Computer Science
Advanced GCE H447
Unit F453

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

Contents

List of Figures

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 of 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

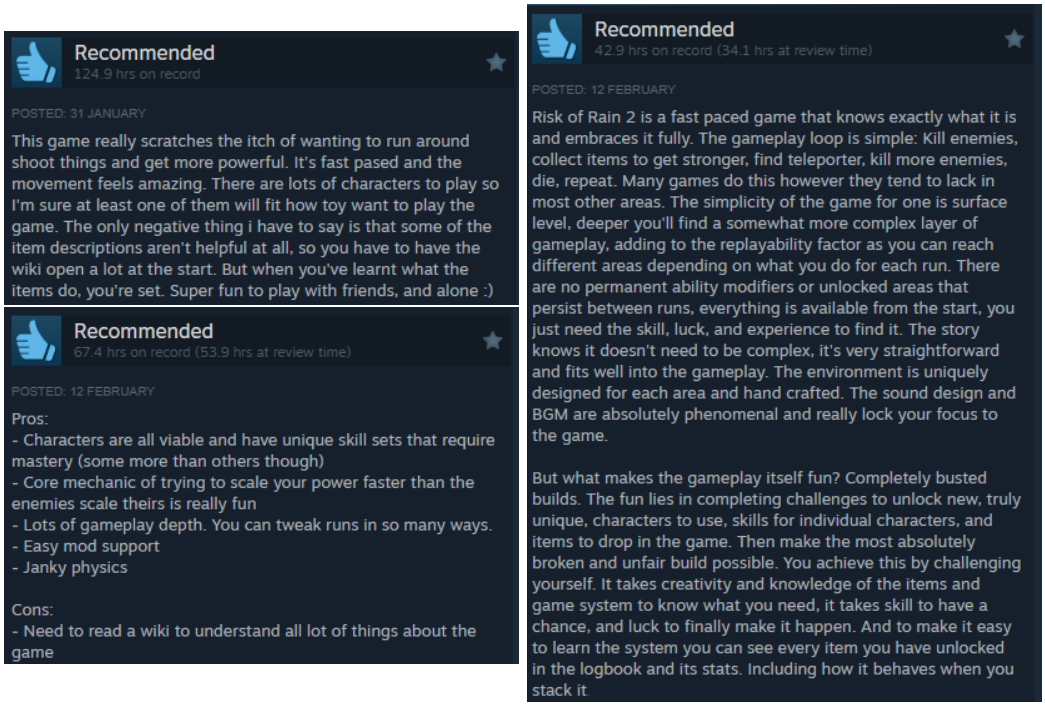


Figure 2: Reviews for Risk of Rain from [Ror]

These reviews show that players enjoy the game for it's 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

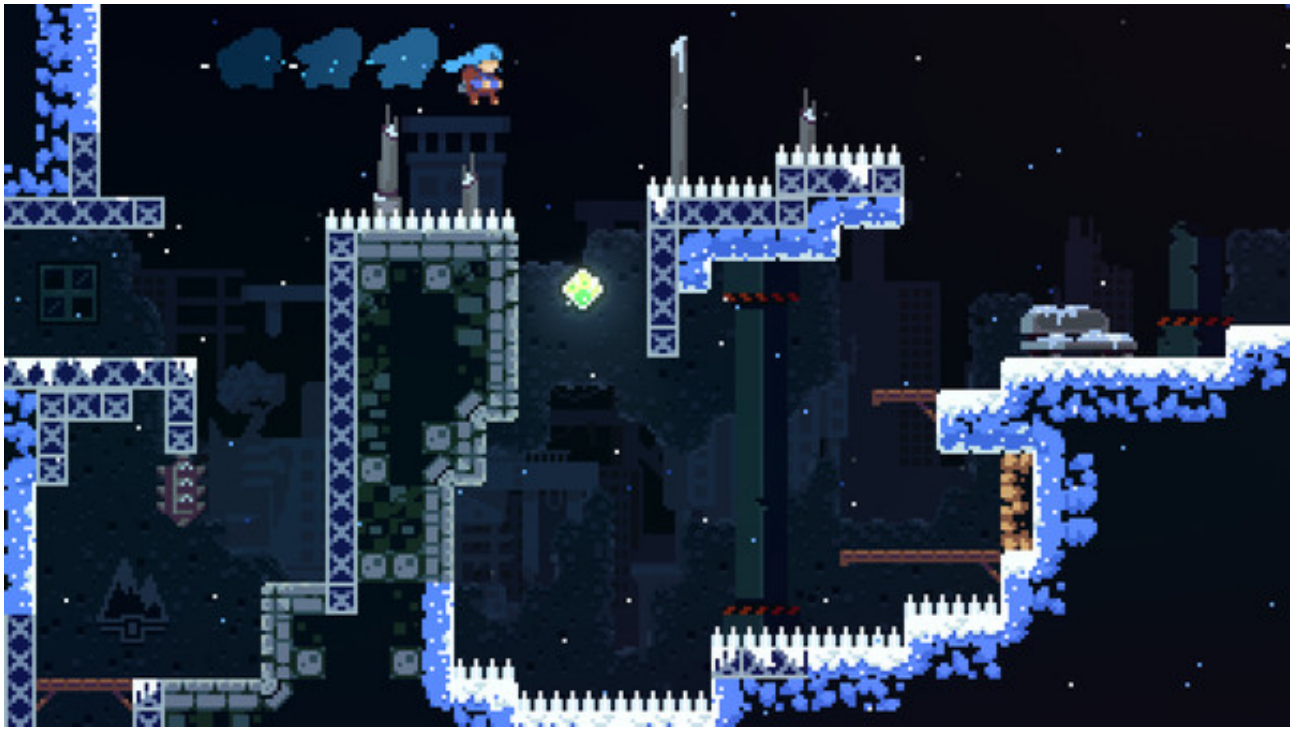


Figure 3: [Cel] Celeste Screen shot

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

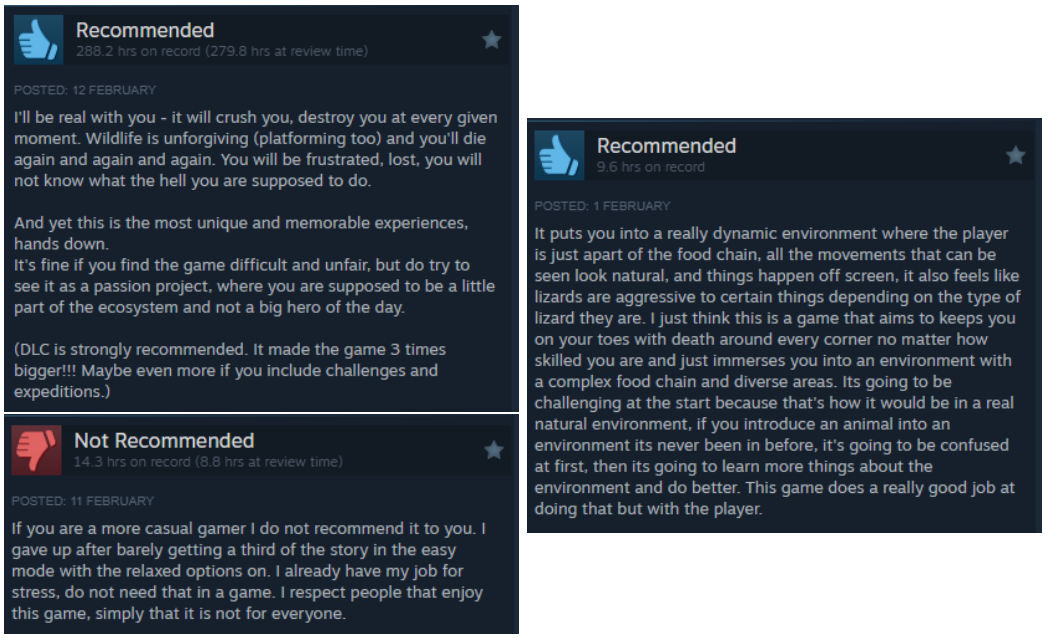


Figure 6: Reviews for Rain World from [Rws]

The community understands that the game is difficult to get into due to it's 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 GDPA

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

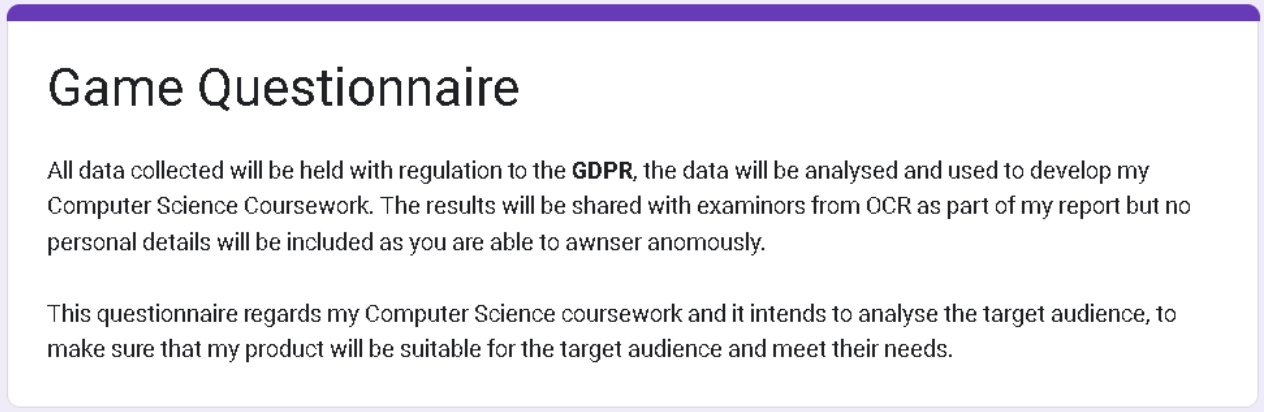


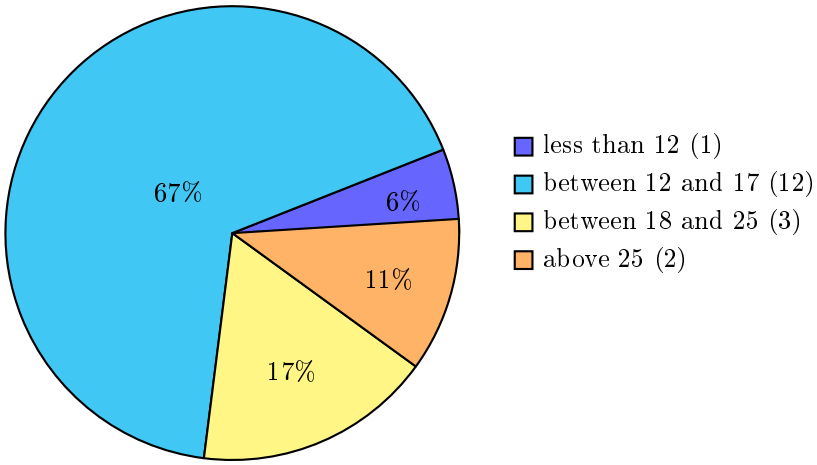
Figure 7: Questionnaire Header

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.

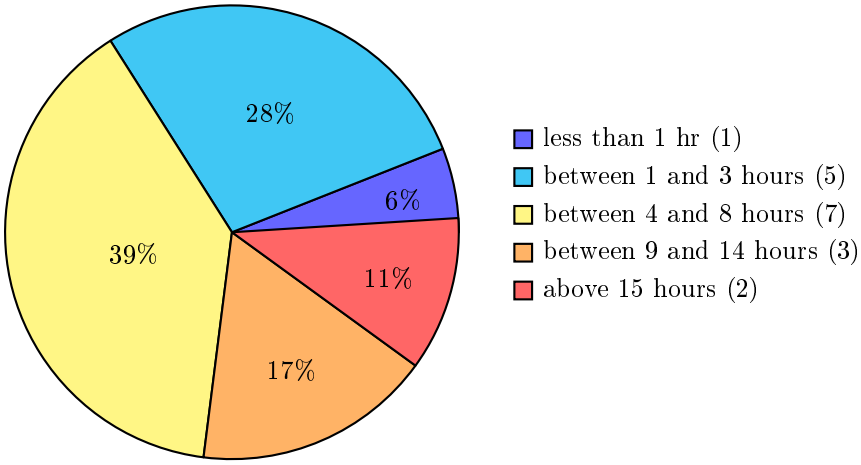
Analysis

Q1: What is your age?



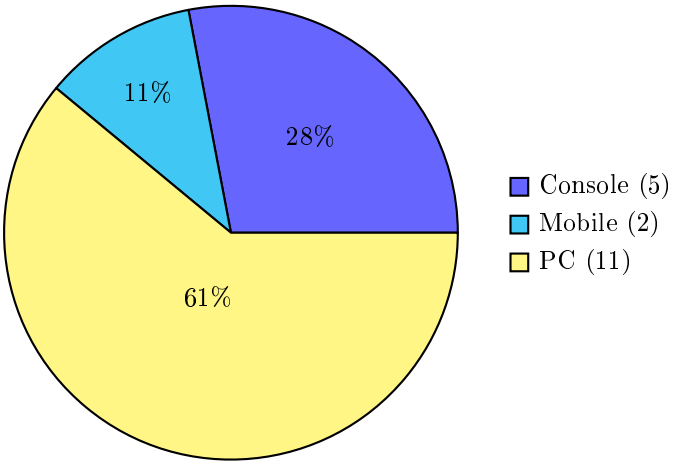
We can see that the target demographic is teens and above meaning that the game can include some violence. It is important to make the game appropriate to the target demographic as they will be the majority of the player base.

Q2: How much do you play games a week?



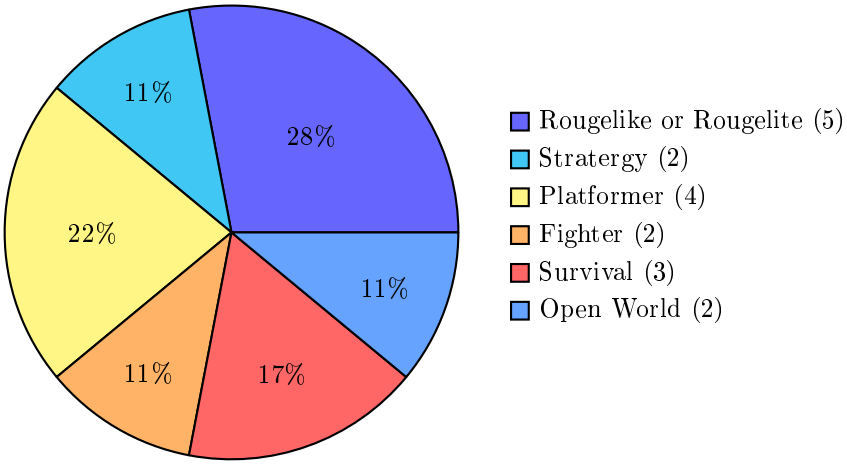
It is important to balance the progress made and the time invested as if people only play 1-2hrs a day as shown by the questionnaire, it is important to make the player feel like they made progress in that time so that they are willing to continue playing. This could be implemented by each run being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

Q3: What platform do you play games on?



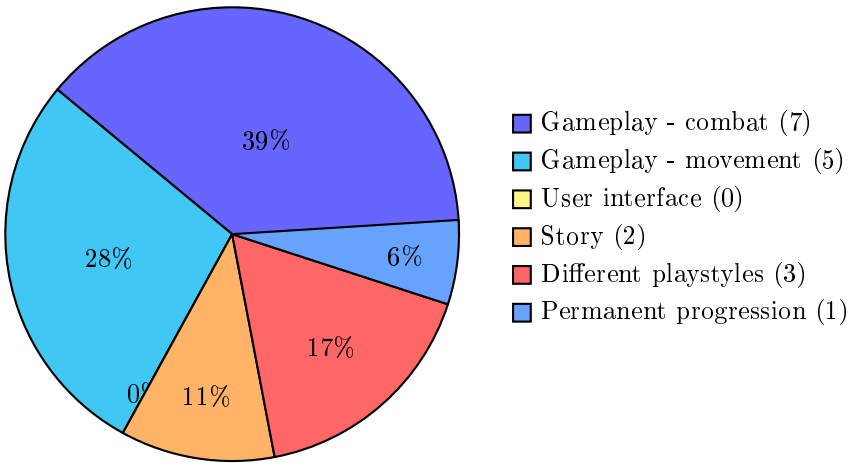
Most people answered replied that they play on pc meaning that it should be ported to pc first. A significant amount of people also play on console so it would be good to port the game to console if possible.

Q4: what genre of games do you play?



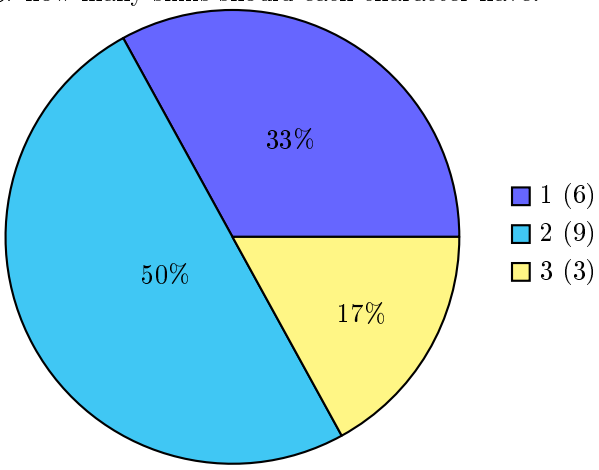
Most people answered Roguelike/Rougelite and platformer meaning I should focus on these two aspects the most when making my game. Survival will be implemented though planning ahead this might be what routes to take and the risk associated with those routes as well as the risk of combat. As well as management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

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



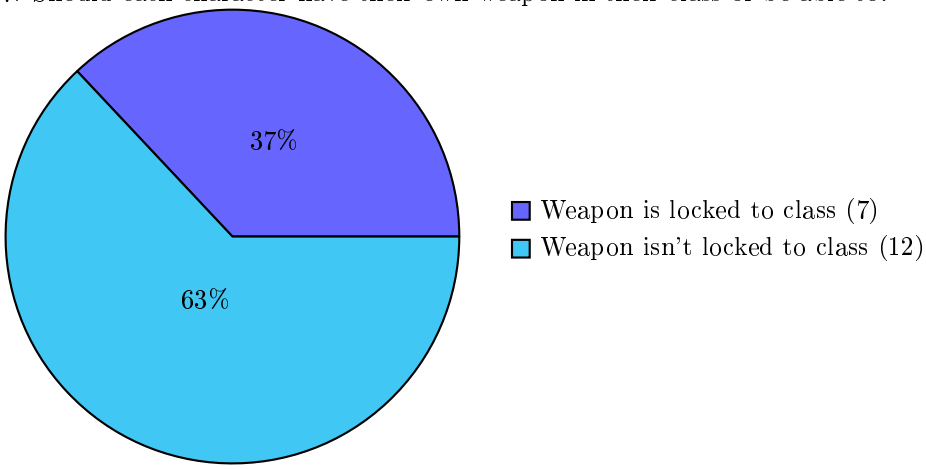
Combat and movement were the most common option so they should be the most polished systems in the game. Though different playstyles was something I should consider. Permanent progression wasn't a popular choice so a roguelike style would be more suitable

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	0.69%	-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%	-7.87%
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	36.02%	+9.82%
Free Hard Drive Space	100 GB to 249 GB	23.46%	+4.34%
Total Hard Drive Space	Above 1 TB	51.99%	-13.18%
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

CPUs so this is a valid statistic to use), Physical CPUs, 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 occidentally 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

Decomposition of 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.

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

4.1 Folder Set up

4.2 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

4.2.1 Create Rooms

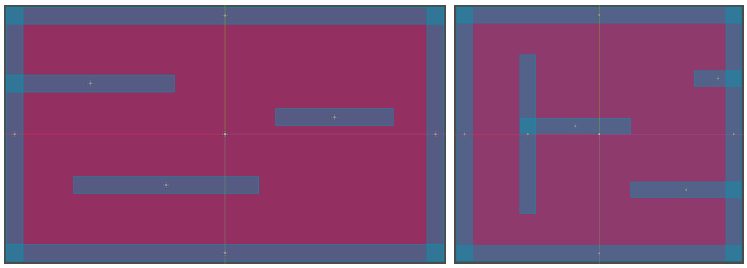


Figure 10: Room layout example

Begin with generating a defined amount of rooms. Rooms are pre-made layouts that a room as seen in figure 10. 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.

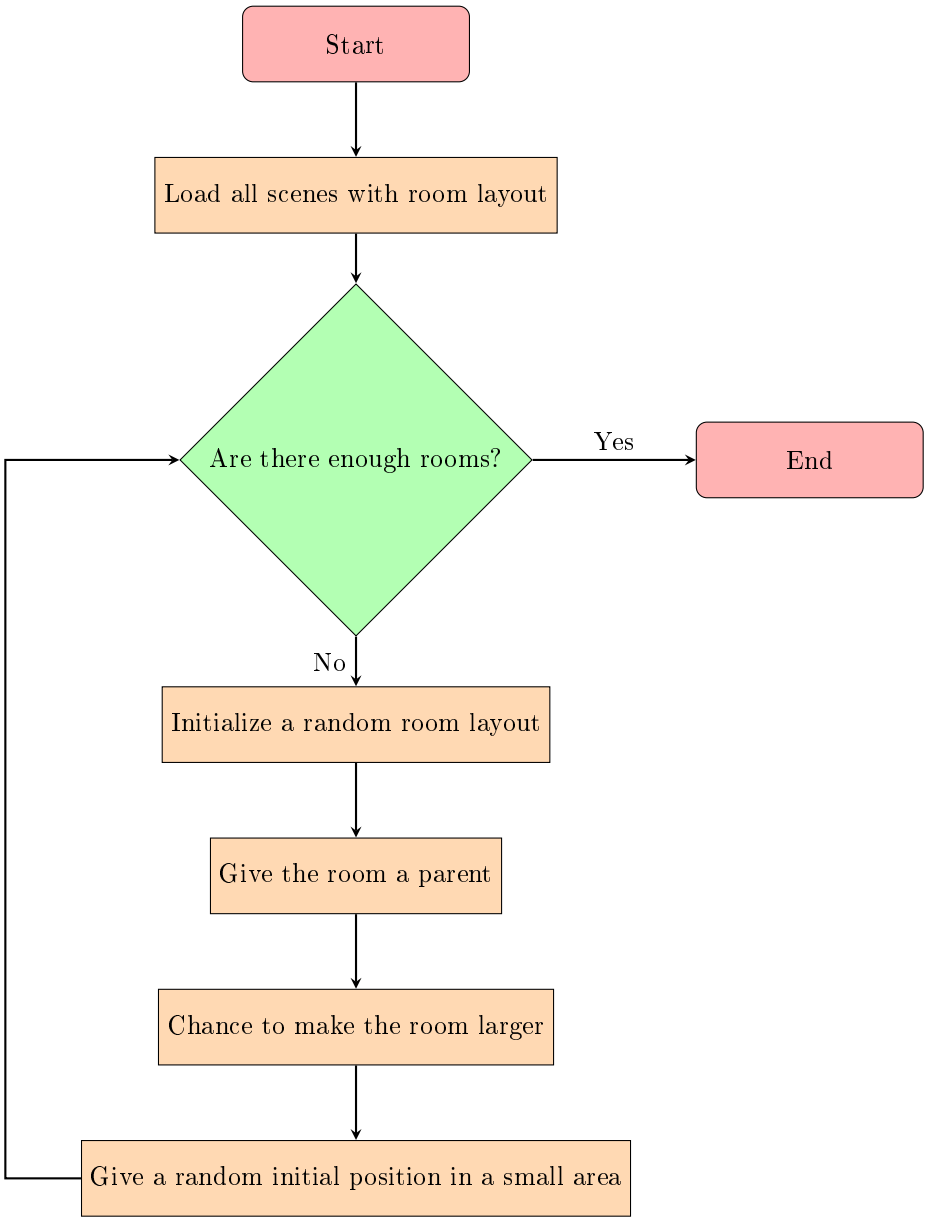


Figure 11: Flowchart for generating rooms

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	Definse amoubt 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
The algorithm will be successful if the following conditions are met:

- There is a correct amount of rooms
- There is variation in rooms generated

4.2.2 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]

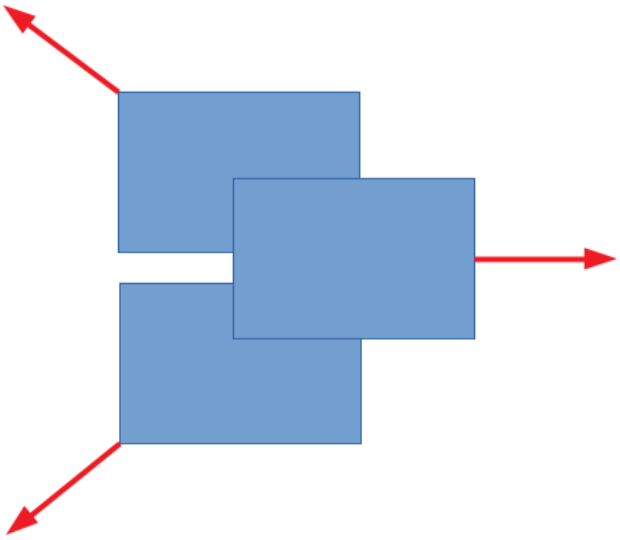


Figure 12: [Dis] Room Movement Direction

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

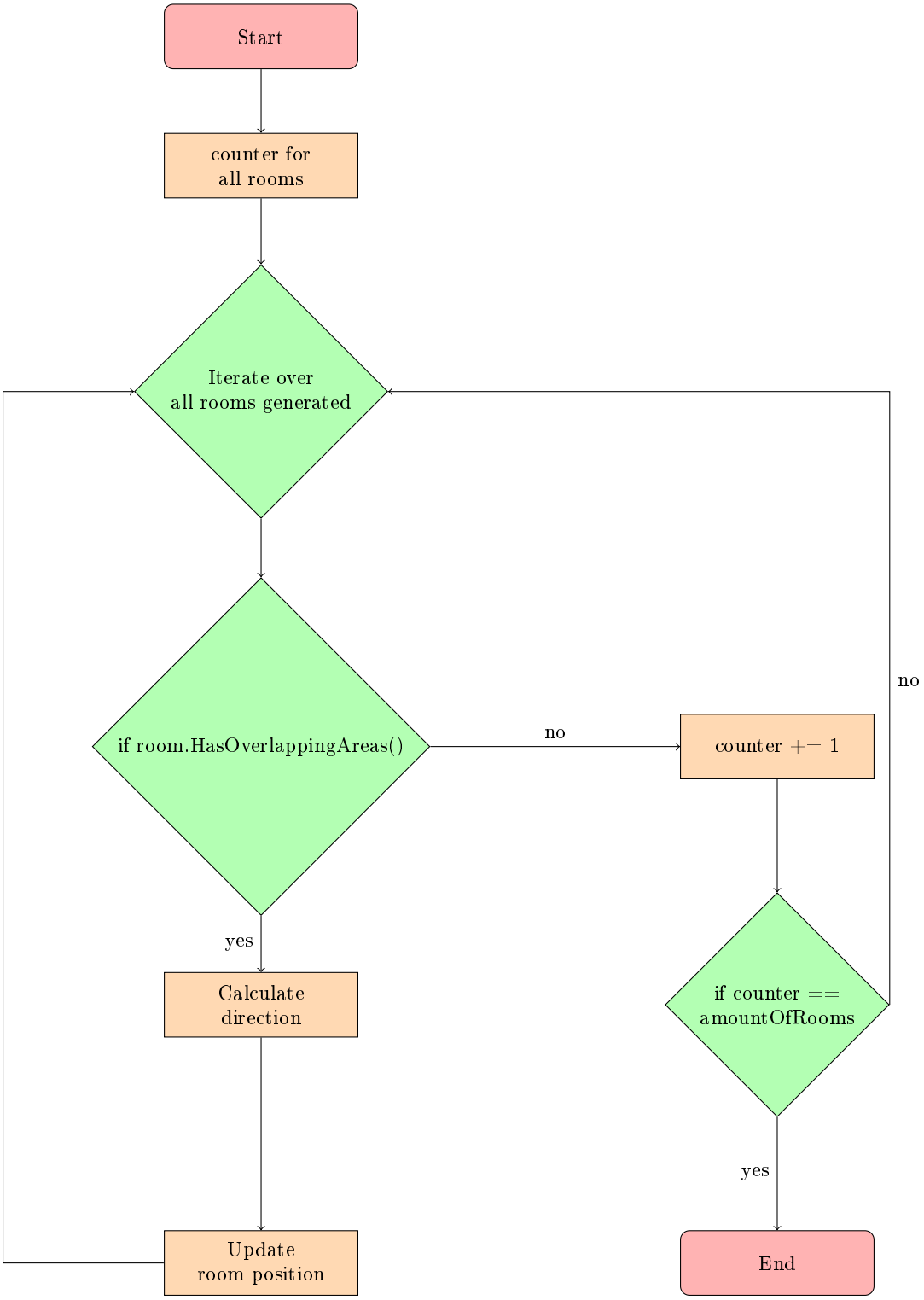


Figure 13: Flowchart for spreading rooms

Pseudocode for figure 13

```
//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
The algorithm will be successful if the following conditions are met:

- Rooms don't overlap
- Rooms are still close together
- Spreads in a reasonable amount of time

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

4.2.3 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

deleter 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.

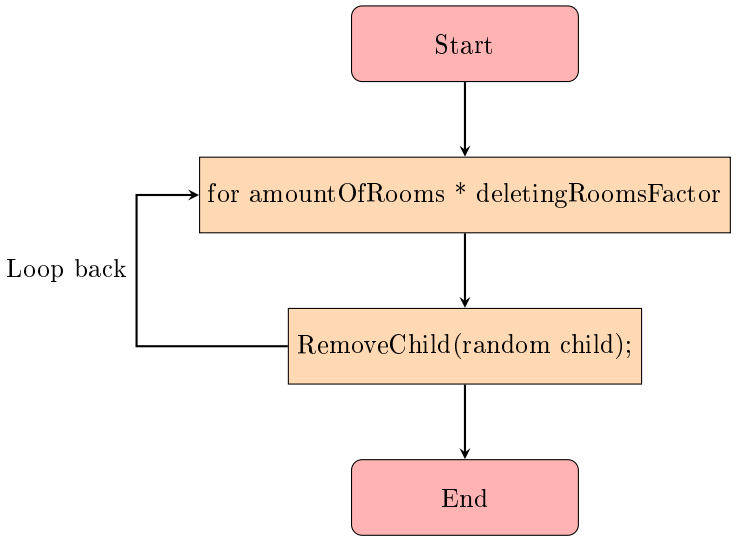


Figure 14: Flowchart for deleting rooms

Pseudocode for figure 14

```
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
The algorithm will be successful if the following conditions are met:

- Rooms are spread apart without being on top of each other
- Distance between rooms varies

4.2.4 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.

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

Edge
Private points: Array[2] Point
Public HasPoints(Point point1, Point point2) returns bool weather the edge has those points

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

Figure 15: Bowyer-Watson Classes

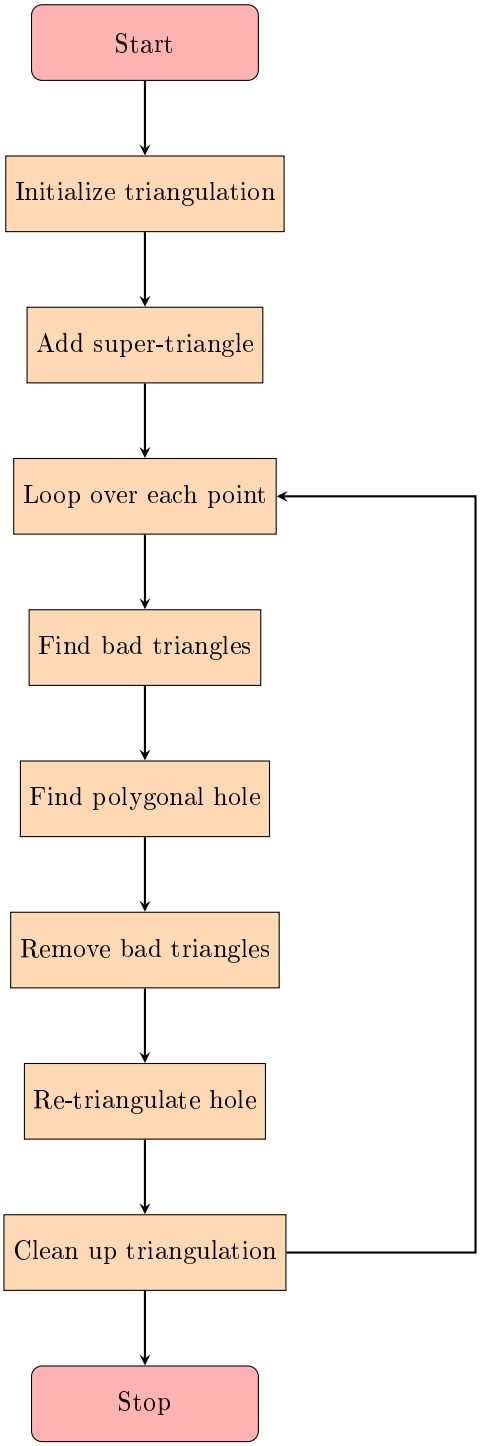


Figure 16: Bowyer-Watson Algorithm Flowchart

Pseudocode for figure 16
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
The algorithm will be successful if the following conditions are met:

- Triangulated in the most efficient way possible
- Every room is included in the triangulation
- Computes in a reasonable amount of time

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.

4.2.5 Remove random amount of edges

Figure 29 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.

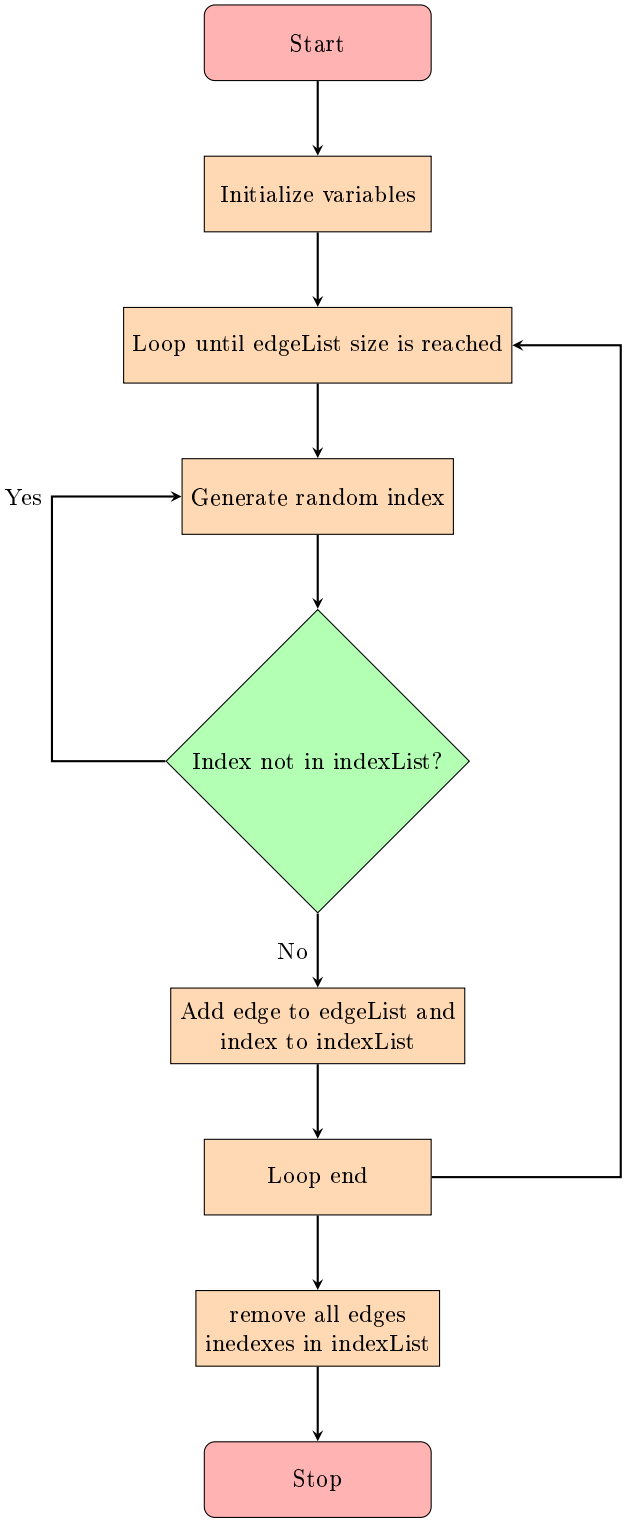


Figure 17: Flowchart for Edge Deletion Algorithm

Pseudocode for figure 17

```
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
The algorithm will be successful if the following conditions are met:

- Random amount of edges is deleted

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.

4.2.6 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.

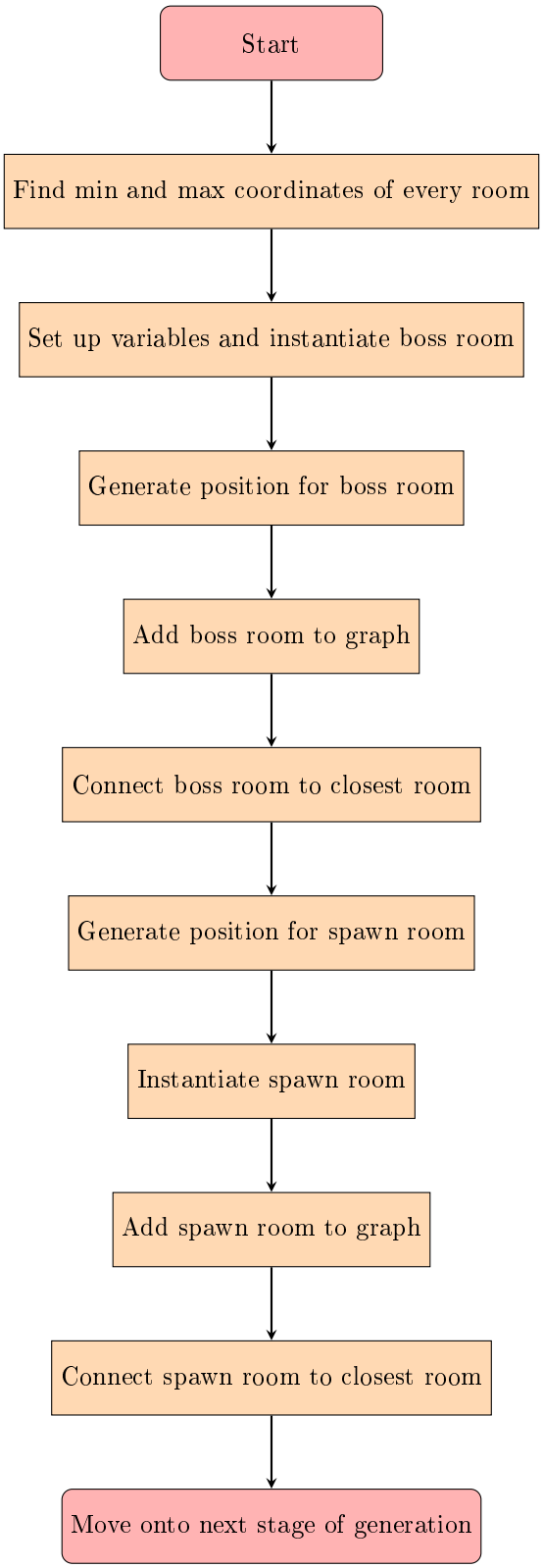


Figure 18: Flowchart for Generating spawn and boss rooms

Pseudocode for figure 18

```
//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";
//genereate 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 then 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);
//lable room to make it easy to identify
spawnRoom.Name = "rmS";
//genereate 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
The algorithm will be successful if the following conditions are met:

- Spawn room and boss room are generated
- Spawn is generated above all other rooms
- Boss room is generated below all other rooms
- Both rooms are generated relatively close to the center

4.2.7 Make sure all rooms are accessible

This section can be decomposed into smaller problems. That being detecting sections and connecting sections. 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.

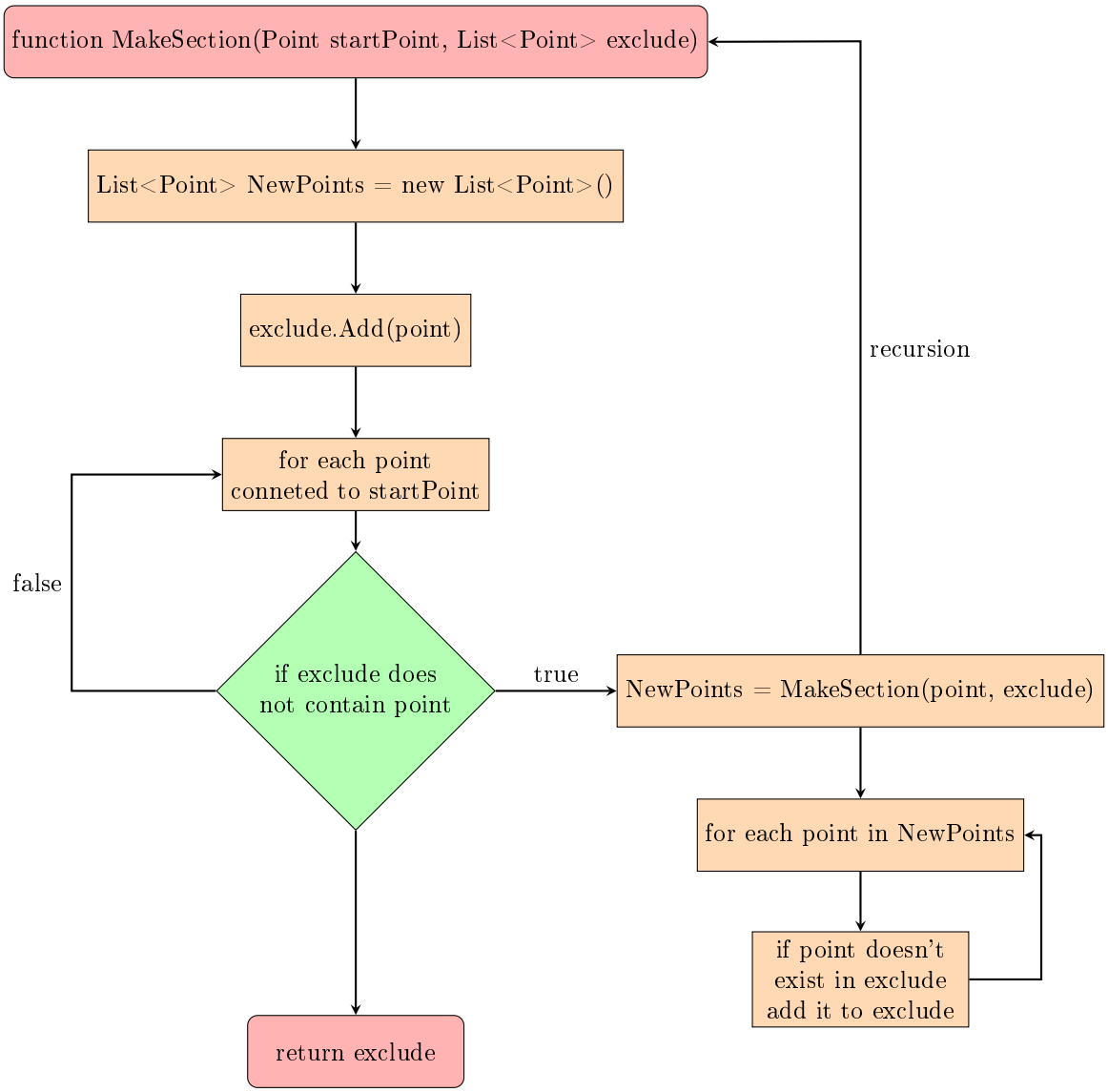


Figure 19: Flowchart for recursive function to find entire section

Pseudocode for figure 19

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
endfuntion
```

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

The algorithm will be successful if the following conditions are met:

- An entire section is detected given a single input point

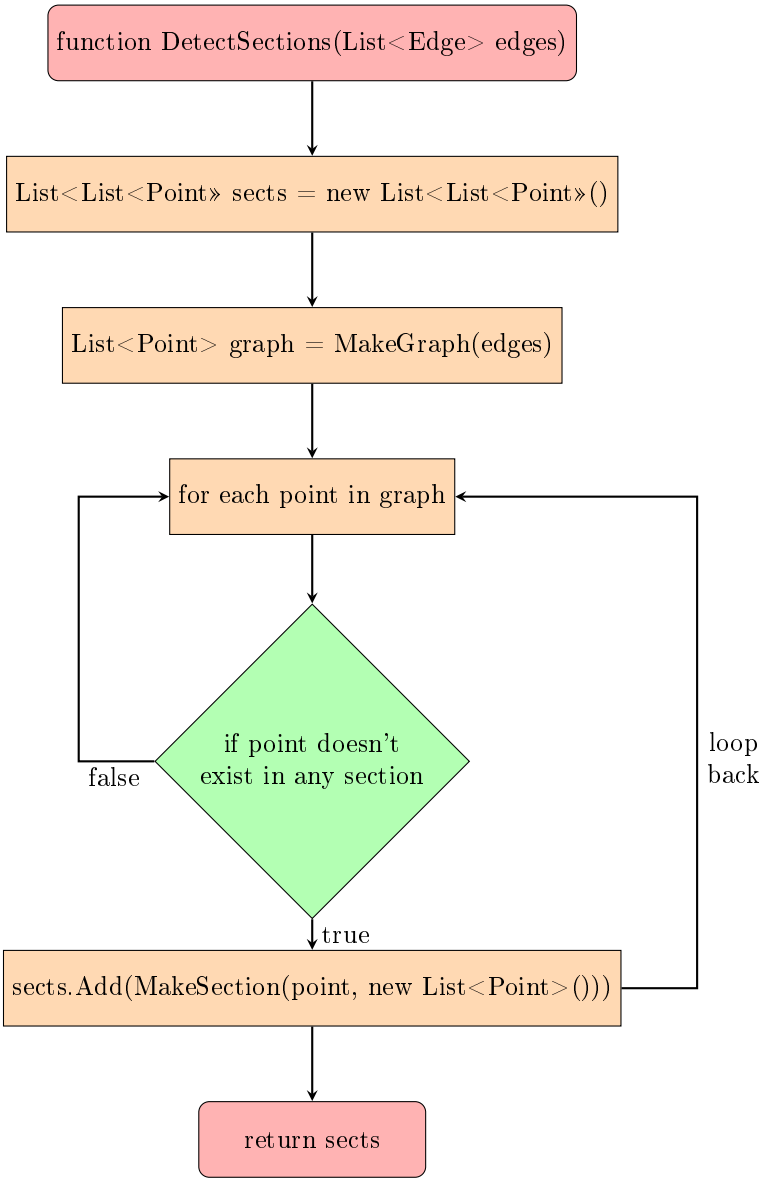


Figure 20: Flowchart for detecting if a new section needs to be made

Pseudocode for figure 20
Algorithm for detecting weather 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 weather 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
The algorithm will be successful if the following conditions are met:

- Detects correct amount of sections
- Doesn't create another section when point being checked is already in a section

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.

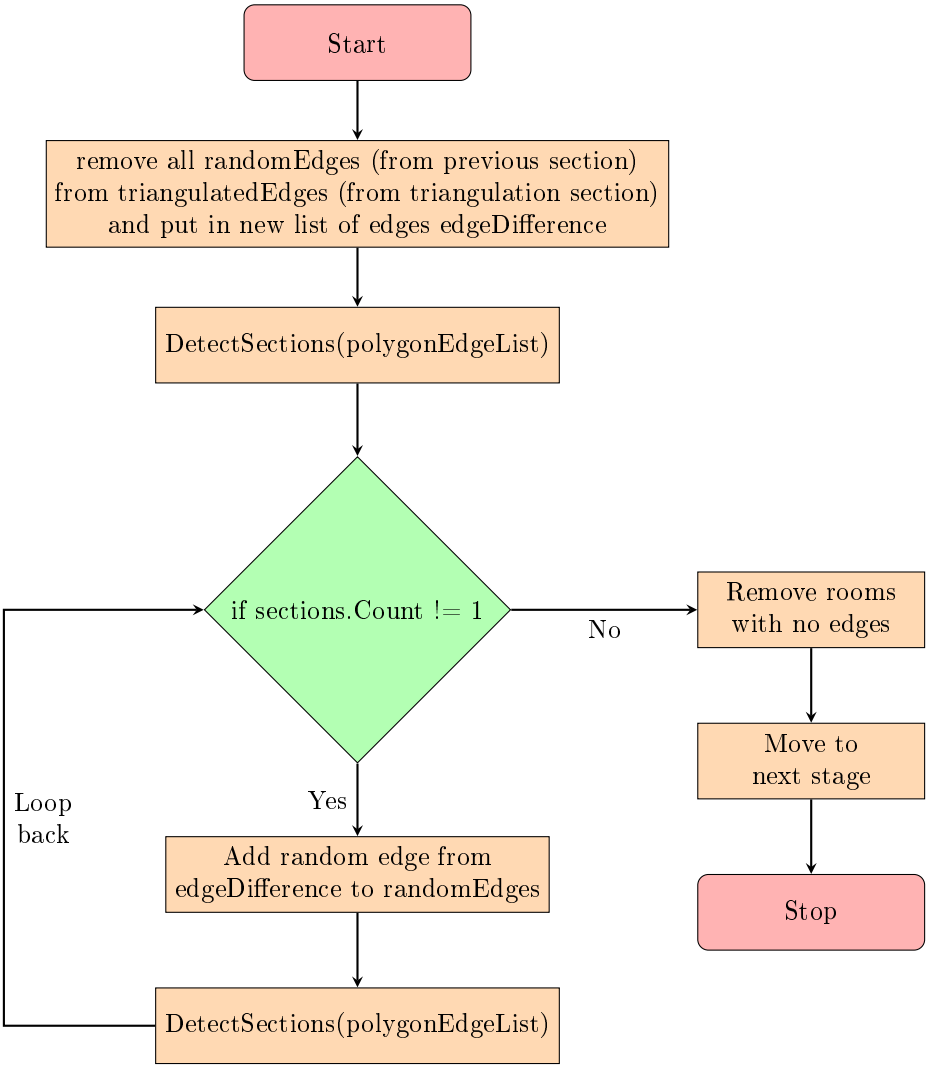


Figure 21: Flowchart for detecting sections in the program

Pseudocode for figure 21

```
//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.RemoveAtIndex(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
The algorithm will be successful if the following conditions are met:

- The list of edges are now all connected to each other
- Removes all rooms without connections to other rooms

4.2.8 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 30. 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.

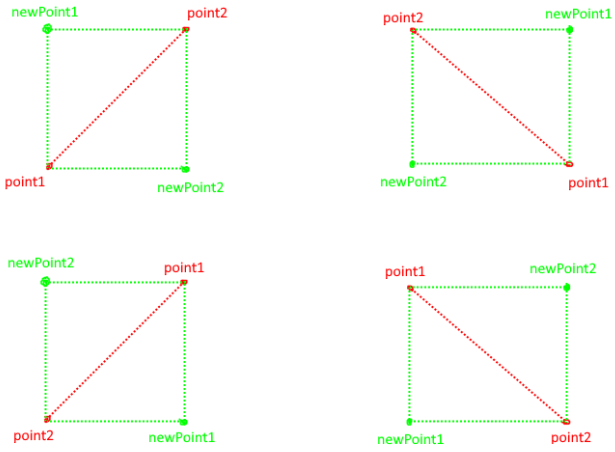


Figure 22: All possibilities for points

The green dotted lines in figure 22 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.

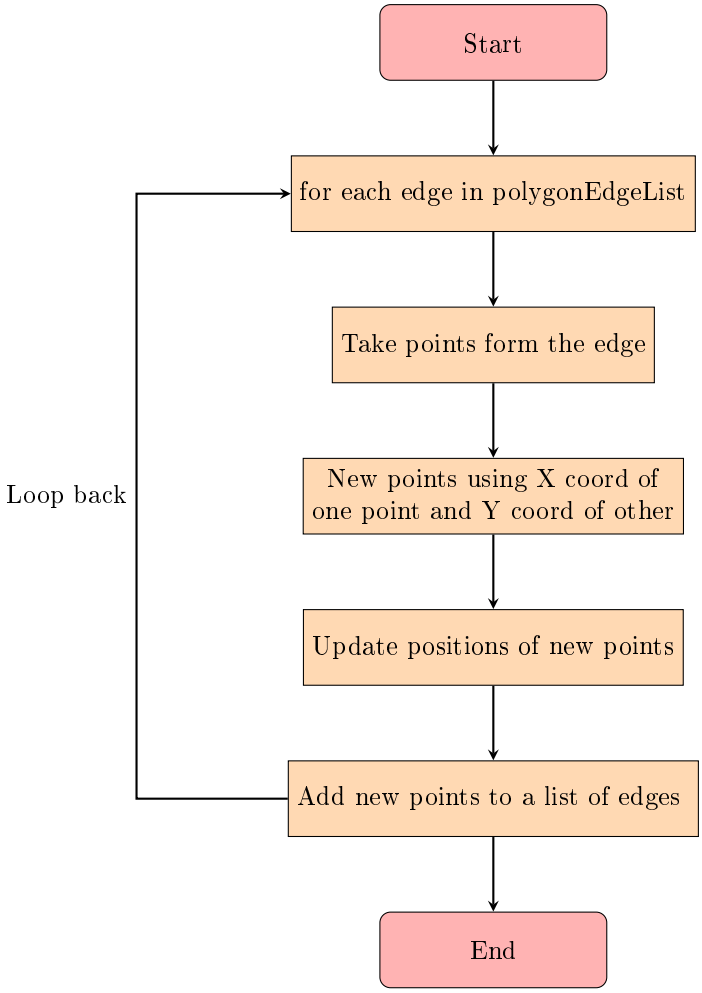


Figure 23: Flowchart for turning edges into horizontal and vertical edges

Pseudocode for figure 23

```
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>	Contaings 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
The algorithm will be successful if the following conditions are met:

- Entire graph is made from horizontal and vertical edges

In figure 31 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.

4.2.9 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.

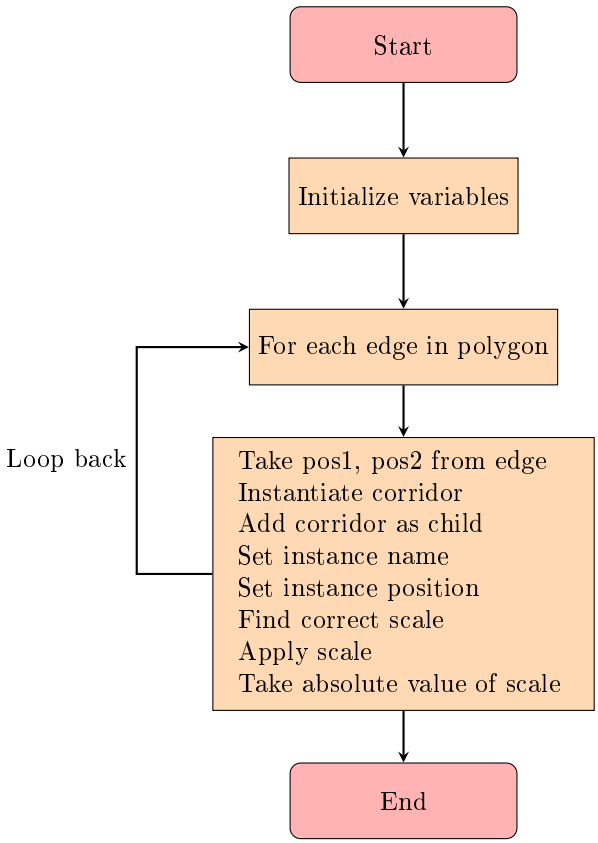


Figure 24: Flowchart for the given pseudocode

Pseudocode for figure 24

The result of this algorithm can be seen if figure 31

```
// 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 corridor 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 coridoors
edge	Edge	The current edge being turned into a coridoor
pos1	Vector2	The start position of the coridoor
pos2	Vector2	The end position of the coridoor
roomRadius	Vector2	How much the coridoor corner is offset from the center
instance	Node	The new coridoor that is instantiated

Identifying test data
The algorithm will be successful if the following conditions are met:

- Each edge has a corridor generated for it
- The corridor is an appropriate size

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.

4.2.10 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.

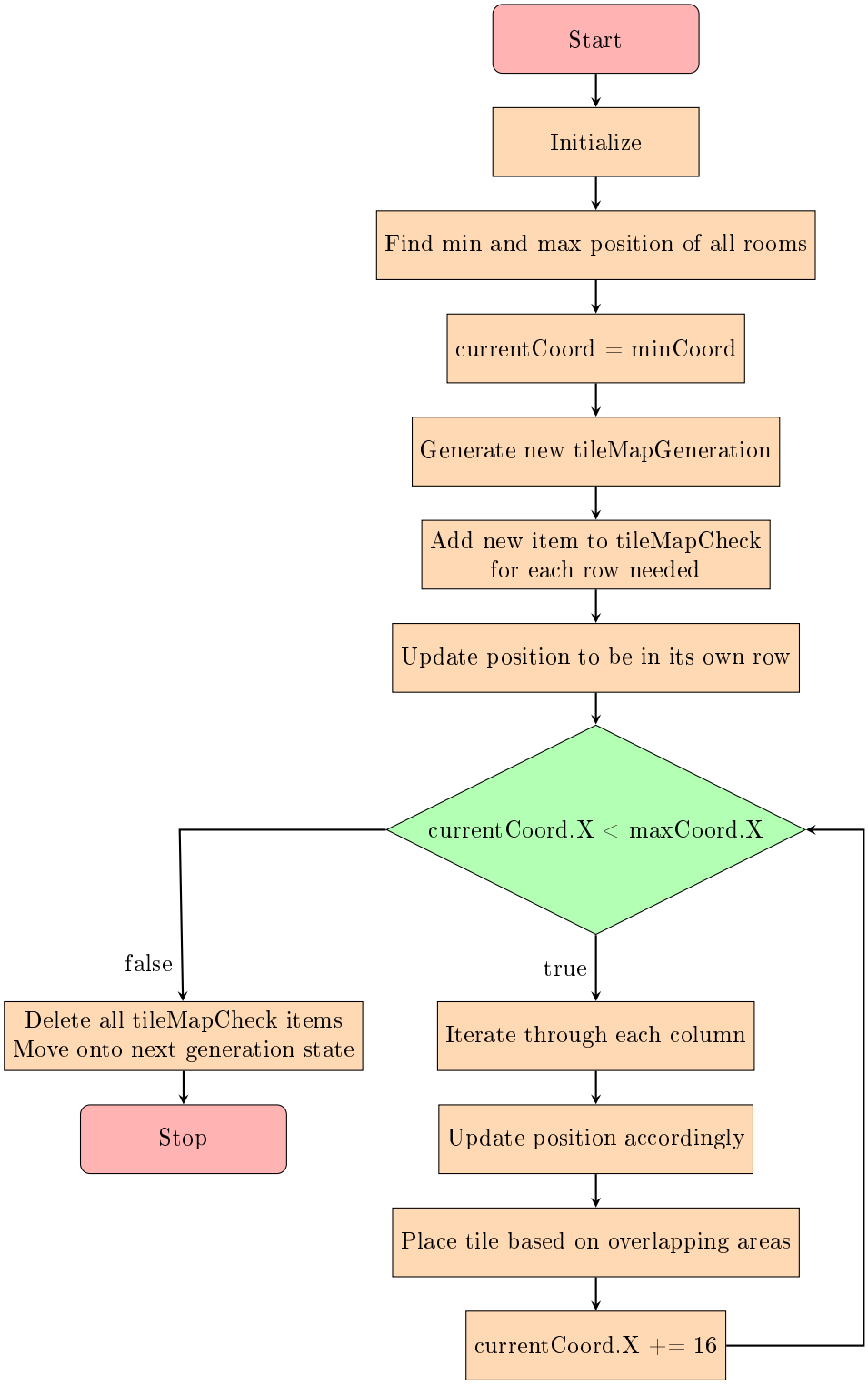


Figure 25: Flowchart for generating the tilemap

Pseudocode for figure 25
**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
The algorithm will be successful if the following conditions are met:

- Takes reasonable amount of time
- Variation in tiles that looks good
- Makes an interesting map for the player to explore

4.2.11 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.

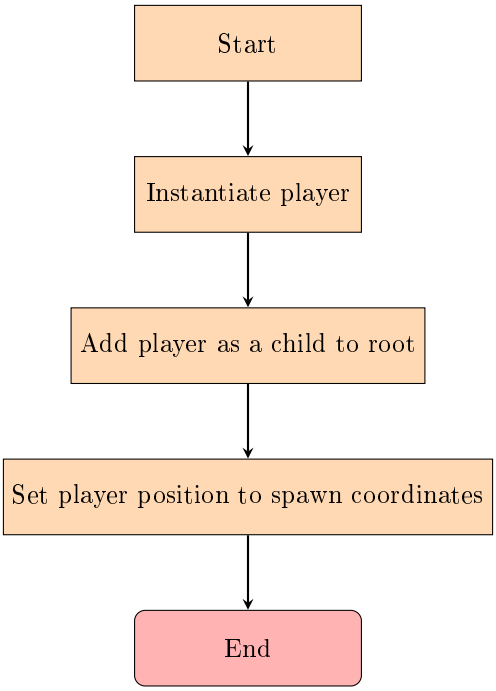


Figure 26: Flowchart for spawning the player

Pseudocode for figure 26

```
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 form previous section

Identifying test data
The algorithm will be successful if the following conditions are met:

- Player is spawned
- Player is able to be controlled

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.

4.3 Player Movement

4.4 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.

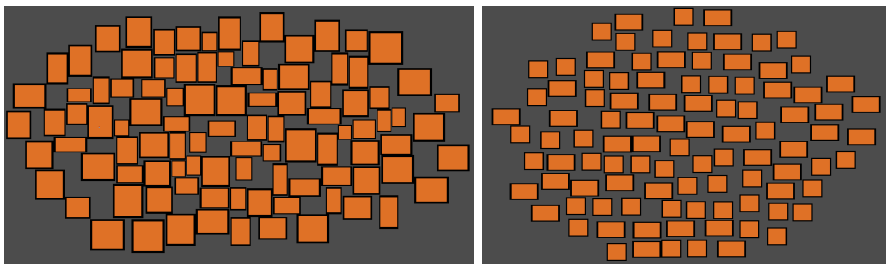


Figure 27: Result of room separation

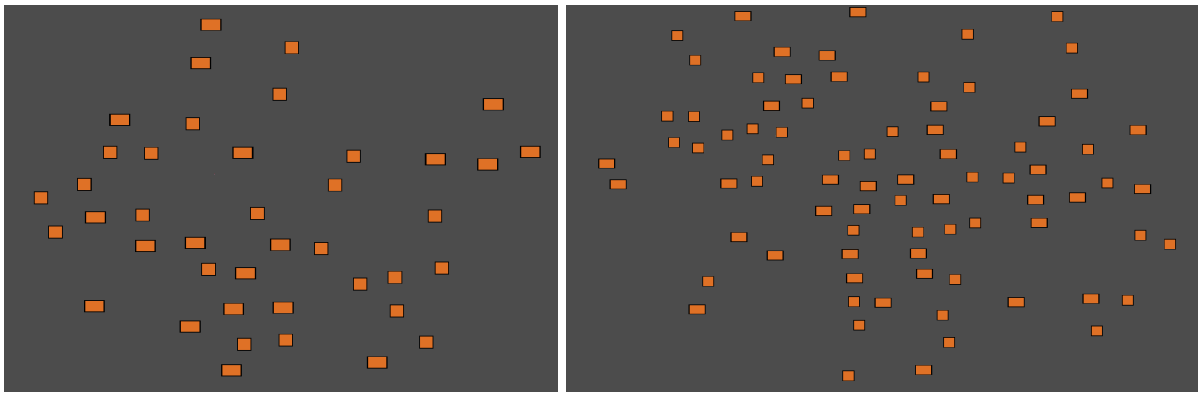


Figure 28: Result of room deletion

4.5 Enemy Design

4.6 Boss Design

4.7 UI

5 Developing a solution

5.1 Temp Fig Holder

6 Evaluation

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.

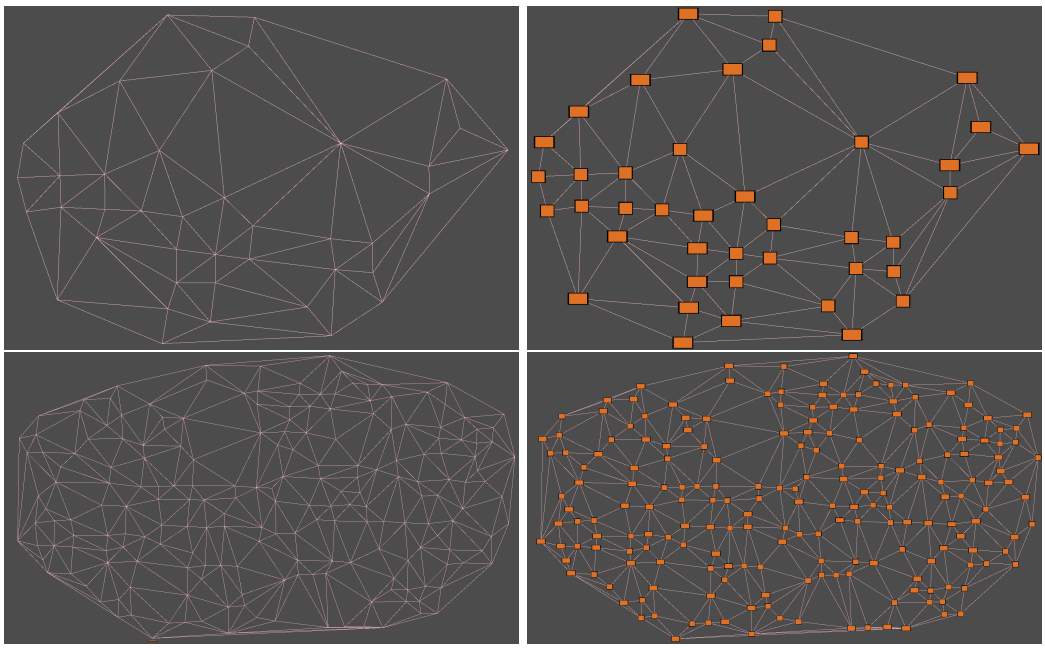


Figure 29: Result of Bowyer–Watson algorithm

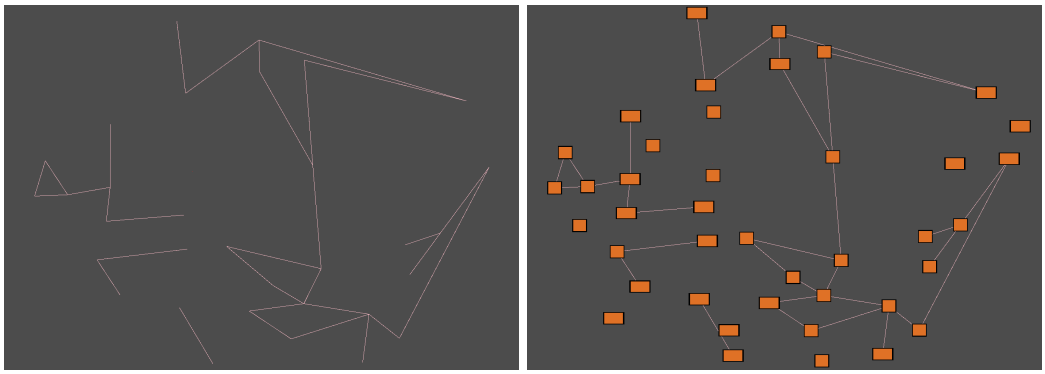


Figure 30: Deleting edges result

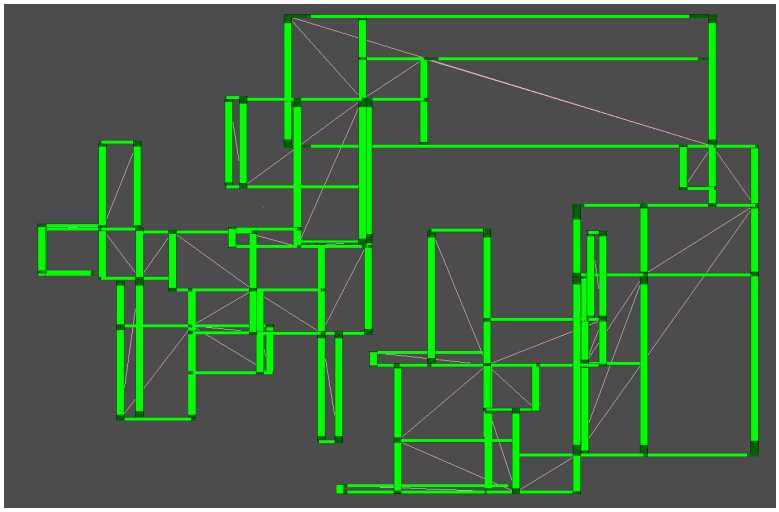


Figure 31: Before (pink) and after algorithm (green)

References

[Bwa] *Bowyer–Watson algorithm*. Accessed: 07/12/23. URL: https://en.wikipedia.org/wiki/BowyerâĂŞWatson_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/.

[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/.

[Ror] *Risk of Rain 2 on Steam*. Accessed: 25/09/2023. URL: 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*. URL: https://docs.godotengine.org/en/stable/about/system_requirements.html#exported-godot-project.