25/09/2023. URL: <https://store.steampowered.com/app/504230/Celeste/>.
- [Dis] *Comparing Algorithms for Dispersing Overlapping Rectangles*. Accessed: 05/12/23. URL: <https://mikekling.com/comparing-algorithms-for-dispersing-overlapping-rectangles/>.
- [Equ] *Finding Circumcenter given three points*. Accessed: 18/12/23. URL: <https://www.omnicalculator.com/math/circumcenter-of-a-triangle>.
- [Gen] *Procedurally Generated 3D Dungeons*. Accessed: 05/12/23. URL: <https://www.youtube.com/watch?v=rBY2Dzej03A>.
- [Rws] *Rain World on Steam*. Accessed: 25/09/2023. URL: [https://store.steampowered.com/app/312520/Rain\\_World/](https://store.steampowered.com/app/312520/Rain_World/).
- [Ror] *Risk of Rain 2 on Steam*. Accessed: 25/09/2023. URL: [https://store.steampowered.com/app/632360/Risk\\_of\\_Rain\\_2/](https://store.steampowered.com/app/632360/Risk_of_Rain_2/).
- [Shs] *Steam Hardware & Software Survey: November 2023*. Accessed: 21/12/2023. URL: <https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam>.
- [Spe] *System requirements*. Accessed: 08/02/24. URL: [https://docs.godotengine.org/en/stable/about/system\\_requirements.html#exported-godot-project](https://docs.godotengine.org/en/stable/about/system_requirements.html#exported-godot-project).