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

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.

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 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   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 a game which is fun and the following concepts allow me to accomplish this

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 all the game code with many, many errors

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.

Abstraction – is 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 points will get the most amount of time making the basics of the game reliable.

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 for example while or for loops which will continue to cycle the code until it is told to stop. These are useful since in a 3D 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 like mortal combat you can chain inputs to do a special attack.

4 Researching The Problem

4.1 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.

Good Qualities

- The game is very beginner friendly and is enjoyable both alone and with friends.

- 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 character makes any play style suitable.

- The game is never "too easy" since there are eclipse challenges which get progressively harder

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 which slow down the game a lot.

- Some things can kill you in one hit making death unavoidable in some cases.

- Sometimes you get runs where you get no good items, and the game becomes a lot more difficult because of bad luck.

What I would include

- The variety in characters and the play styles that they allow

- The different difficulties allowing the player to play at a level that their comfortable with
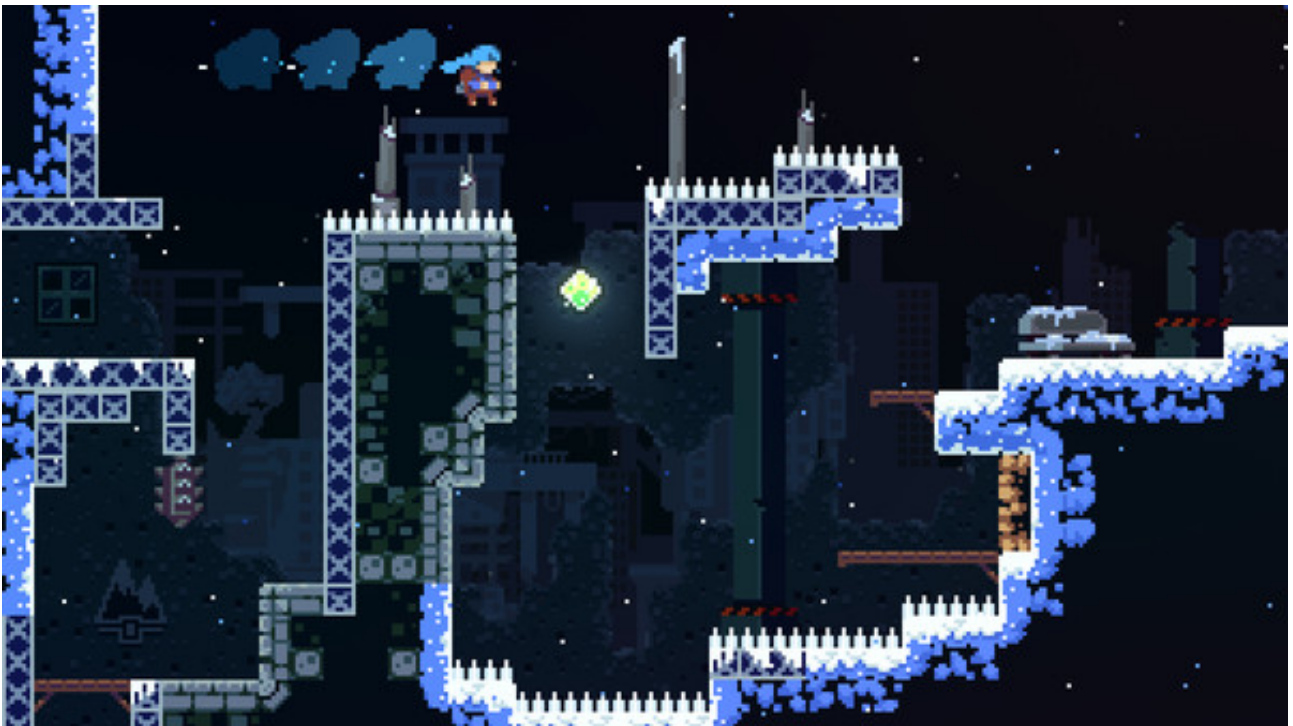
4.2   Celeste

Figure 2: [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.

Good Qualities

- Beating a room feels rewarding

- Caters who anyone who is willing to try it.

- The movement feels fun

- The movement feels consistent

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

- Similar movement system, but something that isn't as complicated so it doesn't overwhelm the player

4.3 Rain World

Figure 3: [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.

Good Qualities

- Game play is fun and keeps the player engaged

- 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 if very rewarding.

- Player is treated as part of ecosystem rather than separate entity

Bad Qualities

- Difficult to get into due to no tutorial

- Game is very unforgiving and a small mistake can result in the player getting killed

What I would include

- A similar combat system

- Similar movement system

- Treating the player as part of the ecosystem

5 Market Analysis

5.1 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).

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 playtesting 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.

5.2   Letter to Client

<div align="right">
31 Bucannon Road
CV22 6AZ
20/11/2022
</div>

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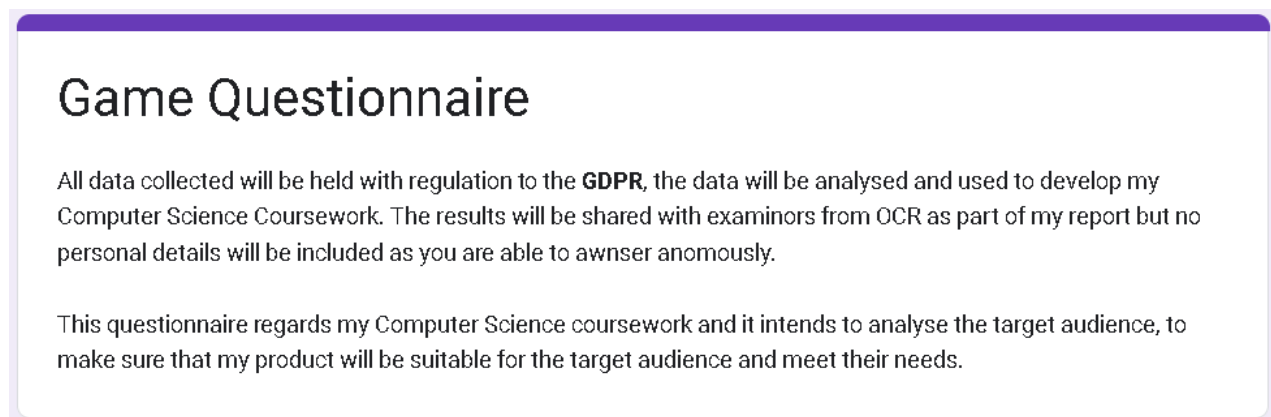I am asking you to become a game tester for my game so that I can get feedback from you as to how to improve the game. In the future I will ask you to give me feedback on the game to improve the quality and usability of the game, this will be in the form of interviews and game testing. This will hopefully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

5.3   Questionnaire

Figure 4: Questionnaire Header

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?

less than 12 (1)
between 12 and 17 (12)
between 18 and 25 (3)
above 25 (2)
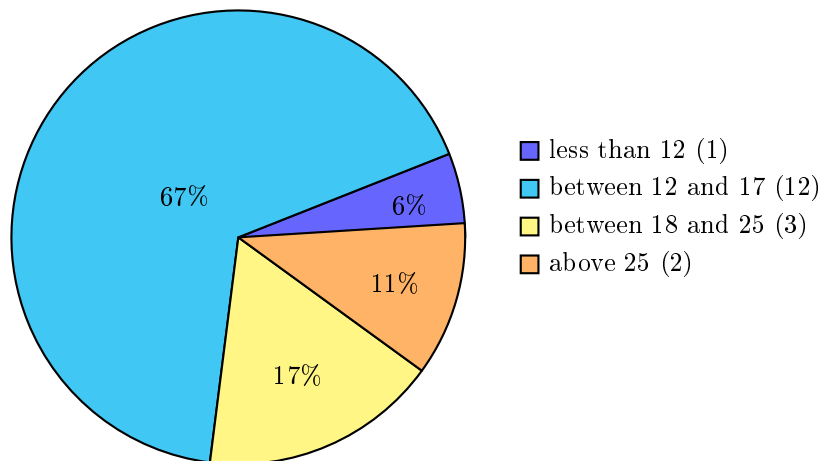
We can see that the target demographic is teens and above meaning that the game can include some violence. It is important to make the game appropriate to the target demographic as they will be the majority of the player base.
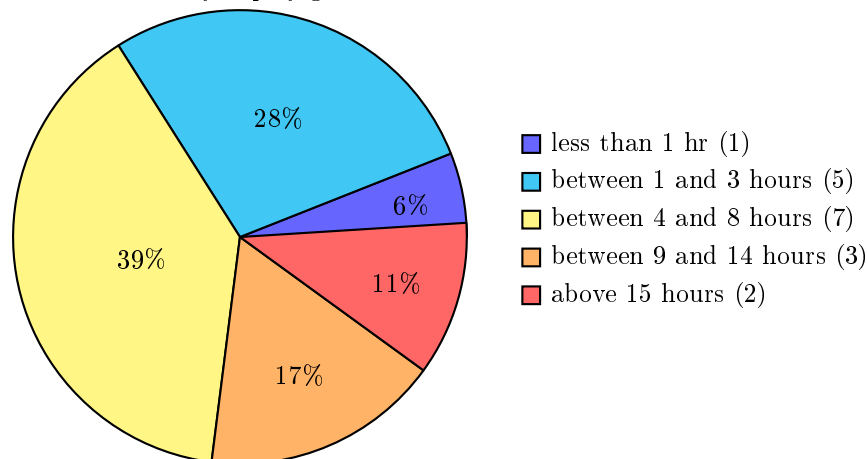
Q2: How much do you play games a week?

less than 1 hr (1)
between 1 and 3 hours (5)
between 4 and 8 hours (7)
between 9 and 14 hours (3)
above 15 hours (2)

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?

Console (5)
Mobile (2)
PC (11)
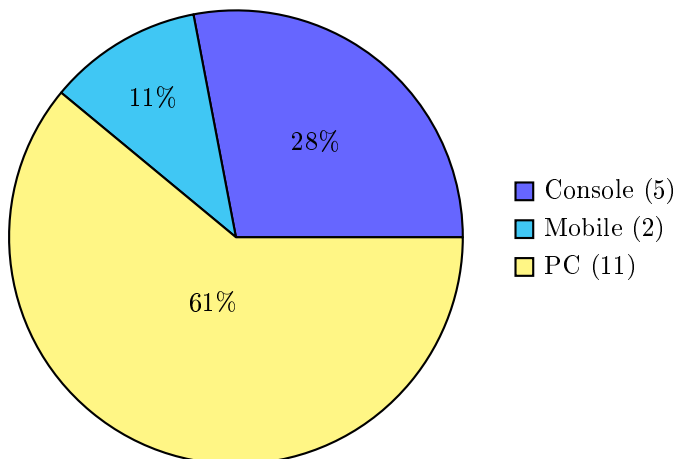
Most people answered replied that they play on pc meaning that it should be ported to pc first. A significant amount of people also play on console so it would be good to port the game to console if possible.

Q4: what genre of games do you play?

- Rougelike or Rougelite (5)
- Stratergy (2)
- Platformer (4)
- Fighter (2)
- Survival (3)
- Open World (2)
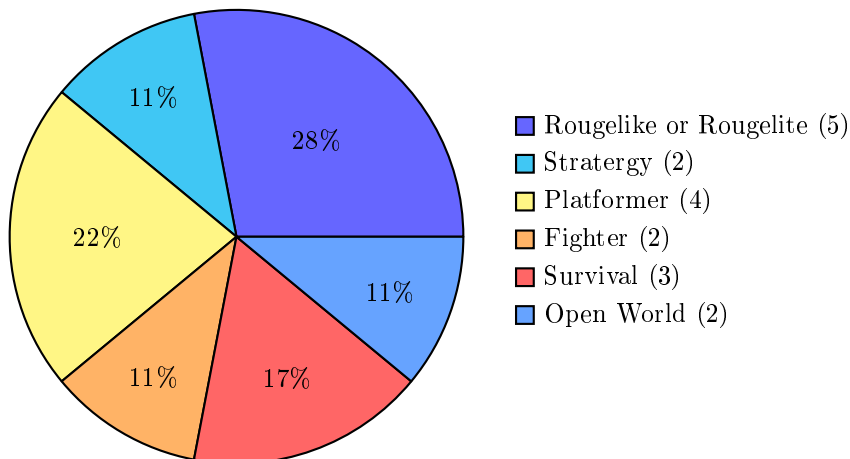
Most people answered Roguelike/Rougelite and platformer meaning I should focus on these two aspects the most when making my game. Survival will be implemented though planning ah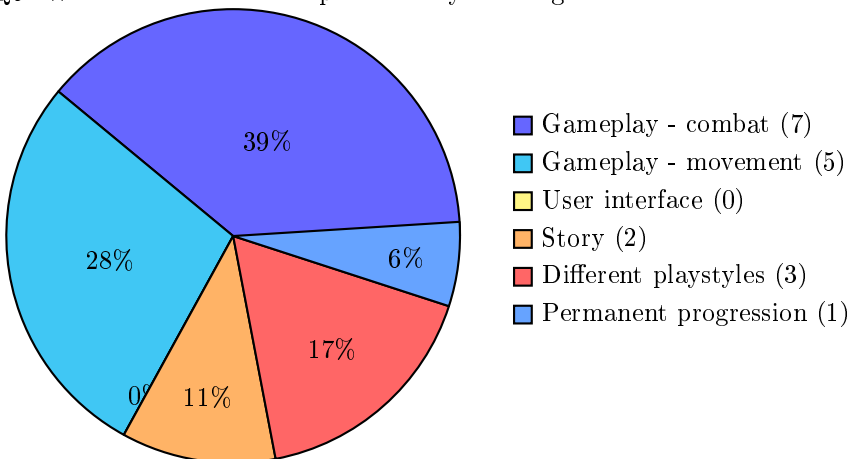ead this might be what routes to take and the risk associated with those routes as well as the risk of combat. As well as management of resources to make sure you survive to the next stage.

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

- Gameplay - combat (7)
- Gameplay - movement (5)
- User interface (0)
- Story (2)
- Different playstyles (3)
- Permanent progression (1)

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?

- 1 (6)
- 2 (9)
- 3 (3)
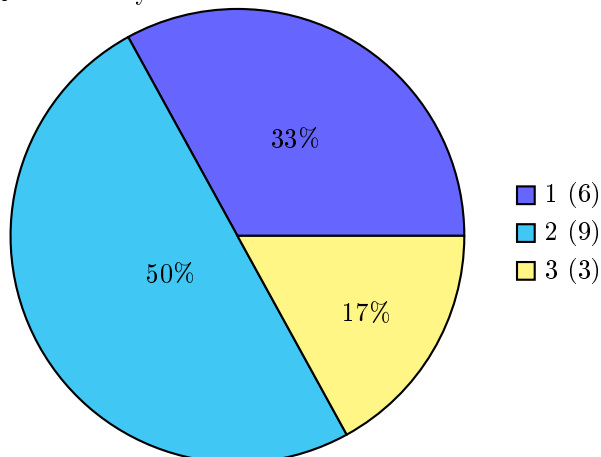
Most people answered 2 so it would be important to make the skills distinct and unique to allow for different

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?

- ■ Weapon is locked to class (7)
- ■ Weapon isn't locked to class (12)
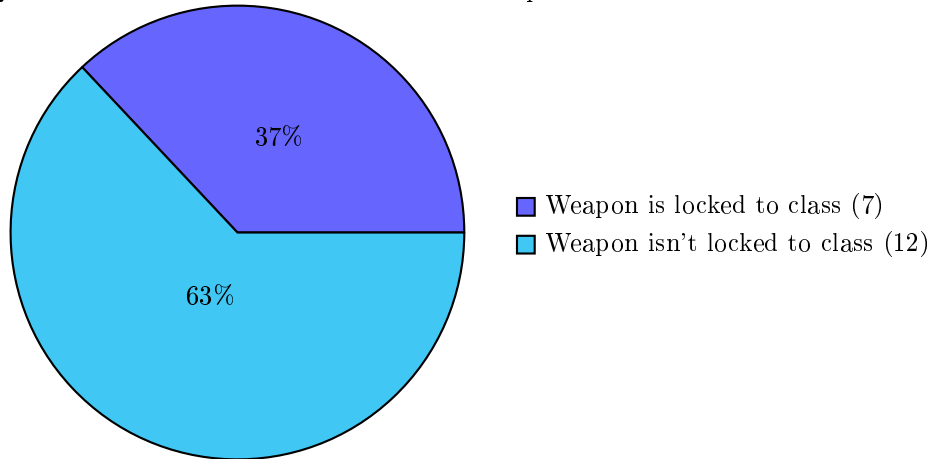
The majority wanted to have weapons that are picked up and not locked to a class therefore I will give each character a starting weapon and give them the ability to change this weapon if they wish to, allowing the player to experiment with different weapons that may suit their play style.

5.4   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.

5.5   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 5: [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 5 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	48.88%	+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%	+9.62%
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 6: [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 6 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.

6   System Requirements

Minimum Hardware Requirements

- Processor:
- Graphics Card:
- Memory:
- Storage:
- Sound Card: yes

Input devices:

- Keyboard – allows you to input characters
- Mouse – Inputs 2D vector coordinate for mouse position

Output devices:

- Display – Outputs image
- Speakers – Outputs sound

Minimum Software Requirements

- OS: Windows 10+
- DirectX version: 12

7   Success Criteria

1. **User Interface**
1.1. **Main Menu**
1.1.1. Does it have a new game button?
1.1.2. Does it have an options button?
1.1.3. Does it have a tutorial option?
1.1.4. Does it have a interesting background?
1.1.5. Does it have a quit game button?
1.1.5.1. Does it ask you "are you sure"?
1.2. **Pause Menu**
1.2.1. Does it pause the game?

1.2.2. Is there an options button?

1.2.3. Is there an exit to main menu button?

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

1.3. **HUD (Heads Up Display)**

1.3.1. Is there a health bar?

1.3.2. Is there a current level box?

1.3.3. Is there a timer?

1.3.4. Are there small indicators for status effects?

1.3.5. Is the HUD customisable?

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**

8 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

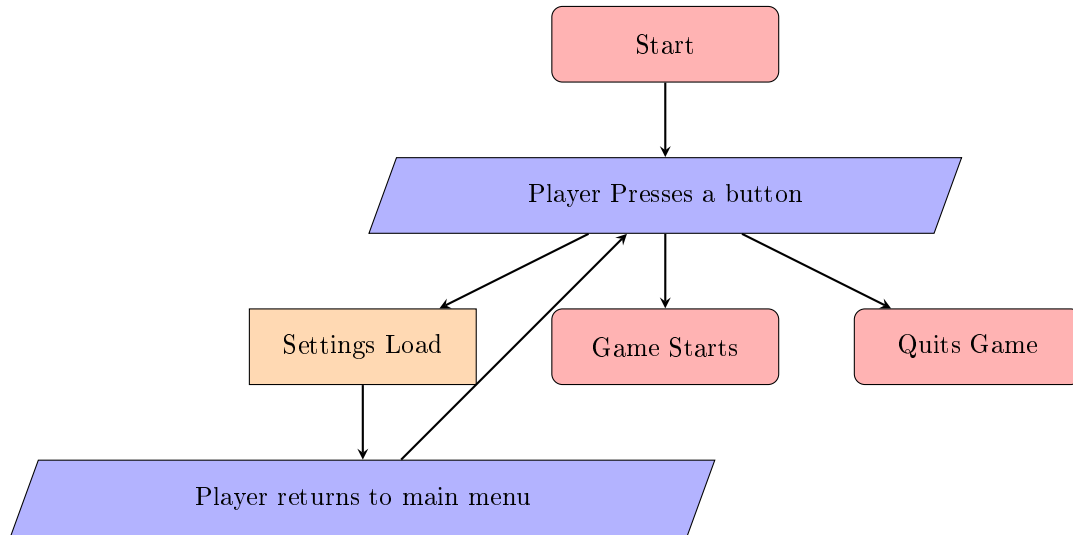CONSTANT − represented as capitalized with underscores

variable − represented as lowercase with underscores

8.1 Folder Set up

8.2 Main Menu

The main menu will be there to greet the player once they open the game and allow them to choose what they would like to do.

These will be in the form of buttons that the user can click on.

8.3 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

- Delaunay triangulation

- Remove random amount of edges

- Make sure all rooms are accessible

- Turn edges into horizontal and vertical

- Generate corridors

- Turn into tile map

Some other miscellaneous algorithms used in generation

- Turning list of edges to graph

- Turning graph to list of edges

8.3.1 Create Rooms

Begin with generating a defined amount of rooms. Rooms are pre-made layouts that a room as seen in figure 7. 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.
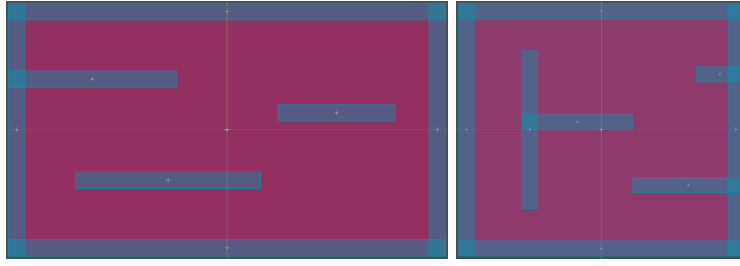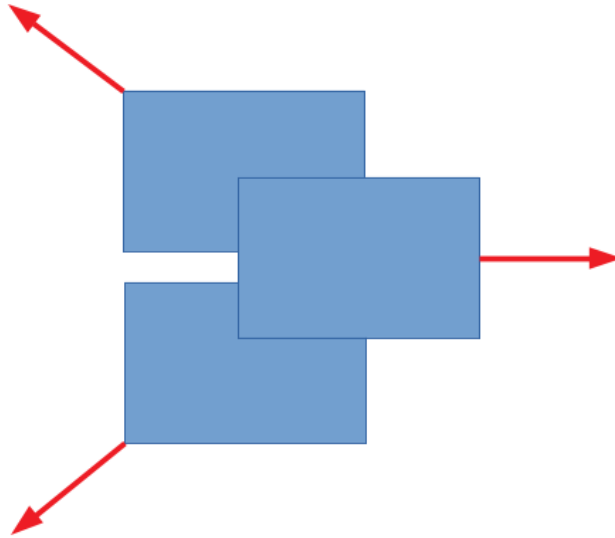
Figure 7: Room Layout

Figure 8: [Dis] Room Movement Direction

8.3.2 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]

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

Pseudocode

```
//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
}
```
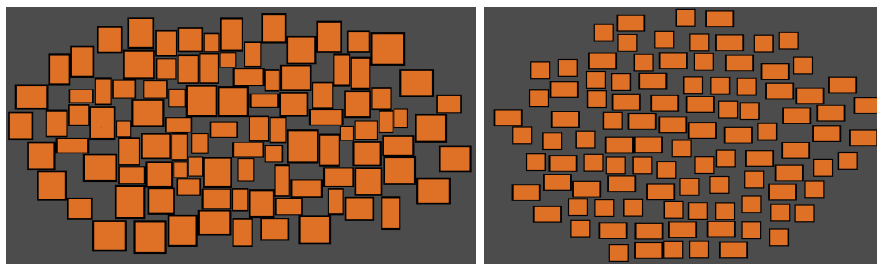
Once complete a result typically looks as such

Figure 9: Result of room separation

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
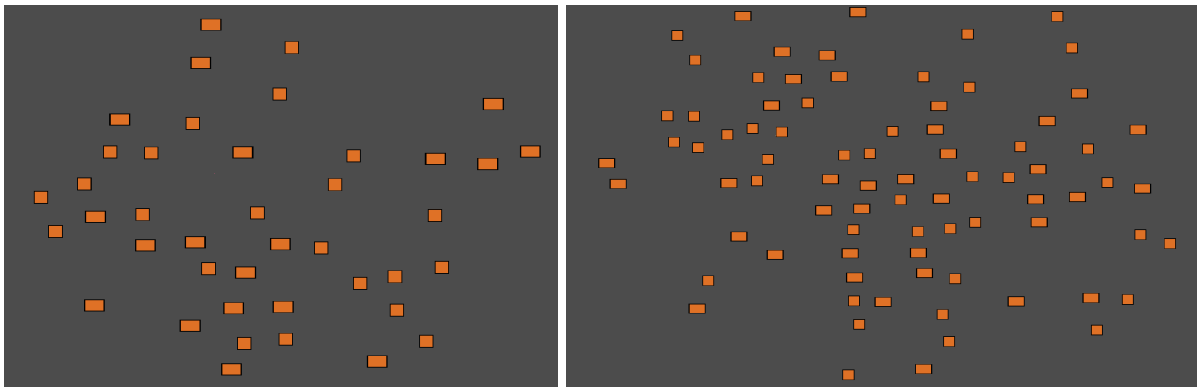

Figure 10: Result of room deletion

8.3.3 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.

Pseudocode
The pseudocode was taken from [Bwa]

```
function BowyerWatson (pointList)
    // pointList is a list of points defining the points to be triangulated
```

21

```
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
```
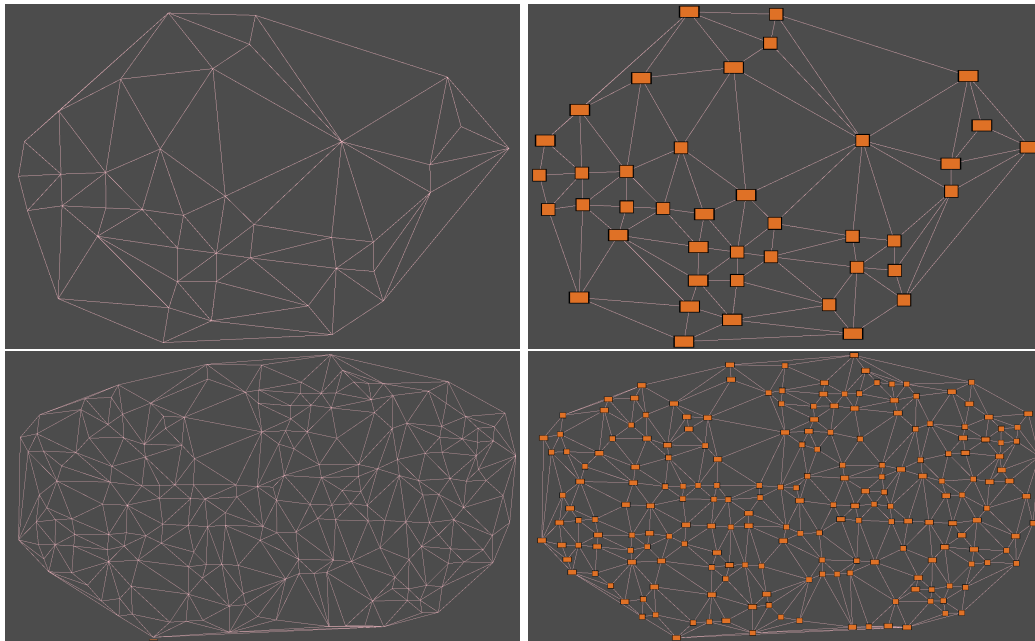

Figure 11: Result of Bowyer–Watson algorithm

8.3.4   Remove random amount of edges

Figure 11 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.

Pseudocode

```
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);
        }
}
```
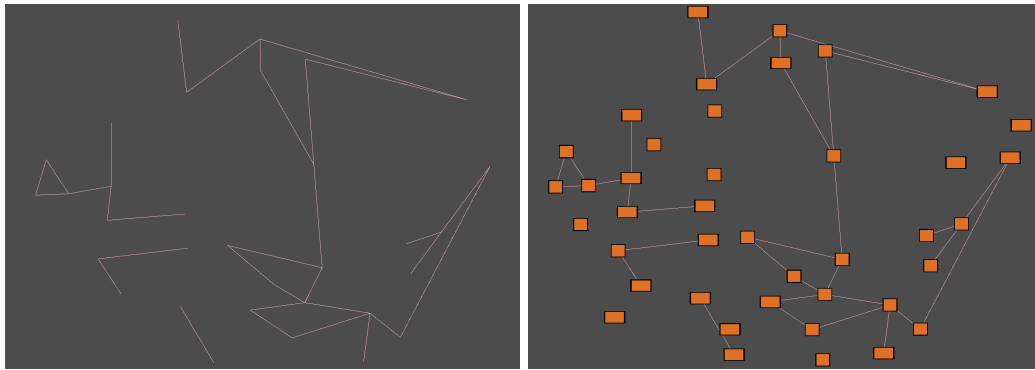

Figure 12: Deleting edges result

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.

8.3.5 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.

Pseudocode

Algorithm for detecting weather a new section needs to be made at the current point

```
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)
bool exists = false
for each point in graph
{
        //check weather point already exists in sects
        exists = false
        for each section in sects
        {
                for each sectionPoint in section
```

```
                    {
                            if (point.Position == sectionPoint.Position)
                            {
                                    exists = true
                                    //don't check rest of points if already
                                        detected it exists
                                    break
                            }
                    }
                    if (exists)
                    {
                    //don't check other points if already detected that it exists
                    break
                    }
            }
            if (!exists)
            {
                    //add section to list of sections
                    sects.Add(MakeSection(point, new List<Point>()))
            }
    }
    return sects
    endfunction
```

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
```

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.

```
//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)
}
```