

'Sunrise'

A multiplayer survival metroidvania
Scholarship Technology



Hello! For the past year, I have been working on a 3D co-op metroidvania, where you play as a lowly creature called a 'skrunk' in a harsh and unforgiving ecosystem. You must survive by hunting prey for food and finding shelter, whilst avoiding becoming prey yourself in a world where anything can kill – including the sun. It's currently available on Itch.io in its current state, but the ultimate goal is to get it up on Steam in the near future. This document will outline the research and brainstorming prior to development, planning development, and the entire design process.

Initial Research & Design

This was the first stage of the development process, which happened before I officially began development. I knew I wanted to make a game, but what kind of a game can I make?

What games do I enjoy playing?

Recently, I've been playing a lot of metroidvanias, platformers, open-worlds, and others. I've also been playing other weird niche indie games. I enjoy these games because they tend to be the most unique on the market. Each one is completely different. There are some similarities (which is a given for games of the same/similar genre), but main mechanics and premises are unique. A few games that I like that stick out to me are Rain World, Animal Well, Outer Wilds, and Lethal Company. I'll only go over Rain World and Lethal Company, as these were the two that I was inspired by the most throughout development.

I don't think there are any other games like Rain World. Its main premise is that it takes place in a large PvEvE ecosystem with some kind of food chain. However, you are at the bottom of this food chain and are tasked with surviving. The game is notoriously difficult because of this. However, since it's PvEvE, your enemies are also hunting your enemies, allowing you to potentially slip through the chaos. It's insanely difficult but very fun.





Lethal Company is already a good horror game. However, its multiplayer co-op is the best in the genre. In most dangerous situations in-game, you have two choices: assist your friends but put yourself at risk, or leave your teammate to save yourself, sometimes not even telling them about the danger. It's hard to put on paper, but I find it fun in having to make dangerous decisions that could cause you or your friends to die. It's something most don't notice, but I find it really

unique. The game doesn't force you into co-operation and is still playable without it.

There aren't many games like these being developed now, at least from what I can see. Making something completely new and different (to an extent) gives variety to the gaming market.

I used to be really into top-rated games that everyone enjoys. Now, I've been drifting into playing more indie games. Indie games tend to be more unique and ambitious as they try to attract players. They don't have big budgets for marketing. Some aren't great, but the ones that are good tend to be insanely good. Gems like *Lethal Company*, *The Upturned*, *Rain World*, *ENA: Dream BBQ*, and *Outer Wilds* are great examples of this.

Brainstorming Initial Ideas

In terms of genres, I know I want to try to make some kind of co-op game. I also liked the idea of *Rain World*'s PvPvE ecosystem and *Lethal Company*'s take on teamwork, so I'll likely use those two games as the biggest inspiration.

With the ecosystem idea, I want to let players pick either to be passive or to be aggressive – and pivot playstyle when they want to. As such, I believe that the best course of action here is to have two main mechanics in balance, both of which interact with each other. For the passive playstyle, I had the idea of giving the players mobility mechanics to help them avoid danger and combat. They would still be forced into combat, but can escape. For the aggressive playstyle, I decided that resources in the world can be picked up and thrown. These thrown items can stun or damage enemies, allowing passive players to get out in a pinch or aggressive players to kill enemies with enough resources. I could also incentivise this behavior with risk/reward, offering potential a potential reward or loss, potentially making the player to pivot playstyles

Main Inspiration

In terms of gameplay, I want to take inspiration from *Rain World* and *Lethal Company*. As stated many times previously, I really like *Rain World*'s PvEvE and *Lethal Company*'s co-op

approach. I want to make a mix of the two games. I want to primarily focus on Rain World's platforming, with secondary focus on Lethal Company's cooperation aspects, as these are what makes these games great.

There is a mod for Rain World (called Rain Meadow¹) that turns it into a multiplayer game. This is like what I want to achieve, but it has a few issues: the mod itself, while popular, is rather unstable and prone to bugs. It's community made, not official. There is a local co-op mode that is official, but it's locked behind DLC and must be played locally (like couch co-op). Additionally, Rain World was never meant to be a multiplayer game. Its mechanics are designed for single player only. Rain Meadow basically forced online multiplayer into Rain World without much care for the multiplayer aspect. There was no reason for players to work together. If you were alone, it was just Rain World with no changes. For a game renowned for being difficult and unfair, it becomes very easy – even with 3 players, which is what I played it with.

Initial Concept

In this 3D co-op metroidvania, you play as a lowly creature in a harsh and unforgiving ecosystem. You must survive by hunting prey for food and finding shelter, whilst avoiding becoming prey yourself. However, you must be quick. You must return to shelter – or find a new shelter – before the sun rises.

The game focuses on logical movement mechanics - the player can climb and crawl in order to catch their own prey or outmaneuver adversaries. The player must also cooperate (or the direct opposite of cooperate) with other players to make survival easier. Other players can save each other from danger but it usually comes at a cost or a risk, which depends on the threat. It may be worth leaving someone for dead to save everyone else.

Each night (the game's rounds, like Rain World's 'cycles'), the players are forced out of their shelter to search for food. They must find and eat enough to sustain themselves so they can survive another day. However, they must also evade other creatures – predators – who also need to sustain themselves. Although, the predators aren't just hunting the player – they're hunting each other as well. These predators are not the only threat to the players: the sun is insanely dangerous and threatens to kill everything. The players must sustain themselves without dying and find a designated shelter before the sun rises.

Target Audience

Ideally, I want to appeal to teenagers and adults who like games similar to what I have outlined in my initial concept. I don't want to appeal to people who have little gaming experience (most of which are either very young or very old). I don't want something that holds the player's hand every step of the way – I want to make something that people actually find fun and challenging in areas. I also shouldn't try to appeal to people who like

¹ [Rain Meadow mod link: https://steamcommunity.com/sharedfiles/filedetails/?id=3388224007](https://steamcommunity.com/sharedfiles/filedetails/?id=3388224007)

different genres of games – people just have different tastes and, no matter how hard I'd try, not everyone would like it.

Technical Limitations

At this stage in development, I didn't know how feasible it was to make a multiplayer game in Godot – or how good Godot's multiplayer functionality is. This may cause a problem as it may be undeveloped and unstable, confusing to use – or may not exist at all. I'll need to do further research to determine if it is possible to make this game in Godot, or if there's a plugin that makes this easier.

This game may also be too ambitious. It's quite large in scope, and this will be my first multiplayer game. I'll need to manage this, and potentially be ready to cut some content in case of time restraints.

Initial Research Questions

To see if my idea is feasible, I'll need to do some research to see if this is possible. Specifically, in the analysis, I'll look into:

1. How are games fun?
2. How is multiplayer handled in Godot? Is this game possible in Godot?
3. What tools are available to streamline development?
4. How am I going to make my game more accessible?
5. What platform should I release my game on? Steam, itch.io?

So, how can I make an interesting and engaging multiplayer game?

Analysis

Fun in video games

In order to make my game fun, I need to understand what fun is. Fun is subjective, and other people find different things fun.

After some brief searching online, I found this article². This article covers all of the elements of good games, but doesn't exactly cover what makes games fun. It goes over game controls, themes, visuals, sounds, and other aspects of good games. These are aspects of good games, not what actually makes games fun. This article is not great for figuring out what makes a game fun.

² <https://www.gamedesigning.org/gaming/great-games/>


Since different people find different things fun, fun is subjective. You can't please everyone. Due to this, there are different types of fun. Robin Hunicke, Marc LeBlanc, and Robert Zubek attempted to formalise this and wrote the MDA Framework to Game Design³. I was given this last year with my previous game project, so I'm already familiar with it. In the aesthetics section, they covered several different types of 'fun' and game examples of games that use them. They also clearly state that it is difficult to come up with a formula that results in 'fun', due to the different types of fun and what people actually find fun. The authors are all affiliated with universities, and one is a professor of game design. It is unlikely that this is biased. These findings make sense. Everyone finds different game genres less or more enjoyable. I'll need to refine what kind of 'fun' I want to appeal to in my GDD. Since it's largely subjective, it needs to reflect my target audience and what they find fun.

Multiplayer

I want to use Godot for my game because I'm already familiar with it. Multiplayer games are very complex, and Godot doesn't tend to do well with complexity. The best way I can figure out how multiplayer works in Godot is through reading its documentation and figuring out how it works in practice. Godot's documentation gives helpful examples of its systems⁴, as well as resources on how to effectively use it⁵⁶.

These links go to Godot's documentation, and explain how Godot's multiplayer systems work. There is an example script in the documentation that shows you how to initialise a multiplayer connection using its MultiplayerAPI. After that initialisation, you can use a MultiplayerSpawner to automatically spawn nodes for a player when they connect (for example: when a player connects, spawn a player scene and assign it to them). However, without multiplayer synchronisation, things will be client-side only. Using a MultiplayerSynchronizer node, you can sync specific variables (such as an object's position) between all players. MultiplayerSynchronizer nodes only synchronise variables, not function triggers and signals. In order for those to sync, you need to make them RPC functions, which can get called on all clients simultaneously or for a specific client.

I also found YouTube tutorials that provide a visual example of how to implement Godot's multiplayer and explain how it works. The best example that I watched is

 Godot 4 Multiplayer Tutorial Part 1, but there are several others that all say mostly the same thing. Godot's multiplayer approach mostly uses the same approach across tutorials, with the only differences in how you want the project to be structured. This will be great for me when developing the game's multiplayer, alongside Godot's documentation.

³ <https://users.cs.northwestern.edu/~hunicke/MDA.pdf>

⁴ https://docs.godotengine.org/en/stable/tutorials/networking/high_level_multiplayer.html

⁵ https://docs.godotengine.org/en/stable/classes/class_multiplayersynchronizer.html

⁶ https://docs.godotengine.org/en/stable/classes/class_multiplayerspawner.html

I think, for my purposes, this multiplayer functionality suffices for my needs. I'll use Godot for my game because it should handle my design and I'm already familiar with it.

Tools and Addons

On Godot's Asset Library and on Github, there are plugins and addons that extend Godot's functionality. These will usually add a new feature in Godot or add a quality-of-life feature that makes development easier. If I find some that assists my current game idea, development could go much faster.

Since I want to do multiplayer, and I know Godot supports multiplayer, I decided to look for any examples of multiplayer templates and resources. I found a curated list of multiplayer game network programming resources⁷, a Godot FPS multiplayer template⁸, another template with example voice chat capabilities⁹, and another general Godot multiplayer template¹⁰. The first Github link leads to a collection of networking resources. These likely won't be applicable to Godot, as it'll likely handle all of that for me, but it's still useful. The other two links are Godot Asset Library links to template multiplayer games in Godot. These are useful examples that I can re-use code from to make my own development easier.

I don't know how to do proximity voice chat – let alone use the microphone – in Godot. In case I choose to do proximity voice chat for my game, I should check if there are resources that make it easier to implement. I found GodotVoipNet¹¹, which is a Godot plugin, adding nodes that handle voice chat for multiplayer games.

Adding accessibility features to my game will be time consuming. I've tried one before and took a big chunk of time from my development time. I found Godot Wild Jam's accessibility scripts¹², which contains a bunch of examples and templates for accessibility settings.

Accessibility

What are some accessibility features that are useful to impaired players? I should add as many accessibility features that are logical for the majority of players. Adding these accessibility options increases the number of players that can play the game, increasing the size of my target audience. I found this list¹³ that contains loads of accessibility ideas,

⁷ <https://github.com/0xFA11/GameNetworkingResources>

⁸ <https://godotengine.org/asset-library/asset/3658>

⁹ <https://github.com/thegatesbrowser/godot-multiplayer>

¹⁰ <https://godotengine.org/asset-library/asset/3377>

¹¹ <https://github.com/Pieeer1/GodotVoipNet>

¹² <https://github.com/GodotWildJam/gwj-accessibility-scripts>

¹³ <https://gameaccessibilityguidelines.com/full-list/>

such as controller support, input rebinding, colour blind options, text chat communication instead of only relying on voice chat, visual clarity, and maybe an 'easy mode'.

These may take a while to implement because it's usually difficult to create these features. So, I can use Godot Wild Jam's accessibility scripts to make my life easier, as discussed earlier when talking about Godot tools and addons I could use. I could also use pre-made options menus such as Maaack's Options Menus¹⁴.

Another thing I could consider (that is not on the list prior) is translating my game to other languages. Not everyone in the world speaks English. By adding simple language options (even just using Google Translate for simple words and phrases), I can increase the size of my target audience, instead of only those who speak English.

Platform

I'm currently unsure which game distribution platform to add my game on. Should I use Steam, Itch.io, or something else to distribute my game?

Places like Steam are popular and allow the game to interface with Steam through Steamworks, which makes multiplayer easier for the player. However, I need to pay a one time fee of \$100 to publish on Steam, plus signing legal contracts¹⁵. Because of GodotSteam¹⁶, it shouldn't be too difficult to implement this. I should be able to use the player's Steam account name as a player name, and allow the player to use Steam's friend system to join friends automatically. It'll also help market my game. Steam can recommend your game on other user's Discovery Queues, making marketing easier. It also has a very large audience. Steamworks' documentation is publicly accessible, meaning I can learn about it before using it¹⁷.

There are other platforms like itch.io¹⁸ and Game Jolt¹⁹. These give me a free place to sell my game. However, they aren't as popular as Steam, limiting how many people will see and play my game. They also don't give anything similar to Steamworks.

I think I'll use Steam to distribute the game. This is largely because of Steamworks, which provides quality-of-life features to players when it comes to multiplayer. It also helps me market my game thanks to its large audience and game recommendation features.

¹⁴ <https://godotengine.org/asset-library/asset/3058>

¹⁵ <https://partner.steamgames.com/steamdirect>

¹⁶ <https://codeberg.org/godotsteam/godotsteam>

¹⁷ <https://partner.steamgames.com/doc/home>

¹⁸ <https://itch.io/>

¹⁹ <https://gamejolt.com/>

Conclusions

My original question was: *How can I make an interesting and engaging multiplayer game?*

To make it engaging, I have to make it fun. The term 'fun' is subjective. The MDA Framework to Game Design attempted to formalise what 'fun' in video games is by suggesting that you appeal to people who like a particular style of 'fun.' I should consider this when finalising who I want my target audience to be. Even then, fun is subjective, and no-one may find it fun. I may have to go back and change my design according to feedback if this is the case.

In terms of multiplayer, I discovered that Godot is perfectly capable of making a multiplayer game like this one. There are many examples of multiplayer Godot games publicly accessible. Additionally, the Godot Documentation provides clear documentation on how the systems work, and simple tutorials on how to implement them.

Multiplayer games typically take a long time to develop. I have a deadline to get this game completed by the end of the year. To speed up development, or extend Godot's functionality, I looked into adding some plugins and add-ons to my game. The ones I'll likely end up using are:

- <https://github.com/Pieeer1/GodotVoipNet>¹¹
- <https://github.com/GodotWildJam/gwj-accessibility-scripts>¹³

As development goes on, I may end up finding more, but this is a list of the ones I found.

To make it so my target audience is as large as possible, I can add accessibility options. But what options make sense to add? I found this list of accessibility features games can add. Although, there are more that aren't on this list. The features I'll likely end up adding are:

- Translations to other languages
- Controller support
- Input bindings
- Colour blind options
- If I'm going ahead with proximity voice chat: optional text chat for those with no mic or speech impediment

These features may take a while to implement. When I was researching addons and plugins, I found Godot Wild Jam's accessibility scripts. This is a set of pre-made scripts for things like controller rebindings and other options, saving me on development time. I'll likely end up using it, because making these options from scratch will take a long time. I can implement more accessibility features requested through feedback, but I'll need to consider the time cost before.

When development is complete, I'll need a game distribution service to sell my game on. Likely options include Steam, Itch.io, GameJolt, etc. However, I'll likely use Steam, due to

Steamworks. Steamworks makes multiplayer much easier on the player, allowing them to join their Steam friends. GodotSteam exists, allowing me to easily implement this. Steam is also significantly more popular, allowing my game to reach more people.

Game Design Document

Before starting development, it is essential that I have a solidified plan that I can stick to to avoid the aforementioned scope-creep. A game design document is essentially a final blue-print for a game, defining its vision, mechanics, technical requirements, etc. I'll use it as a final plan for my game that I will stick to throughout the rest of development. As things changed throughout development, I constantly updated this to match the new iteration – although ensuring that I wasn't making drastic changes.

Overview

Game Summary/Synopsis

In this 3D co-op metroidvania, you play as a lowly creature in a harsh and unforgiving ecosystem. You must survive by hunting prey for food and finding shelter, whilst avoiding becoming prey yourself. However, there is a greater danger that nothing can overcome...

Genre

- 3D Platformer/Metroidvania
- Survival
- Co-op multiplayer

Player Experience/Game Loop

Each night (the game's rounds, like Rain World's 'cycles'), the players are forced out of their shelter to search for food. They must find and eat enough to sustain themselves so they can survive another day. However, they must also evade other creatures – predators – who also need to sustain themselves. These predators are not the only threat to the players: the sun is insanely dangerous and threatens to kill everything. The players must sustain themselves without dying and find a designated shelter before the sun rises and incinerates everything still roaming around.

Target Audience

My game should be targeted towards people who would like this style of game – it should not appeal to someone else with different game preferences. Since my game is a survival co-op game, I’m not going to advertise it to someone who enjoys more competitive games. I also know how frustrating it is for players to sit through endless tutorials, so I will not include loads of tutorials. This may deter people with not a lot of gaming experience from playing – including audiences that are too young and too old.

Additionally, I want to keep playtimes short to allow for people who do not have long to play games. I also want to keep the game more casual, which means that overly competitive players may not enjoy it. This game is also not really for new gamers – they should have some experience, preferably in first-person games.

To help get an idea of what players I want, I’ve constructed some user personas – fictional people that represent my target audience:

Example of a person who would like this game	Example of a person who would not like this game
<p>Age: 16-45 (players in this range usually have decent gaming experience)</p> <p>Gaming Experience: Competent (previous experience in similar games)</p> <p>Favourite Genres: Cooperative, Survival, Action-Adventure</p> <p>Favourite Games: Left 4 Dead 2, Lethal Company, R.E.P.O.</p> <p>Motivation: Plays games to relax/unwind/have fun with friends</p> <p>Pain Points: Cannot play for hours at a time.</p>	<p>Age: 3-12, 60-104 (too young and too old)</p> <p>Gaming Experience: Novice (little experience in similar games)</p> <p>Favourite Genres: Competitive, PvP, FPS</p> <p>Favourite Games: Minecraft Skyblock, Valorant, Counter-Strike</p> <p>Motivation: Plays games to be competitive/to win</p> <p>Pain Points: Overly competitive, gets bored easily – especially in slower-paced games</p>

Of course, real audiences are not going to be this binary. These are mere examples of the types of player I want to appeal to – there is a lot of leeway within these personas in terms of the types of players that would enjoy this game.

Other than players, there are other stakeholders that I need to address – mainly commercial platforms and project collaborators (if I wasn’t doing this project alone):

Stakeholder	Their needs/tasks	How I’ll address their needs
Commercial platforms (i.e. Steam)	They want a decent product that plays well, is fun, and that they can sell.	Ensure that the game is fully functional and enjoyable for players.
Collaborator (e.g. other programmers and developers)	They need to be able to work efficiently – they do not want to spend an eternity figuring out how my project works, they just want to do their job	Structure my project well, using good code practices to ensure readable code, as well as a well structured project

Concept

Here, I'll go over the MDA analysis – which was highlighted before in the initial research earlier.

Primary Mechanics

My main focus is on the player's movement – climbing and crawling. I want to give players a huge amount of vertical mobility. They need to be able to out-maneuver predators and be able to reach/catch food. Given that the game takes place in an ecosystem, I want to stay away from fantasy/unrealistic abilities. Instead, I want to give the player really good movement abilities that make sense for the world.

Secondary Mechanics

When a predator catches a player, they are still alive but trapped (like Left 4 Dead special infected). They can be rescued by other players. However, doing so would anger the predator even further, making it more dangerous and putting everyone else at risk. The act of doing so may attract more predators as well. At points, players might have to question if it's worth leaving someone for death if it means they (and other players) get to survive.

Fighting is a last resort measure. Players shouldn't be able to attack directly. Instead, they can use items they find around the environment (rocks and other debris) to throw at their attackers. This should stun the predator, not kill (killing should be a lot harder). This is how players free other players from attackers. They lose the items after throwing it and have to go retrieve it if they want to use it again.

Gameplay Overview (Dynamics)

Each night, the players must go search for food. Usually, this food should be in hard to reach places or moving erratically. They have to catch this food and make their way back to their previous shelter, or continue exploring to find a new shelter in a new area (with potentially more plentiful food supplies). Failing to return to shelter with enough food would result in the extremely hot sun rising and incinerating the players.

The sun is not their only issue though: Other creatures (predators) roam this ecosystem, and they're also hungry. The players are not strong, but they can out-maneuver their pursuers. The players can fight back, but this should be a last resort (or for the skilled) – the predators are much stronger than the players.

Aesthetics

The main theme of the game is the struggle for survival – the main gameplay loop and mechanics all revolve back to the struggle of trying to survive.

In areas, I also want the player to feel tense, challenged, and senses of discovery and fellowship with other players – just as if they were a part of a group in that situation.

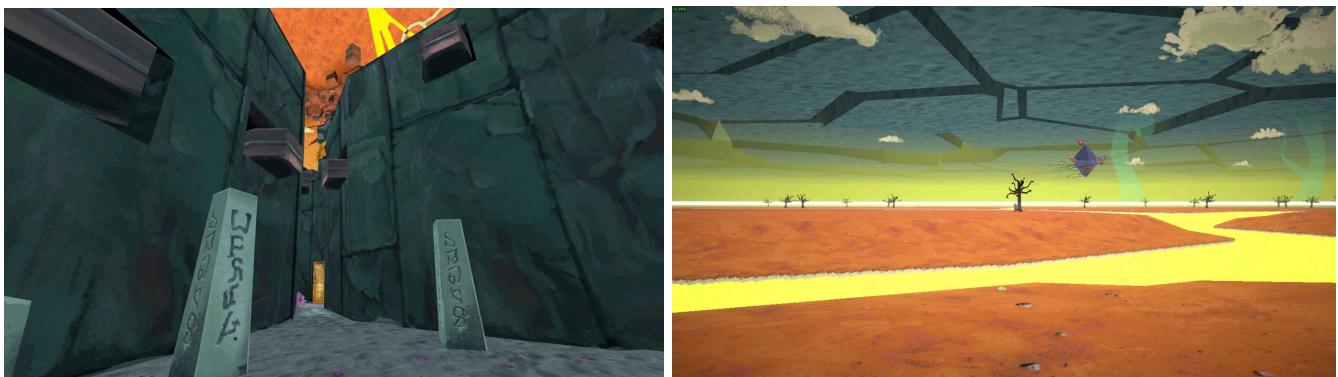
Art and Visual Concepts

Visuals

General Ideas

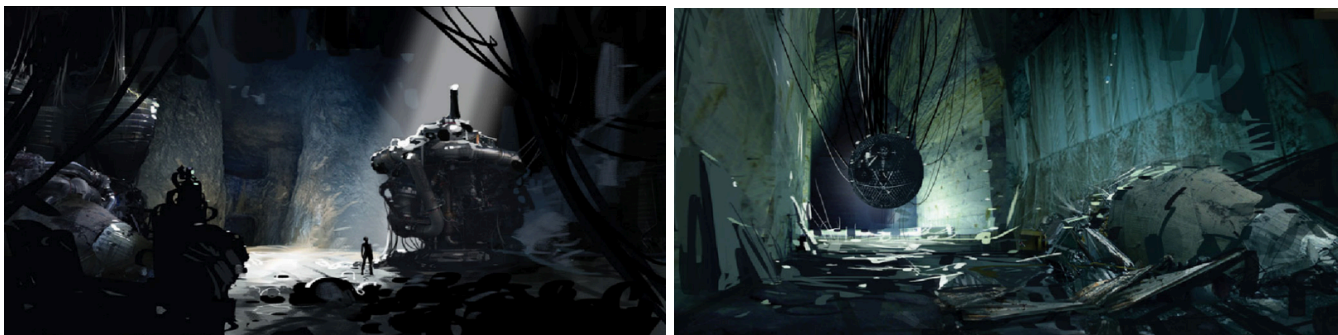
died due to the sun. Thus, herbivores also die out, leaving carnivores and omnivores as the only creatures left in this ecosystem. Due to the scorchingly hot sun, all of the water boils and evaporates into clouds. Thus, at the start of every night when it cools down, it'll be raining like crazy.

In terms of in-game appearance, I want to use a retro aesthetic like ENA: Dream BBQ – with really simplistic, pixelated textures.



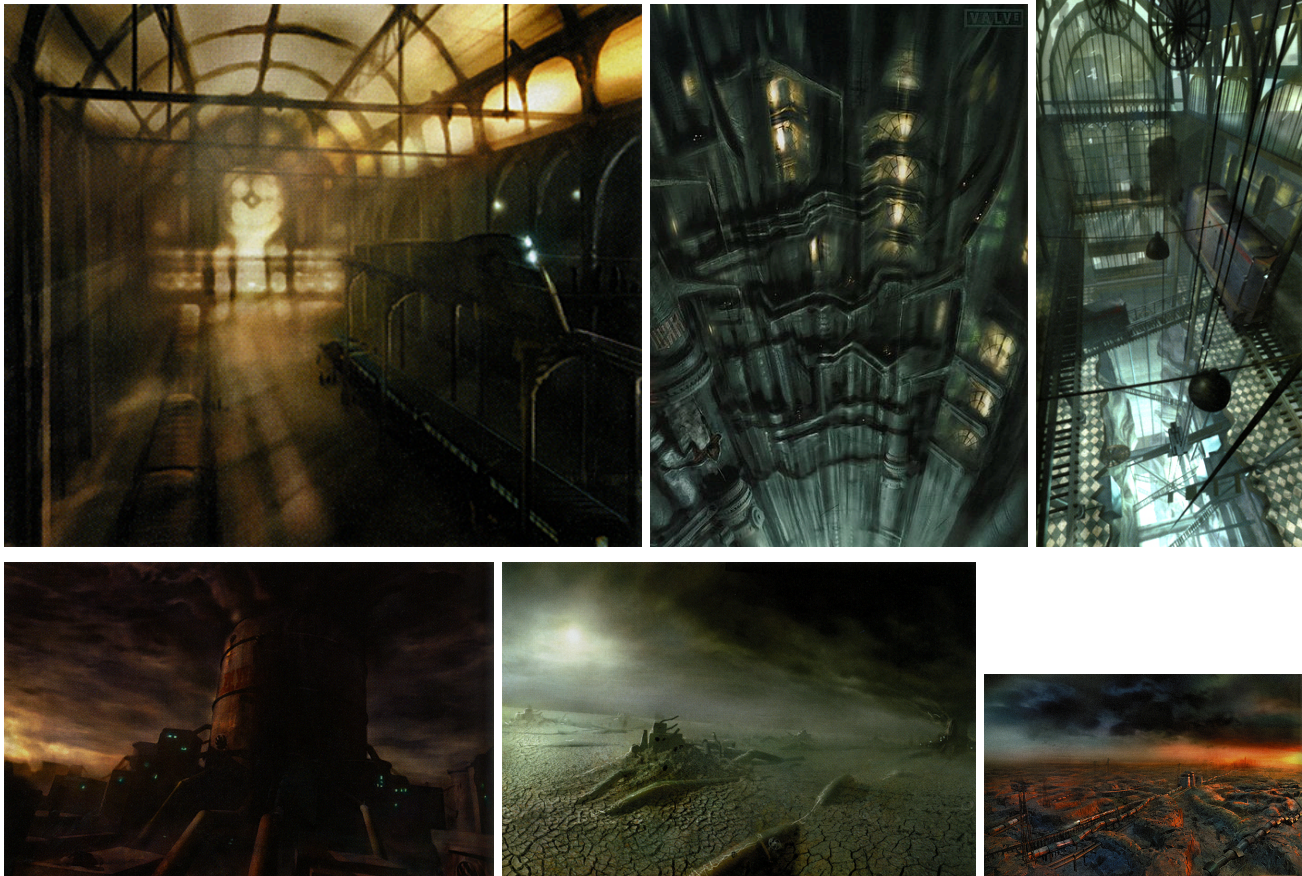
In terms of general game world design, I mainly used the images below as reference and inspiration for the overall appearance of the game world.

Images from Portal / Portal 2's concept art, by various unknown artists²⁰



²⁰ [https://theportalwiki.com/wiki/Category:Portal_2 - The Final Hours Images](https://theportalwiki.com/wiki/Category:Portal_2_-_The_Final_Hours_Images),
[https://store.steampowered.com/app/104600/Portal_2 The Final Hours/](https://store.steampowered.com/app/104600/Portal_2_The_Final_Hours/)

Concept Art for Half-Life 2, by Victor Antonov²¹



I also saw the state of the Clyde Dam as I was driving to Wanaka over Easter. The state of the concrete of the dam inspired me – the old dilapidated look of it. The building itself is mostly greyscale. I could replicate something similar for my game world by using mostly greyscale cold colours and only using colour for highlights and game elements (like players, creatures, visual set-pieces etc).



In terms of colour scheme, to juxtapose the bland nature of a greyscale environment, I decided to use a complementary

²¹ [https://en.wikipedia.org/wiki/Viktor_Antonov_\(art_director\)](https://en.wikipedia.org/wiki/Viktor_Antonov_(art_director))



colour scheme. A complementary colour scheme is two colours directly opposing each other on the colour wheel that provide a harsh contrast between the two colours – exactly what I need. I picked a blue and orange colour scheme, as these two directly conflict with each other on the colour wheel, and generally look good together. I can also convey purpose with these colours due to their colour connotations (e.g. blue conveys passive elements, while orange conveys dangerous elements).

Characters and Creatures

For the player's design, I was initially thinking of making them look like Rain World's slug cats. Particularly, I liked the design of the Rivulet slugcat from Rain World's Downpour DLC, as it took the original slugcat design but still made it look unique.

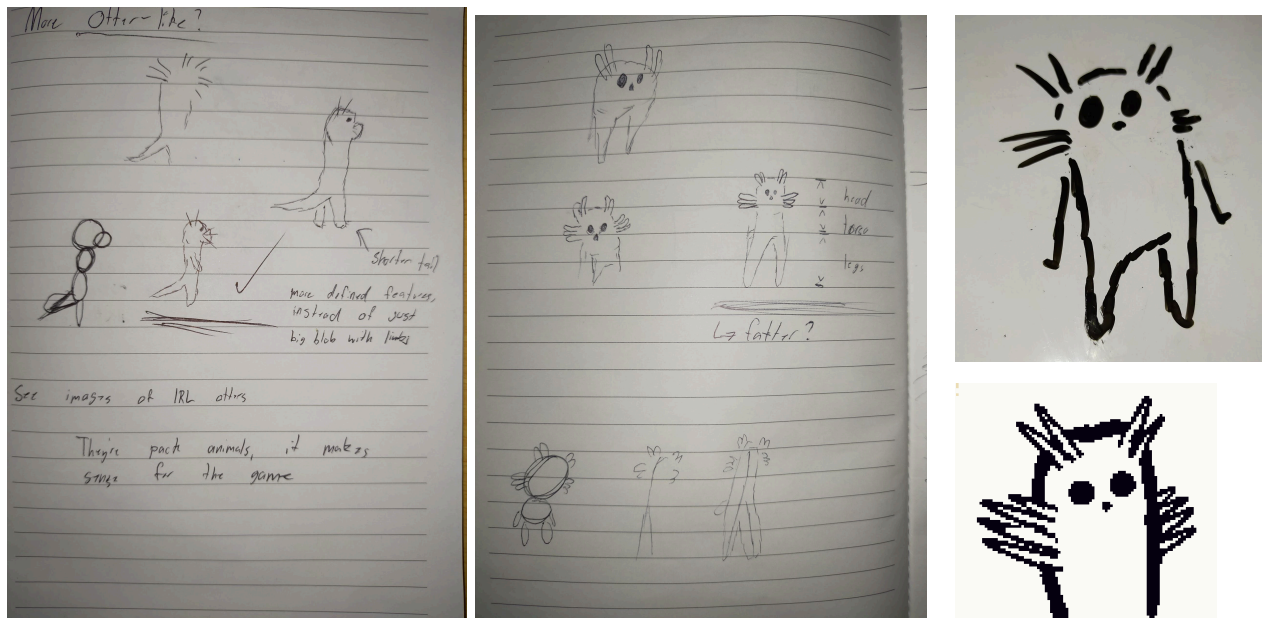
Rain World's Slugcat (left), and its Rivulet Slugcat DLC variant (right)



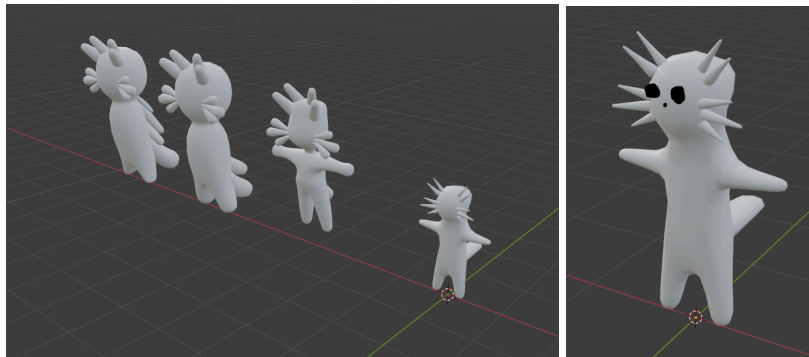
I eventually got the idea of making them look like otters (largely because otters seem to match with how I want the player to act in game, and their place in the food chain). I also wanted to keep the large whiskers from Rivulet.



These are some sketches of what the player eventually turned into, based heavily on real world otters and the Slugcat from Rain World.



From these, I made some 3D model blockouts for a final in-game 3D model of the player. I generally prefer these versions as I'm a lot better at 3D art than 2D art.



I decided to name the player creatures 'skrunks', after the original placeholder name I gave the project's repository (originally titled 'Project Skrunk').

Storyline/Lore

This is a storyline I came up with for the game, which is likely to change and get revised as the game's development goes on. This is also more of a very rough example of how the game's story plays out in game.

The player skrunks start off unassuming, roaming around as-per-usual in their lives. At the start of the game, we are shown their previous peaceful past – and what happened to the sun. The old society disappeared around this time. The skrunks enter Offshoot in search of peace, but are guided toward the central tower by the Assistants – a small friendly creature resembling an eyeball. The skrunks just want a permanent place to settle down.

There is no peace there, however. They are forced to keep roaming – and they can't return back the way they came. The higher floors in the tower seem like the luxurious remnants of a previous upper-class society. The subterranean basement is where all the machinery and technology lies – some of it spanning for miles. Dotted around the central tower are rooms bearing the same insignia as the skrunk's safe rooms – most are sealed and crossed off, while some are open. Inside these rooms are the chained bodies of the previous society with unknown machinery gnashing out of them. They look dead – like ghosts. Despite the scene, the player can still rest here. Doing so triggers a special encounter.

Upon falling asleep, the players are sent to a mysterious utopic dreamworld, inhabited by the civilisation – they retreated here because they hated the state of the real world. If they see a skrunk, they try to kill it, kicking them out of the digital dream world. Exploring and sneaking far enough leaks you to the Administrator – an AI left to manage this world. It guided you here using its Assistants (the maintainers of the equipment in the real world). The administrator tells the skrunks that life here is miserable, and acknowledges your plight. It offers you a choice: go to the basement and destroy it all, or use this utopia to create a fake utopia for yourself to live in forever.

Both of these endings are inherently bad. Shutting it all down causes everything to shut down, meaning your saferooms stop working, and so the ecosystem is finally put out of its misery. Life can begin anew here. Living in the utopia isn't great, as the skrunks are now trapped there forever and grow bored, much like the old society did. They eventually lose the charm of the utopia, go insane, and yearn for death – just like the old society.

The whole story revolves around the idea that we look to technology to distract us from the pain of the real world.

Audio Music

I want to keep music to a minimum. I want to get better at sound design overall, so leaning for a SFX oriented approach to game audio would be best.

I want to do something like Rain World's threat music. Rain World's threat music is a special music track unique for each region that grows in intensity depending on how much danger the player is in. See [Rain World | Threat - Outskirts - Layered](#) for an idea of what I'm going for. However, this is actual music, which I just said I want to keep to a minimum. Instead, I'll try to make the music sound like it's being produced somewhere from the environment.

One unique thing I will do is have special music play when/before the sun rises. I want this to be like the track [End Times](#) in Outer Wilds (which plays just before the sun explodes in-game)

Sound Effects

I want to get better at sound design in general, so I'm including this section. The soundscape should be quite industrial and deep, to convey the history and danger of this place the player is in.

Occasionally, there should be sounds that just cannot be identified as something that the player knows (strange sounds).

Game/User Experience

User Interface

The in-game HUD should be as minimal as possible. The only thing that the player should get is a timer to see how long until sunrise, and how much food they need to survive (Rain World). That's it. The HUD will not include any status of the other players (like Left 4 Dead does).

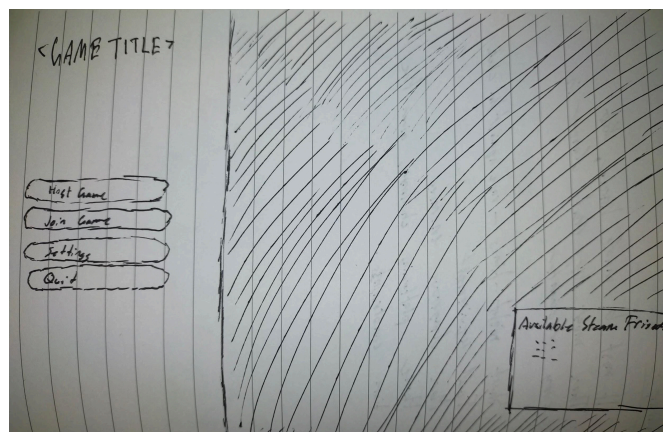
To the right is an example of Rain World's HUD. This is exactly the type of thing I want my game's HUD to look like.



Title Screen

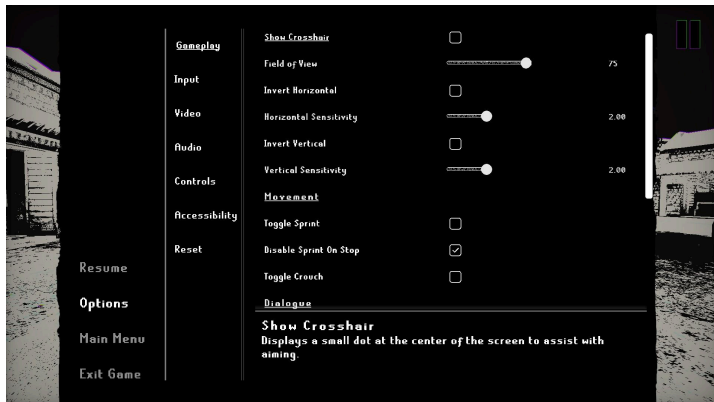
For the game's title screen, outside of actual gameplay, I came up with this sketch for its appearance and layout.

I decided to align the buttons along the left and leave most of the available area for background to set the scene of the game. Along the left,



there are options to host a game, join a game, open the game options menu, and quit the game and close the program.

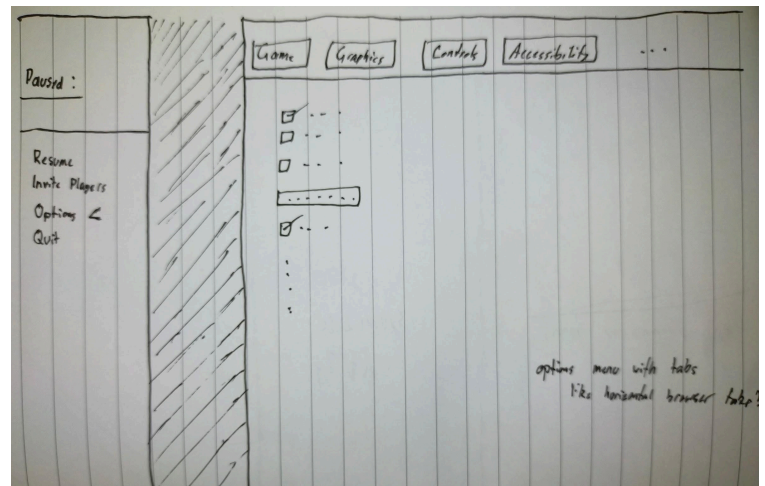
Options Menu



For my options menu, I quite liked the layout of the options menu from ENA: Dream BBQ. On clicking the options button, the minimal pause menu expands to fill the whole screen with the options menu. One problem I have with it is it's only vertical lists – nothing special.

I decided to make another concept based on this, but with horizontal tabs instead of vertical, which is to the right.

Shifting the submenus to the top makes the options menu resemble a web browser. Since almost everyone uses a web browser regularly, it seemed like a smart idea making it consistent with what users already know.



Controls

Controls should be simple, but given the vertical mobility options, I will need to modify some things.

For all horizontal movement, WASD should be used. WASD is used commonly for PC games for general movement. Same with the Spacebar and Control for jumping and crouching respectively. However, Rain World's climbing uses W and S for upward and downward climbing respectively. That works well for Rain World because it is 2D, but I'm using 3D – W and S are already in use (which are used for walking forward and backward respectively). For climbing up and down, Shift and Control will be used instead. These controls for vertical movement came from Outer Wild's ship flight mechanics, where Shift and Control are used to fly up and down.

I could include controller support but, since this is a PC game, this is not the highest priority. I do not use a controller for games and don't know what kind of control scheme would be best for this game. If I end up deciding to implement controller support near the end of development, I'll need to do some research on how to make understandable controls for controllers – especially with the vertical climbing.

If these controls appear to be incorrect for my game, I will figure this out during playtesting. From there, changes will be made to make it easier to control – likely using playtester input to make them the best possible.

Relevant Implications

I'll address some relevant implications to ensure that any issues that may arise from my project are addressed way before development starts.

Aesthetics

Games should be fun and enjoyable for players. If players aren't having fun, they'll simply put down the game and go play something else. I've outlined a solidified plan/idea for my game, and what makes it fun (outlined in my concept earlier). Fixing this in place early is essential, as it defines the scope of my game – I know exactly what it is and what makes it fun. To ensure that the game is fun and sticks to its scope, I'll playtest frequently to ensure that it is fun for players – if playtesters aren't enjoying it, something has to change.

Functionality/Usability

No-one wants to play a broken mess of a video game. I need to ensure that the game is fully playable without broken mechanics, game crashes, or other issues that ruin the player's experience. If the player is experiencing these issues, they likely will become frustrated and will stop playing. To address this, I'll actively debug the game to reduce the possibility for bugs and issues. I'll also utilise the playtesting sessions to see how the game holds in the hands of actual players, and fix issues found from them.

Legal/Ethical

Multiplayer games involve exposing players to the open internet. This, of course, has cyber-security issues as malicious actors can send malicious data to other players, which can do some dangerous stuff to players. Naturally, players should be safe when playing, so I'll need to design key networking systems to avoid . Although, no matter how hard I try, people will find a way to break it, which leads to:

Sustainability

As new security vulnerabilities are found within my game code, I need to be able to fix it in a timely manner, or allow a maintainer to fix it. Failing to fix any security vulnerabilities as they are found may cause the game to become too risky for players to play without putting themselves at a cybersecurity risk. As such, I should format code nicely and in a way that's easy for me to understand, but also easy for a maintainer to understand my code and fix issues.

Requirements

For my outcome to be considered 'complete,'

- My game must be a survival/exploration game.
- My game must be cooperative game
- My game must be difficult
- My game must be engaging and fun for my target audience
- My game must be easy to learn
- My game must have intuitive controls and accessibility options
- My game will be released on Steam

To complete each of these, I've broken each of these down into attainable measures:

- For my game to be a survival/exploration game,
 - The player must eat food to survive
 - The player must explore to find food
 - The player must avoid danger and survive
- For my game to be cooperative game,
 - The player must cooperate with other players to survive
 - The players can put themselves at risk to save other players
 - The game will be made more difficult failing to cooperate (It won't make the game impossible, just minor punishments. Shenanigans are fine)
- For my game to be difficult,
 - The player will be weak and fragile
 - The player can be easily killed if they are not careful
 - Cooperation is encouraged because it makes survival easier
- For my game must be engaging and fun for my target audience,
 - I will appeal to my desired target audience
 - I will playtest often to ensure that my game is fun – if not, make design changes to correct
 - I will ensure that my game won't be too difficult and frustrate players, causing them to quit
- For my game to be easy to learn,
 - My game will have simple and logical mechanics that are easy to understand
 - My game's loop will be simple to understand
 - The player shouldn't require prior context to pick up and play at any point

- For my game must have intuitive controls and accessibility options,
 - My game will have controls that make sense
 - My game will have several accessibility features such as input rebindings and colour adjustments to assist those with colour blindness
 - My game will have several translations to other languages, making the game accessible to non-English speakers

Resources

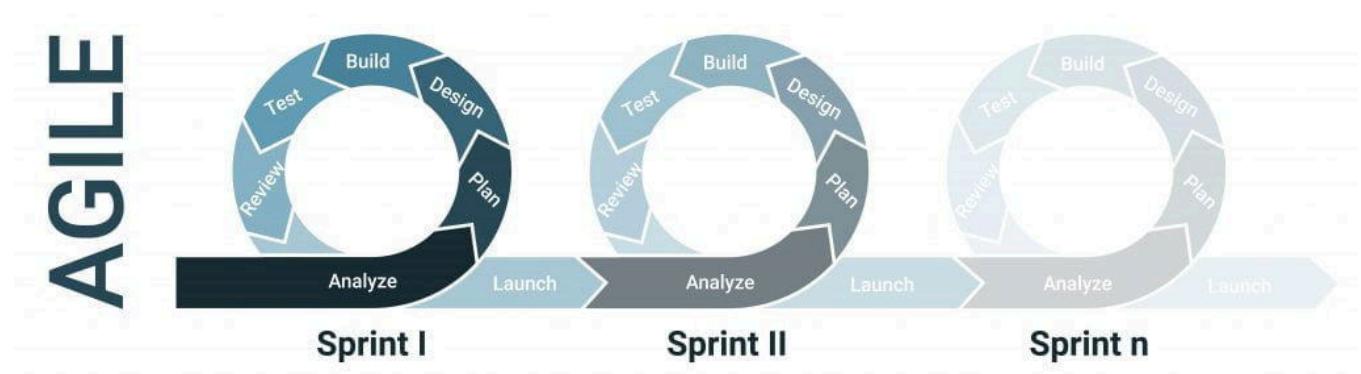
Godot Engine (+ plugins and addons)

I already have loads of experience with Godot, and I know that it supports multiplayer from the previous investigation. There are also a lot of resources and plugins I can use for the game and for the multiplayer systems – which were highlighted earlier from prior research (e.g. GodotSteam).

Miscellaneous

I will also likely use Blender, Krita, Audacity, LMMS, and Git/Github as other software. Blender is a 3D modelling suite that I'll use to develop my game's 3D assets and levels. Krita is a 2D drawing program which I'll use to create textures for the 3D models, and create any UI required. Audacity and LMMS are audio programs, handling sound design and music respectively. Git/Github will be the version control system that I'll back up my project to, mainly to avoid data loss and to revert breaking changes if necessary.

In terms of development approach, I'll use an Agile approach. Agile works by breaking the stages/cycles of development into 'sprints'. Agile allows my development to be adaptable and focused on user feedback. To obtain user feedback, I'll playtest at the end of each sprint to pick up on bugs and issues on a regular basis. This will all be kept track of using a kanban board, hosted as a Github Project²².



²²

<https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects>

An alternative method is Waterfall, which involves following a linear, sequential method to project management. This does not allow for much adaptability like Agile does, so I'll be using Agile for development.

Development Risks

Scope Creep

Scope creep is the growth in a project's scope over its lifetime, which causes a project to become far larger than anticipated and causes development time to sky-rocket.

My game is already very large. Increasing its scope further would inevitably lead to running out of development time. I should use what time I have to only develop the features I have planned. I simply cannot get distracted, otherwise the final game may not be completed by the end of this year. To try to avoid this, I will create a solidified plan now to explicitly define what is and is not a part of this project.

Time Management

I do not have a lot of time to develop this game that is already large in scope. I'll counteract this by using Github Projects (which is a Trello alternative) to manage my time and what I'm developing at different stages of development. This visualises my progress and lets me easily see what needs to be done.

Development Timeline

I can see that I am very pressed for time with this project. To counteract this, I will only focus on the core design. I cannot let scope creep have an influence on this. In case I do not have enough time, I may have to simplify my game design or cut planned features.

The final deadline I have set for this game is somewhere in October 2025 (presumably early October, with some leeway). I began this project sometime in November/December 2024.

Sprint 1, and the game's overall concept, should be done by January. I can extend this into February if needed (which I might do because this game is considerably large in scope). This allows me to begin playtesting as soon as school comes back. Sprint 1 should be a prototype version of the final game, with the gameplay loop and mechanics functional. Given the multiplayer logic, I expect this sprint to take the longest.

With sprint 2, I want to get it done by June 9. This gives me plenty of time to implement the game's art style, any game design changes from playtesting, and more content.

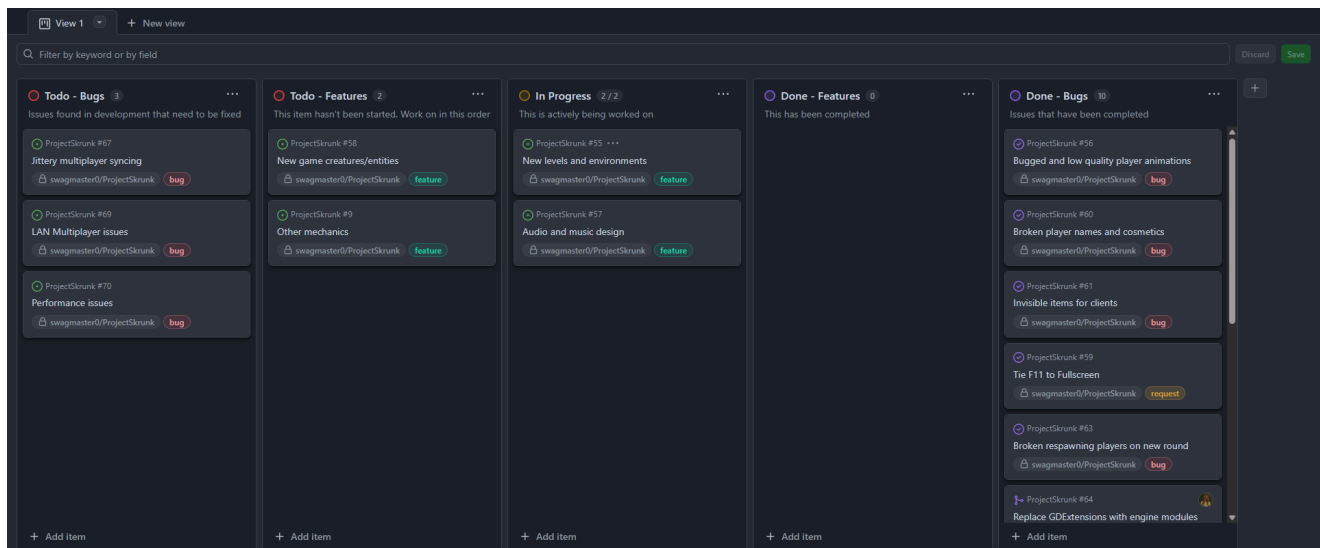
For sprint 3, and the time this project is due, is sometime in October. This deadline gives me loads of time to finalise my game before the final submission.

Development

After defining my game, it's now time to begin development. What follows are all of the development processes and logs, highlighting key areas of development.

For my development process, I used an Agile approach. I broke development into 3 main sprints – sprint 1 focuses on making a working prototype, sprint 2 focuses on improving the prototype from feedback and implementing game art, and sprint 3 prioritises on polishing and fixing issues before the final release.

For each sprint, I use Github Projects to keep track of project management – which is a kanban board. Each item in the project is its own issue in Github (even if it's a feature addition). In code commits, I get to reference commits in the commit history, which helps clarify which commits are working towards what features/bugs. An example of a Github Project used throughout development is below.



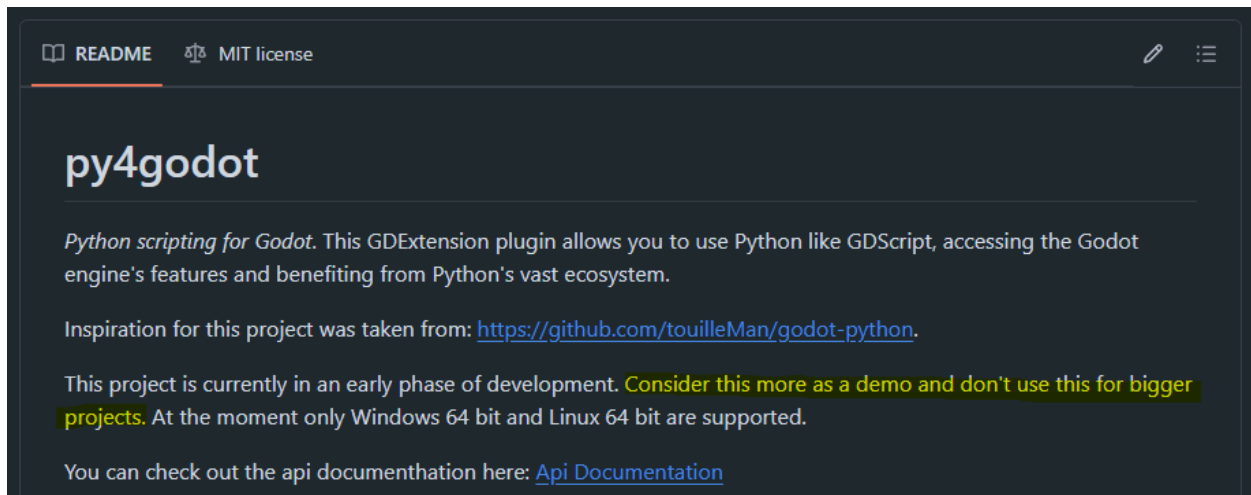
Programming Language

Before beginning development, I already questioned the programming language to use. I've decided to use Godot 4, since I'm used to it already. However, Godot's GDScript language is limited to just Godot. I want a language that has a wider use-case, not just GDScript.

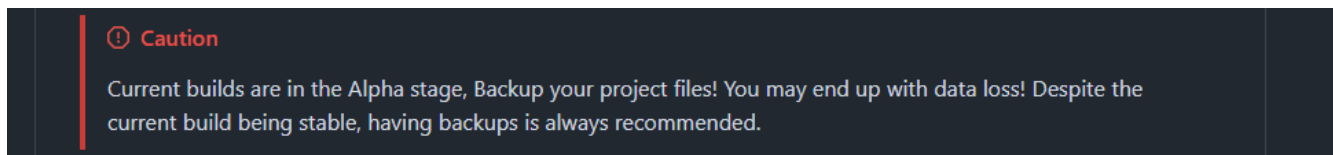
Another built-in option is C#. Godot's 'Mono' builds allow for this. However, these require a .NET SDK, which usually has to be installed externally. Also, Godot cannot export projects for Web platforms and compatibility for mobile platforms is experimental.

Since Godot is open source, people can create addons that allow for the use of external languages. Namely, Python and C++ support were added by the community. However, the addon creators discourage usage for main projects as they are early in development. Development, namely exporting your project, can also be made extremely difficult

Python²³



C++ (Jenova)²⁴



Due to the experimental nature of these plugins, I'm more-or-less forced to use GDScript or C#. Since Godot is built for GDScript and every Godot tutorial is for GDScript, I'm likely forced to use GDScript at some point in development. Since I still want to use another language, I'll come up with a compromise: I'll use a mixture of both C# and GDScript in my game, although I'll likely end up using mostly C#. The downsides of C# are mostly mitigated with my project, as I plan to export for desktop platforms – not the web or mobile.

Sprint 1

This sprint will be the bare minimum of what I want out of this game (a basic prototype).

²³ <https://github.com/niklas2902/py4godot>

²⁴ <https://github.com/Jenova-Framework/J.E.N.O.V.A>

Generally, this sprint will accomplish:

- Main player implementation, including player mechanics like crouching and climbing
- Enemy design and implementation
- Level design and implementation
- Game logic and core game loop, and saving game
- Multiplayer functionality

I aim to have this sprint complete by January / February 2025 (this sprint started November 2024). I feel like this may take a bit longer though, due to the pain of multiplayer networking.

Development

Player

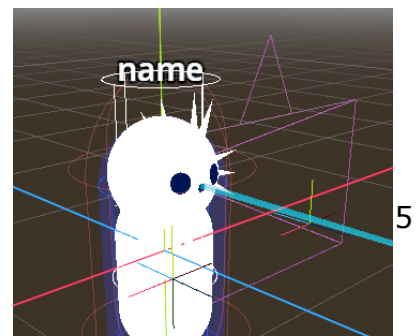
Instead of using a `CharacterBody3D` for my player, I originally used a `RigidBody3D`. This is because I wanted a more physics intensive player, at the cost of added complexity – like *Rain World*'s active ragdoll player. At some point in development, I figured it wasn't worth the added complexity and converted it back to a `CharacterBody3D`.

For my last game project in Godot, my player script was ~600 lines long and most of the code was thrown into the `_PhysicsProcess()` function without much organization. For this game, I'm taking a different approach. Everything is still in the `_PhysicsProcess()` function, but everything is sorted into handler functions, which handle player input. These handler functions are triggered by `_PhysicsProcess()`, which then triggers the main functions (e.g. `Move()`, `Jump()`, and `Crouch()`). I'm very glad I chose this deliberately organised method, as the player script ended up being over a thousand lines long.

My main focus for the player is their mobility, so I'll mainly discuss this.

The player's jump is handled in `JumpingHandler()` which is triggered every frame within `_PhysicsProcess()`. The handler function reads the player's input and triggers the corresponding function depending on their input. For example, if the player pressed the jump key and the player is standing, they do a regular jump. If the player is crouched when they press it, nothing happens. This is because of the player's leap – if the player is crouched and they hold the jump key, they charge up a leap. Releasing the jump key after 0.5 seconds causes the player to leap forward a good distance.

Like jumping, the player's climb is handled in `ClimbingHandler()`. This checks the player's input and if there is an available surface to climb on. If they can climb, it runs the `InitialiseClimb()` function using the `Path3D` node of the climbable as a parameter, which sets up the player's



annoying to set-up in the editor, so I may consider an climb. The player's climb works by following a `PathFollow3D` node attached to the `Path3D` climbing surface. This is alternative option in the future. The player uses Shift and Control to climb up and down, and WASD to climb horizontally. If the player presses space when climbing, they leap off the climbable, returning to their default state.

Enemies

For my game's creatures and enemies, I decided to make a base class for every entity in the game. This base class inherits from `CharacterBody3D` and the script itself contains functions and variables that all enemies will need. All entities/enemies will extend from this base class. The player also inherits from this class as I want to keep things consistent.

For last year's project, the way I handled keeping all enemies consistent was a bit odd. I used a custom node that I called the 'info' node, which handled health and other enemy-specific stuff. This was a very round-about way of doing it as if I wanted to deal damage to an enemy, I'd need to look through its children, find the info node, and deal damage to it. With my new approach, the 'info' node *is* a part of the entity, and I can call entity-specific functions with only access to the root entity node – no need to search for a specific child, making it easier for me.

The class itself consists of static calculations for finding and navigating to other entities, and handles the entity's health, stunning, and death mechanics. Health, stunning, and death behavior is relayed to the inheriting entity via signals, and is sent across RPC to ensure that all players see the change (more on that in Multiplayer).

The portion of code that handles the health, stunning, and death behavior is below.

```
CSharp
public void Stun(Vector3 knockback) {
    // Relay this stun to all players
    Rpc(MethodName.StunHandler, knockback.Normalized() * 2);
}

[Rpc(MultiplayerApi.RpcMode.AnyPeer, CallLocal = true)]
private void StunHandler(Vector3 knockback) {
    EmitSignal(SignalName.Stunned, knockback);
}

public void Damage(int damage)
{
    if (WillDieFromDamage(damage)) {
        Rpc(MethodName.SetHealth, 0);
        Rpc(MethodName.DeathHandler);
    }
}
```

```

    }
    else Rpc(MethodName.DamageHandler, damage);
}

public bool WillDieFromDamage(int damage) {
    if (Health - damage <= 0) return true;
    else return false;
}

[Rpc(MultiplayerApi.RpcMode.AnyPeer, CallLocal = true)]
private void DamageHandler(int damage) {
    Health -= damage;
    EmitSignal(SignalName.Damaged);
}

[Rpc(MultiplayerApi.RpcMode.AnyPeer, CallLocal = true)]
private void DeathHandler() {
    EmitSignal(SignalName.Death);
}

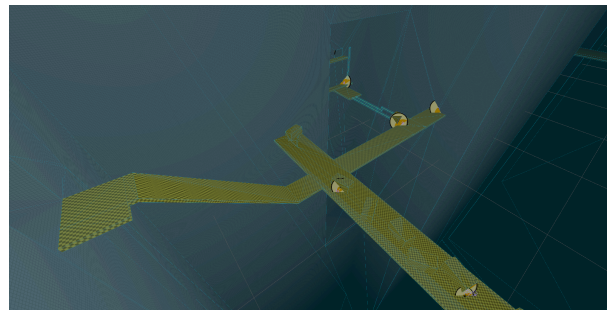
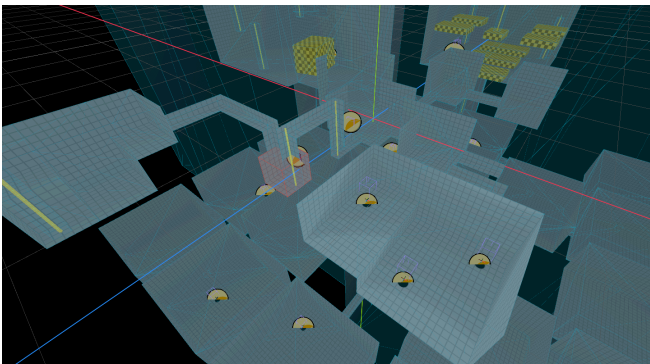
[Rpc(MultiplayerApi.RpcMode.AnyPeer, CallLocal = true)]
public void SetHealth(int health) {
    Health = health;
}

```

Levels

Handled by [Level.cs](#).

I only made 1 reasonably large level for my game's prototype. This was made using Godot's CSG nodes, as this allowed me to easily test the level within the engine (e.g. ensuring jumps are possible). Screenshots of the developed level are below.



The level itself is structured so that the players start in a forced 'tutorial' area, where game mechanics are taught – such as movement and spawn rooms. Then, the game opens up a bit and lets the player roam free. All paths that the player can go all lead to the same area, which leads to the prototype's end.

The game uses 'spawnrooms' for spawning, round success calculations and saving. At the start of each round, the players spawn in the saved spawnrooms (or use the default spawnroom). In order for the player to save and have a successful round, all players must find/return to a spawnroom after having eaten enough food.

However, when I was developing the levels, I was experiencing major performance issues in the editor – which would inevitably get reflected in the final game.

Godot Versions

In the Godot version at that time, Godot's CSG library was rather unoptimised and wasn't being maintained. I also read in a Godot blog post that v4.4-dev6 was implementing a new CSG backend called Manifold (see their blog post about features and changes implemented in v4.4-dev6²⁵). I tested this out with the current project I had, and saw a big performance increase (there was still some struggle, but it's significantly better compared to the old CSG implementation).

I could use Godot v4.4 development builds moving forward and use all of the new features being added in v4.4 early (like the built-in Jolt physics, CollisionShape3D colours, favoriting editor items, better debugging for running games, etc). However, this runs the risk of file corruption and other annoying bugs that come with using an unstable Godot version. At the time of writing this, the latest release is v4.4-beta1, which means they are not adding any more new features and are solely working on bug fixes and stability.

I'm thinking I will use the pre-release builds of Godot. I'm using Git, and can revert changes in the event of a file corruption. With any bugs I find, I can report them to Godot engine contributors and hopefully get them fixed (or potentially fix them myself, since the engine is open source). When the stable build of 4.4 eventually releases, I'll begin using it.

Game Logic

Game logic is largely handled by a single node called *Loop*, handled largely by the script `Loop.cs`. Some logic is also handled by other scripts, like `Global.cs` (which handles game saving). Before the round starts, the host spawns in a sealed spawnroom and waits for players to join (kind of like a 'waiting room'). When the game is first initialised, all of the

²⁵

<https://godotengine.org/article/dev-snapshot-godot-4-4-dev-6/#replace-internal-csg-implementation-with-manifold-library>

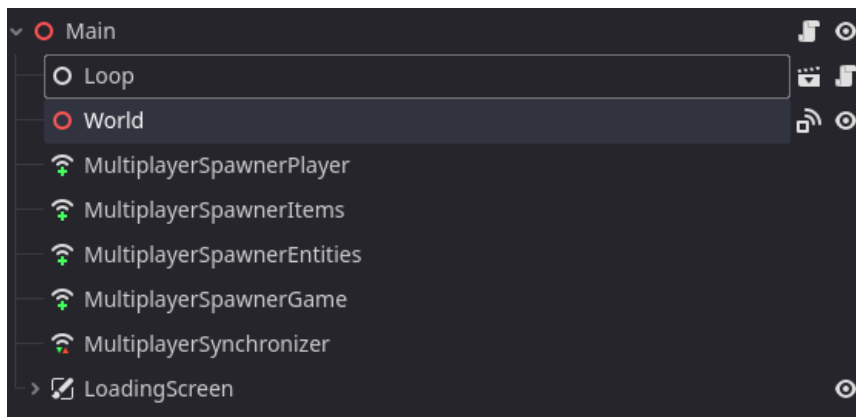
game's objects and entities get spawned and scattered around the map. When the host wants, they can start the game manually, unsealing the spawnroom and starting the round. The game's round timer is also started at this time – if they do not manage to get to a spawnroom in time with enough food, they begin dying when that timer runs out.

If the player successfully saves using a spawnroom, the game gets reset and the game loop repeats from the 'waiting for players' step. The game is saved for the current spawnroom that the player used. However, if all players die, the round gets reset and they get sent back to their old saved spawnroom.

Multiplayer

For multiplayer, I handled it using Godot's high-level networking nodes, as discussed earlier from the analysis about Godot's multiplayer implementation. **MultiplayerSpawners** are set-up within the main game scene (**main.tscn**), while **MultiplayerSynchronizers** are used within the game's entities, items and other stuff that needs syncing over the internet. Throughout my scripts for these items, I heavily use RPC functions, which syncs the function call over for all players.

The node structure of my game's main scene is below – the root **Main** node contains the base networking code, the **Loop** node contains the **Loop.cs** script which handles the game's logic, and all of the multiplayer nodes are visible.



The only thing I had to manually create was the initial connection between clients. However, when I tried playtesting it over the internet, it did not work – it only worked reliably with LAN. For this to work over the internet, I'd need to open a server manually on my router (which is likely too difficult for typical end users). From this point, I decided that I needed to change my approach.

I found some options to work around this, which are as follows:

- UPNP (Universal Plug and Play)²⁶

²⁶ https://en.wikipedia.org/wiki/Universal_Plug_and_Play

- Simple solution that's already built into Godot.
- Poses a serious cyber-security threat as it does not authenticate users²⁷ – it's usually disabled by default on most routers for security reasons. Building a multiplayer game that relies on UPNP is heavily discouraged.
- Steamworks API (as discussed earlier)
 - Use Steam's servers as a relay for P2P connection, using GodotSteam to implement. Dodges the requirement of port forwarding, as Steam handles that. I was already planning to ship on Steam regardless.
- Netfox²⁸
 - Uses a server as a relay to connect clients using Netfox. Requires a server as a relay run by me for the game to work.

Due to the cyber security threat that UPNP creates, I decided to ignore it. I decided to try both Steam and Netfox. I created two different git branches to test them.

With Netfox, I never ended up getting it working. I decided to simply stop trying to get it working, largely because I'd need to host my own server. Steam's servers would be much more reliable than the singular server that the developer had public. This is quite similar to the approach that Steam has, except I don't have to host my own server.

In the end, I decided to use Steamworks networking for my game. Steamworks lets me use Steam's servers as a relay for my game, essentially using Steam as a middle-man for connection. I used GodotSteam to implement this (which allows me to access Steamworks API within Godot) and SteamMultiplayerPeer (using either Expressobits's SteamMultiplayerPeer²⁹ plugin or GodotSteam's SteamMultiplayerPeer³⁰ engine module, both of which allow me to use my old networking system using Godot's high-level networking nodes). This was likely the best decision, as I was already planning on releasing it for Steam.

Other Additions & Bug Fixes

Debugging multiplayer with Steam was a bit of a pain. Before, I could have two instances of the game open at once and test multiplayer with that. With Steam, however, I had to use two computers (or a virtual machine) to test my game – you couldn't connect if both players have the same Steam account, and you can't be signed into two different Steam accounts on the same computer.

To solve this, I re-implemented my original design of LAN connection, as an alternative connection method. I created a duplicate of the `main.tscn` scene which extends from its

²⁷ https://en.wikipedia.org/wiki/Universal_Plug_and_Play#Authentication

²⁸ <https://github.com/foxssake/netfox?tab=readme-ov-file#readme>

²⁹ <https://github.com/expressobits/steam-multiplayer-peer>

³⁰ <https://codeberg.org/godotsteam/multiplayerpeer>

original script, but overrides specific functions to use LAN networking instead of Steam's networking. Steam is still required, as the game uses your Steam name.

Feedback

A collection of various playtesting videos of my development can be found at [▶ Sprint 1 - Prototype](#). These are all playtesting sessions with friends, with varying levels of success (the success rate tends to increase with newer videos). With these playtests, I was constantly iterating on the previous test to get the game to a fully functioning state. The early playtests were focused on getting the game working and stomping bugs, while later playtests were spent figuring out the gameplay (which is only possible when the game is in a working state).

I also uploaded my prototype game to an inter-school game jam on Itch.io. Unfortunately, I did not get any feedback from this.

Sprint 1 Reflection

This sprint has taken far too long – but that's to be expected. This is a multiplayer game – it's meant to be a complex development cycle. In the end, I finally got a fully functional prototype in May. With future sprints, if I want to meet my intended timeline, I need to work faster

Sprint 2

This sprint will house all of the changes and bug fixes from sprint 1's playtesting. Afterwards, I'll implement all of the art elements to the game. This sprint likely won't contain major changes towards the game.

In this sprint, I will:

- Fix bugs and game design issues from Sprint 1
- Finalise concept art + Implement art into game
 - Add new enemies, turn into a full game.
 - Add new levels
 - Add creature designs
 - Add player designs
 - Add level designs

This sprint should be done by June 15, 2025. I officially ended sprint 1 on 12 March, which was longer than I anticipated. This date should give me sufficient time to get this done.

Development

Bugs and Other Issues

Unsurprisingly, most of the issues that arose were due to multiplayer. There were also issues with the microphone of other players in-game.

Microphone Issues

The microphone of other players in-game was very poor quality, and constantly created ear-hurting popping noises. I believe this was caused due to performance reasons. The voice data was being sent and received by an RPC function using a reliable transfer protocol, meaning that it ensured that the data was received on the receiving end. This is reliable, hence the name, but also really slow and not recommended for sending multiplayer game data, as they're supposed to be quick and snappy. I set the transfer mode to unreliable, which means it just spits the data at the client as fast as it can, not caring if the data is actually making it. This is theoretically faster and minimised the problem slightly, but did not solve it.

I then thought that it was something to do with Godot's microphone implementation. Since I was already using the Steamworks API, I decided to replace my existing system with Steam Voice. This wasn't too hard to implement into the game, and actually simplified the voice system. However, it still had the same problem.

I eventually decided to profile the code using Visual Studio to look for bottlenecks in the code (where the computer is taking a long time to calculate something). Below is a portion of the function that handles playing the voice data on the clients end.

CSharp

```
// Iterate through all of the audio frames available
for (int i = 0; i < (int)Mathf.Min(_playback.GetFramesAvailable() * 2,
VoiceBuffer.Count); i++)
{
    // Convert from Steam's audio format to Godot's format
    int RawValue = (int)VoiceBuffer[0] | ((int)VoiceBuffer[1] << 8);
    RawValue = (RawValue + 32768) & 0xffff;
    float Amplitude = (RawValue - 32768) / 32768.0f;

    // Push the converted frame to the audio player
    _playback.PushFrame(new Vector2(Amplitude, Amplitude));

    // Remove the used frames
    VoiceBuffer.RemoveAt(0);
    VoiceBuffer.RemoveAt(0);
}
```

Visual Studio's profiler indicated a bottleneck at the last 2 lines of that for loop, where the used frames are being removed. Upon some research into this, it turns out that removing the items at the start of a huge array causes everything else in the array to get shifted forward to replace the removed slots, which is the performance intensive action due to the size of the array. This performance delay caused a noticeable delay in the rate that audio data was being played, causing the popping voice audio. I solved this issue by, instead of removing the data at the start of the array, reverse the array at the start and play audio frames from the back of the array (which was the old front of the array).

```
CSharp
VoiceBuffer.Reverse();

for (int i = 0; i < Playback.GetFramesAvailable(); i++)
{
    if (VoiceBuffer.Count <= 1) return;

    // Convert from Steam's audio format to Godot's format
    // We're using the audio samples from the end of the array this time
    int RawValue = (int)VoiceBuffer[^1] | ((int)VoiceBuffer[^2] << 8);
    RawValue = (RawValue + 32768) & 0xffff;
    float Amplitude = (RawValue - 32768) / 32768.0f;

    // Push the converted audio data to the audio player
    Playback.PushFrame(new Vector2(Amplitude, Amplitude));

    // Delete the used samples from the end of the array
    VoiceBuffer.RemoveAt(VoiceBuffer.Count - 1);
    VoiceBuffer.RemoveAt(VoiceBuffer.Count - 1);
}
```

Performance

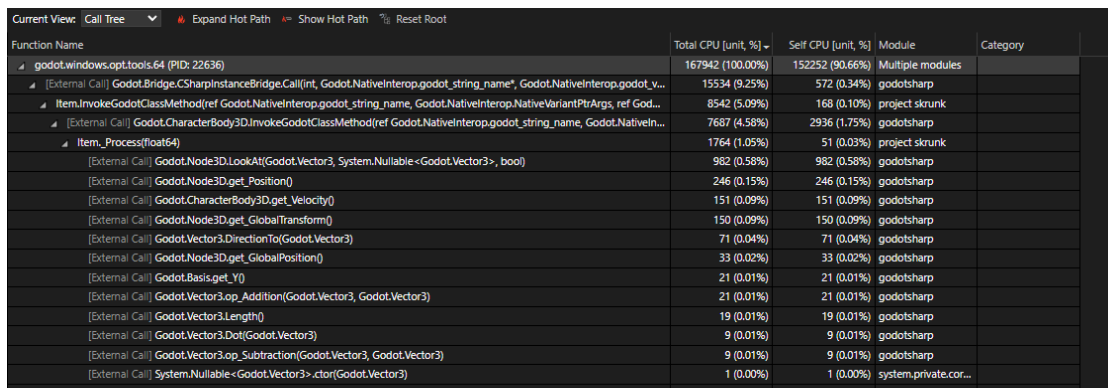
During playtesting, the host's framerate plummeted for every player that connected to the server, causing the whole server to lag as the game was synced less for players. In my testing, I found that roughly 400fps was lost on my computer for every player that connected to the server (it ran usually at a thousand when no-one was connected). I originally thought that this was caused by the amount of data the host has to send to clients – almost everything in the game is calculated by the host and sent to other players. That logic wasn't a huge problem – however, everything in the game was being calculated and sent, despite it not being used (i.e. an enemy in a really far room roaming).

Since some stuff doesn't need to be synced over multiplayer, I decided to disable processing on creatures and items that were distant from any player. These changes caused the 400 fps loss per player to turn into a 100 fps loss per player, which is

significantly better. However, in wide open spaces, creatures can be visibly seen activating and de-activating when a player approaches.

In sprint 3, this system should be improved upon – for an eco-system, the creatures should interact with each other even when the player isn't around. They should be stumbling into a scene that was playing out without them. I want to change the system to a system where everything is calculated 24/7 on the host, but nothing is transferred over the internet unless a player is nearby. I'd need to monitor the performance impacts of this change, however. I can also add a player cap to prevent too many players from joining and tanking the host's framerate – I need to find an ideal player count though.

I also profiled my running game to identify performance bottlenecks that weren't caused by multiplayer syncing – the results of the profiling can be seen below, showing the thing with the worst performance – items.

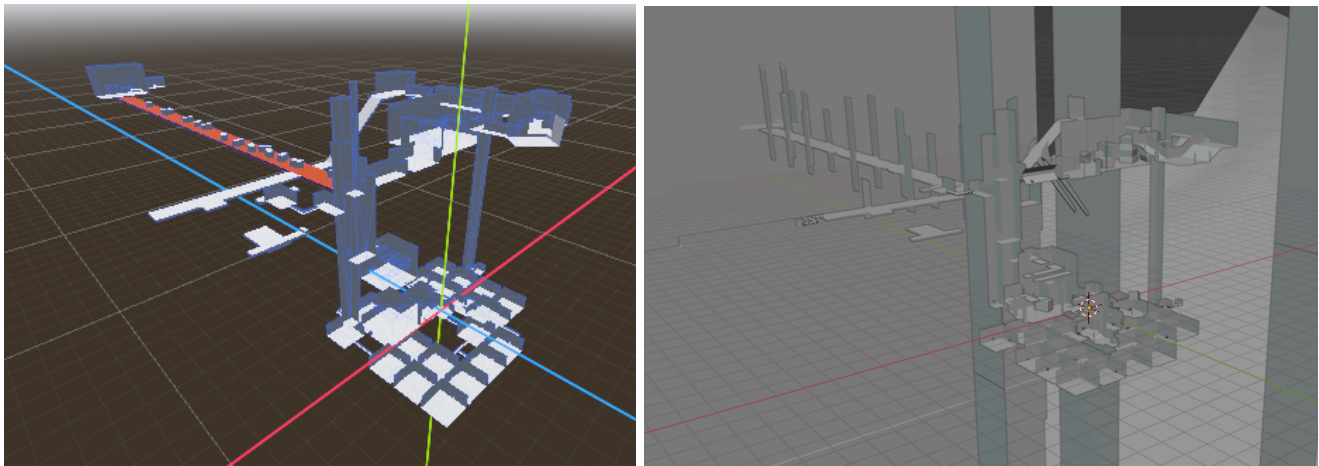


Function Name	Total CPU [unit, %]	Self CPU [unit, %]	Module	Category
godot.windows.opt.tools.64 (PID: 22636)	167942 (100.00%)	152252 (90.66%)	Multiple modules	
[External Call] Godot.Bridge.CSharpInstanceBridge.Call(int, Godot.NativeInterop.godot_string_name*, Godot.NativeInterop.godot_v...	15534 (9.25%)	572 (0.34%)	godotsharp	
Item.InvokeGodotClassMethod(ref Godot.NativeInterop.godot_string_name, Godot.NativeInterop.NativeVariantPtrArgs, ref God...	8542 (5.09%)	168 (0.10%)	project skunk	
[External Call] Godot.CharacterBody3D.InvokeGodotClassMethod(ref Godot.NativeInterop.godot_string_name, Godot.NativeIn...	7687 (4.58%)	2936 (1.75%)	godotsharp	
Item.Process(float64)	1764 (1.05%)	51 (0.03%)	project skunk	
[External Call] Godot.Node3D.LookAt(Godot.Vector3, System.Nullable<Godot.Vector3>, bool)	982 (0.58%)	982 (0.58%)	godotsharp	
[External Call] Godot.Node3D.get_Position()	246 (0.15%)	246 (0.15%)	godotsharp	
[External Call] Godot.CharacterBody3D.get_Velocity()	151 (0.09%)	151 (0.09%)	godotsharp	
[External Call] Godot.Node3D.get_GlobalTransform()	150 (0.09%)	150 (0.09%)	godotsharp	
[External Call] Godot.Vector3.DirectionTo(Godot.Vector3)	71 (0.04%)	71 (0.04%)	godotsharp	
[External Call] Godot.Node3D.get_GlobalPosition()	33 (0.02%)	33 (0.02%)	godotsharp	
[External Call] Godot.Basis.get_Y()	21 (0.01%)	21 (0.01%)	godotsharp	
[External Call] Godot.Vector3.op_Addition(Godot.Vector3, Godot.Vector3)	21 (0.01%)	21 (0.01%)	godotsharp	
[External Call] Godot.Vector3.Length()	19 (0.01%)	19 (0.01%)	godotsharp	
[External Call] Godot.Vector3.Dot(Godot.Vector3)	9 (0.01%)	9 (0.01%)	godotsharp	
[External Call] Godot.Vector3.op_Subtraction(Godot.Vector3, Godot.Vector3)	9 (0.01%)	9 (0.01%)	godotsharp	
[External Call] System.Nullable<Godot.Vector3>.ctor(Godot.Vector3)	1 (0.00%)	1 (0.00%)	system.private.cor...	

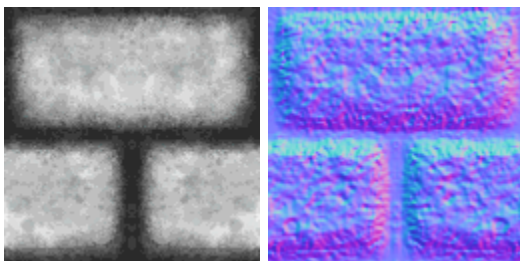
Turns out game items use up a lot of the player's CPU, particularly with the `LookAt()` function (which is an expensive calculation) and some of the item's physics calculations involving `MoveAndSlide()` and collision checking in general. `LookAt()` is used to make the time face the direction it's travelling. To reduce `LookAt()` calls, I set an export variable where, if true, the `LookAt()` calls would be made. This way, I can use `LookAt()` only for items I want, reducing the total number of calls to the function. I still achieve the desired result, with less overall calls to `LookAt()`.

Game World Design & General Art Direction

I worked on decorating my existing game level and creating an art style for my game simultaneously. From my existing level's blockout using CSG nodes, I exported it from Godot and into Blender, where I did all of the decorating and texturing for it. Blender allowed me to decorate easier than I would have in Godot, as it's a proper 3D modelling program. However, in Blender, I couldn't add game elements like climbables or enemy spawners – those had to go back into Godot and manually animate. I used Blender to finalise the geometry of the game level – adding new areas and adding decorative geometry to the game world. Below are images of the original imported geometry (left) and the final decorated and untextured geometry (right).



With my textures and materials, I had a few options on how to handle this – procedurally generate the textures in Blender, procedurally generate them in Godot, or manually draw the textures and import them that way. I eventually decided to manually draw the textures using Krita. Blender procedural generation requires me to bake the textures to a separate image file that Godot can read from, which is a pain for me. Godot procedural generation is slightly better, but would overall take longer to create textures due to the code-based nature of Godot’s shaders. Alternatively, I could use Godot’s Visual Shaders for the textures, but that forms the same problem as it’s largely derived from the code-based shaders. As such, I decided to manually draw simple textures in Krita and import those into Godot. Each texture is drawn in black and white and then the colour is shifted in-game to what I want, meaning I can re-use the same texture in different colours for different areas. Normal maps are generated for these textures via the automatic normal map generation present in Krita – which holds surface information about the texture, making it look bumpy. Both the black and white images and normal maps are exported and used in Godot. Examples of textures used in-game are below, with the greyscale image on the left, and the normal map on the right.



Because I wanted the textures to be simple and low in file size, the images themselves are 128x128. This is where I developed the subtle pixel-art aesthetic. As such, I decided to add a subtle shader effect to the player’s camera. The game is pixelated, and then colours are quantised to both achieve a retro-aesthetic. I intentionally wanted the effect to be subtle. The shader code used is below.


```

C/C++
shader_type canvas_item;
render_mode unshaded;

uniform sampler2D SCREEN_TEXTURE: hint_screen_texture, repeat_disable, filter_nearest;
const int pixel_size = 2;
const float quantisation_colours = 30.0f;

void fragment()
{
    // Pixelisation
    float x = float(int(FRAGCOORD.x) % pixel_size);
    float y = float(int(FRAGCOORD.y) % pixel_size);

    x = FRAGCOORD.x + floor(float(pixel_size) / 2.0) - x;
    y = FRAGCOORD.y + floor(float(pixel_size) / 2.0) - y;

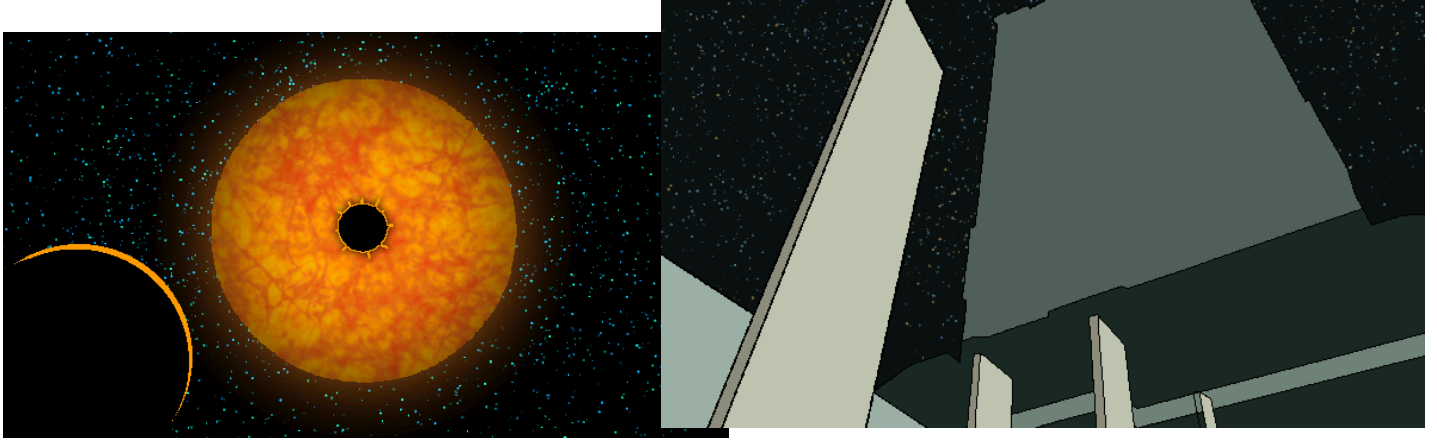
    vec4 pixelisation = texture(SCREEN_TEXTURE, vec2(x, y) / vec2(1.0 /
SCREEN_PIXEL_SIZE.x, 1.0 / SCREEN_PIXEL_SIZE.y));

    // Quantisation
    float r = round(pixelisation.r * quantisation_colours) / quantisation_colours;
    float g = round(pixelisation.g * quantisation_colours) / quantisation_colours;
    float b = round(pixelisation.b * quantisation_colours) / quantisation_colours;
    vec4 quantisation = vec4(r, g, b, pixelisation.a);

    COLOR = quantisation;
}

```

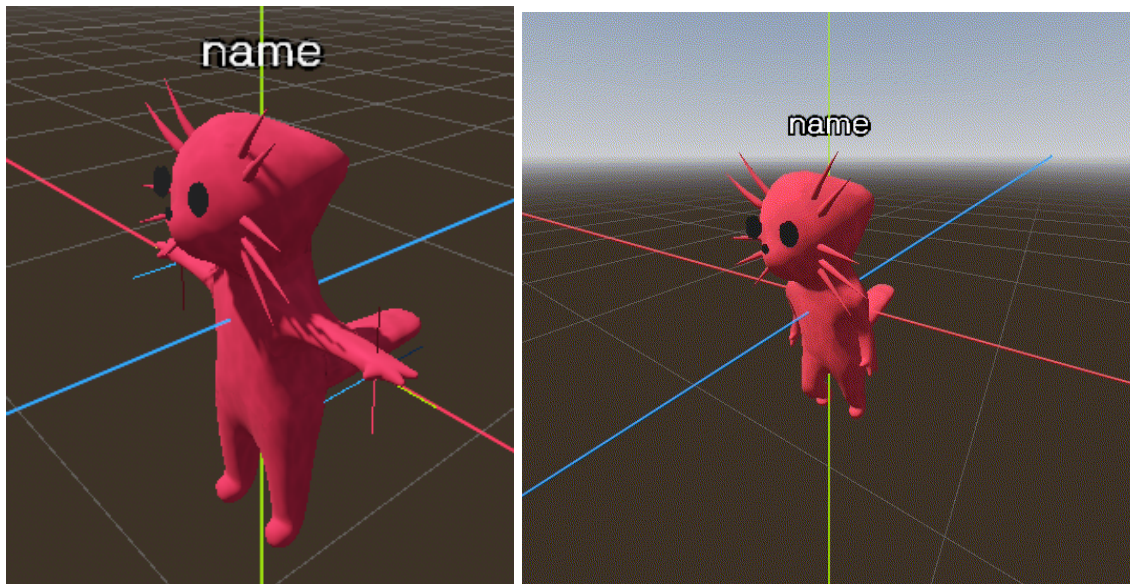
I also drew some backgrounds for the game's UI. These largely served as concept art that I drew while trying to come up with a game art style. Some backgrounds are included below.



Player Design & Animations

Working with my concept player 3D model (from my game design doc), I rigged it, animated it, and finally implemented it into the game. The animations for the player are inspired by how Rain World's slugcat moves (can be seen below)

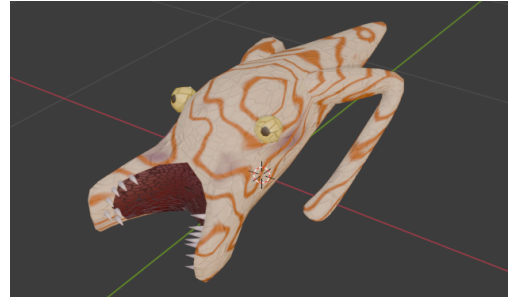
Final model of the player character (left), with example running animation (right)



Creature Designs

My current creatures are literally just coloured spheres – for the final game, that has to change. I decided that the creatures should be based on a combination of real-world animals, as it fits the ecosystem theme I'm going for.

For the current unnamed enemy, I was largely inspired by alligators (for most of its body) and frogs (for its legs). I decided to go for an alligator due to their violent nature, and went with the frog legs due to its leaping behavior. I decided to go for something unique with the vertical jaws that move horizontally so I'm not completely ripping off real world nature. Its final design is below. With this final design, I decided to rename this creature to the Leaper.



For the prey entity, I decided to make them small birds – in this case, crows (but they ended up looking more like bats). The final design is below.



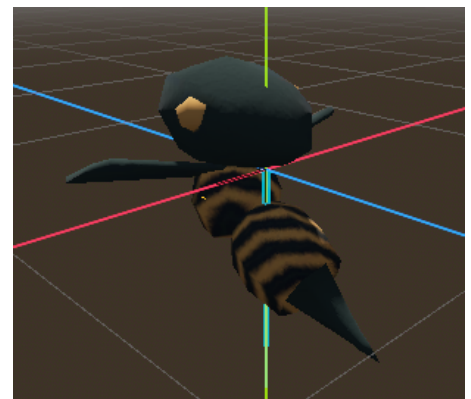
New Creatures

I'm adding 1 new creature this sprint – the Flimling (name is subject to change). In terms of development, the prior creatures were a mess and were a nightmare to work on – so I decided to use a more structured approach. I used a state machine and thoroughly planned out how I wanted the creature to work in code comments.

It has two states – roaming and aggressive (where it starts roaming). In roaming, it roams randomly around its surroundings but, if it gets too far away from its starting 'den', it'll go back towards it. At its starting den, it spawns an egg – which is edible for players, immediately giving them max food. Upon eating this egg (or by attacking the flying creature), it adds the nearby creatures to a "marked for death" list, and enters the aggressive phase.

When in the aggressive phase, it loops through all of the creatures that are marked for death and kills them. Starting with the first one, it navigates to that entity (always knowing where it is). When it gets within a 5m radius, it lunges at the creature, dealing damage on contact. It then continues lunging at the creature until it is dead, where it'll then go to the next creature in the marked for death list and repeat. When the marked for death list is empty, it goes back to the roaming phase.

Its visual design is largely based on hornets or wasps (or some other similar creature). The final design for it is to the right.



New Levels

I decided to move new level development into sprint 3 due to time constraints within sprint 2. As such, this will not be covered here.

Feedback

From the Techquity game jam I've been uploading my in-development game to, I have received no feedback. A reason for this is likely because school firewalls *really* don't like my game – it gets blocked by the school's firewall, likely due to the Steam API implementation. As such, participants in the inter-school game jam couldn't playtest my game at school, and nobody played the game at home on their own. My only form of feedback is from the scheduled playtests I run with friends, which I continued doing for this sprint.

My playtesters liked the artstyle, but didn't really comment on anything. There weren't many gameplay changes – it's virtually identical gameplay-wise to the last sprint – so I can't really take on feedback changes. I can't really make changes from the feedback, so I largely used this as an opportunity to test and debug multiplayer and game logic (of which there were many errors).

Sprint 2 Reflection

I'd say this sprint went overall very well. However, there was an issue with time constraints (which should have been expected with a multiplayer game). Mid-way through the sprint, my family and I ended up going on a long holiday, which impacted how much I could work on the game. Additionally, I wasn't prioritizing key features and bugs and ended up spending a good chunk of time creating features that weren't on my design documents. Due to this, I had to shift new level designs to the next sprint. I need to be better with time management for sprint 3 and for the rest of development.

Overall, not too many bugs were found throughout playtesting (the most notable of which were some significant performance issues). The playtest's gameplay has also been virtually identical to sprint 1's due to the lack of gameplay changes. The playtesting videos and debug logs for this sprint can be found in [📺 Sprint 2 Playtesting](#) .

Sprint 3

This sprint will be fixing any issues I find in the last sprint, implementing last minute ideas, creating more general content, adding features pushed back from previous sprints, and final polish.

This sprint will:

- Implement feedback and bug fixes from playtesting
- Add new levels and environments
- Add new creatures/content in general
- General finalisation, audio implementation, polish, and other last-minute changes

In the end, I should have produced a finished game that meets my initial specifications. This sprint, and the end of development, should end by October/November 2025.

Development

Feedback and Bug Fixing

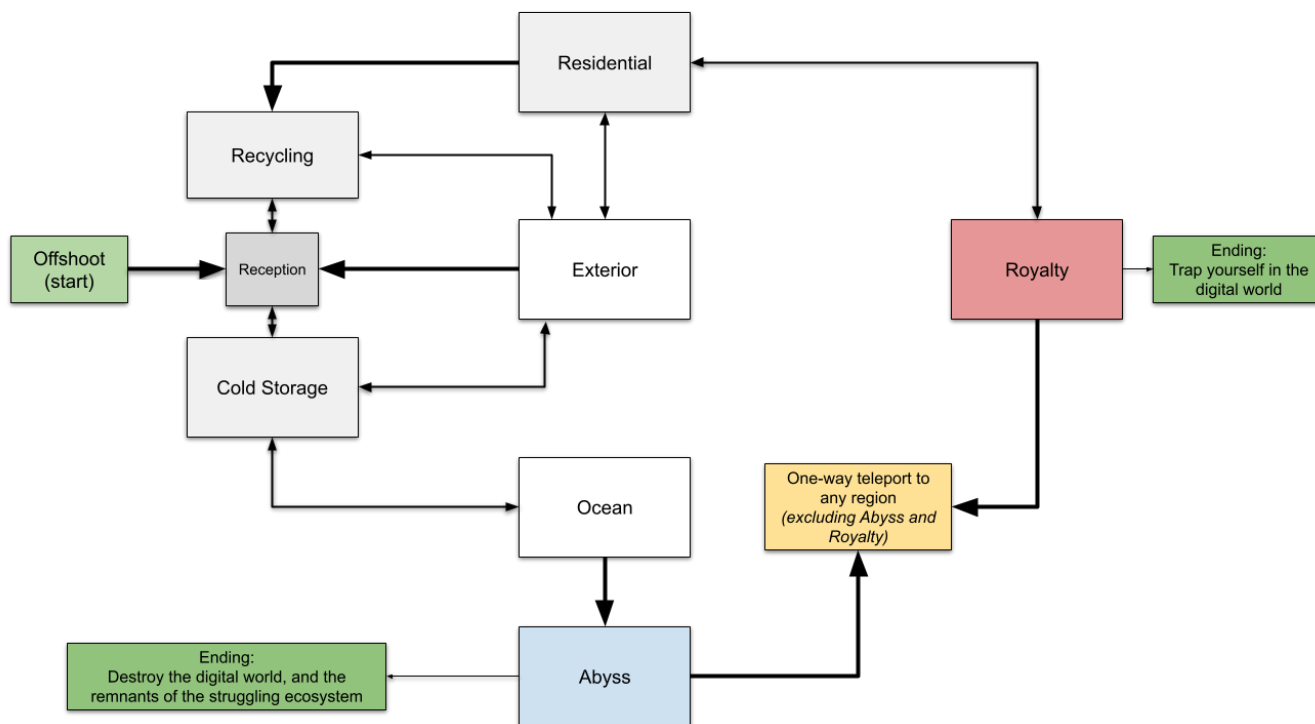
There weren't too many significant issues evident from playtesting. The biggest problem I saw were performance problems – the game was using up way too much of the CPU to run effectively. Again, I originally thought this was because of some very unoptimised code in my game, but when I checked Godot's profiler, I discovered that there were ~10,000 nodes and ~20,000 objects existing in the main game world. I originally believed that this number of nodes would only affect memory usage – but it actually does affect CPU usage. Godot's **SceneTree** (at the root of every running Godot project) handles Godot's node system and has to 'process' each existing node (unless the node has processing manually disabled). Hence, each node in the scene causes a slight performance hit and. With my insanely large number of nodes, it was causing a pretty significant performance hit – but why were there so many nodes in my scene?

I eventually traced the source of it to my game's climbable objects. For context, my game's climbable objects are paths defined by a **Path3D** node. The player can grab onto the pre-determined path, move along it, and leap off again – which is how the player's climbing works. However, Path3D nodes are completely invisible – it needed a mesh. Originally, in the prototype, I was creating a mesh for it manually – which was tedious and time consuming. I eventually got so annoyed with doing it manually that I made a script that places **MeshInstance3D** nodes along the path (by default, every 0.2m), automating the process. This, of course, generated thousands of nodes because climbable objects are almost everywhere in the game world. To fix the performance problems, I had to rework this system to use significantly less nodes. I eventually reworked the system so that it uses 1 mesh for each edge in the path, cutting back on the nodes created and fixing the problem.

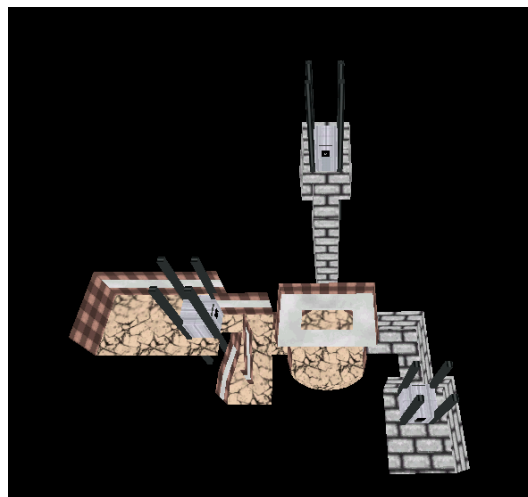
New Levels and Game Environments

As one of my goals this sprint is to add more general content, I added more regions for the player to explore. I also gave all of the regions names – with the original starting region developed in the prototype being named 'Offshoot.' Before creating levels aimlessly, I created a quick diagram to determine how levels should flow into each other, and how

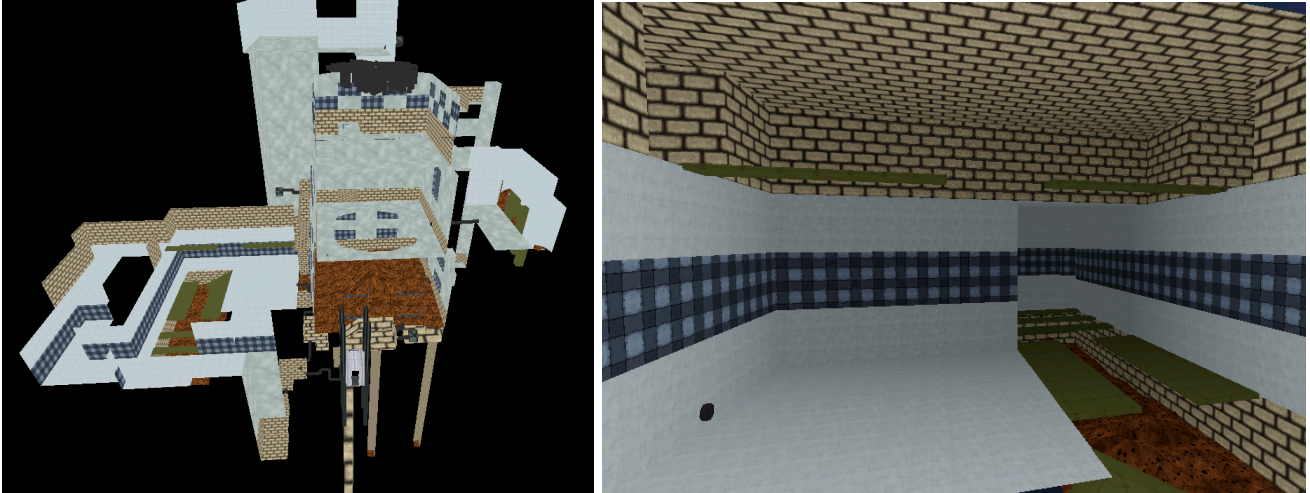
they lead to the endings for the story created in the game design document. The arrows show how the player can transition between regions, with the bold arrows indicating that the player can only go one-way.



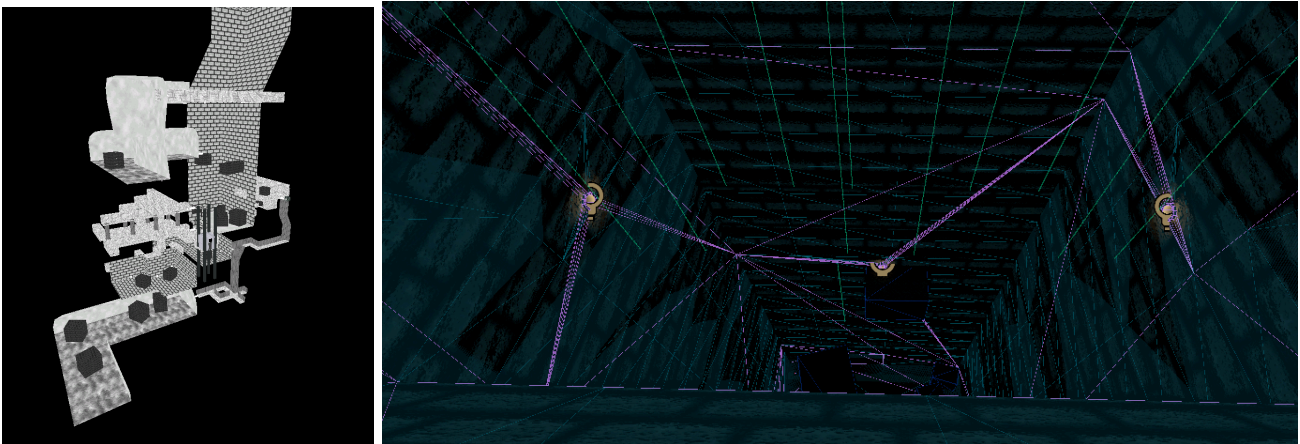
Just off Offshoot (the player's starting location, which is the prototype level developed in previous sprints), I added a small branching region – dubbed 'reception.' This region currently exists as a small stepping stone to go to their choice of Recycling or Cold Storage.



With Recycling, I mainly focused on horizontal platforming. There are several platforming encounters that players have to get through. There are also many pitfalls for the player, forcing them to watch their step or face death. In some areas, I'm concerned about the difficulty of some platforming. I'll need to see how playtesters find it before making changes, as I don't want to make the game braindead easy.



With Cold Storage, I focused on climbing when it comes to its platforming challenges. I deliberately made this region darker. Again, some of these may be too difficult for players, but I'll only know from playtesters.



Unfortunately, I ran out of time to develop the rest of the regions, which also means that the game cannot be completed officially.

New Creatures

In terms of new creatures, I added the kidnapper. I wanted something to be static in the game world, instead of constantly roaming around – which allows the player to avoid it with their mobility. As such, it will only attack if you touch one of its tendrils.

Other than the kidnapper, I never ended up adding new creatures – mainly because of a lack of ideas. I'll try to look over previous concepts and inspiration, or maybe try to find new inspiration sources. Alternatively When playtesting comes, I'll ask for ideas and see what they think.



New Content

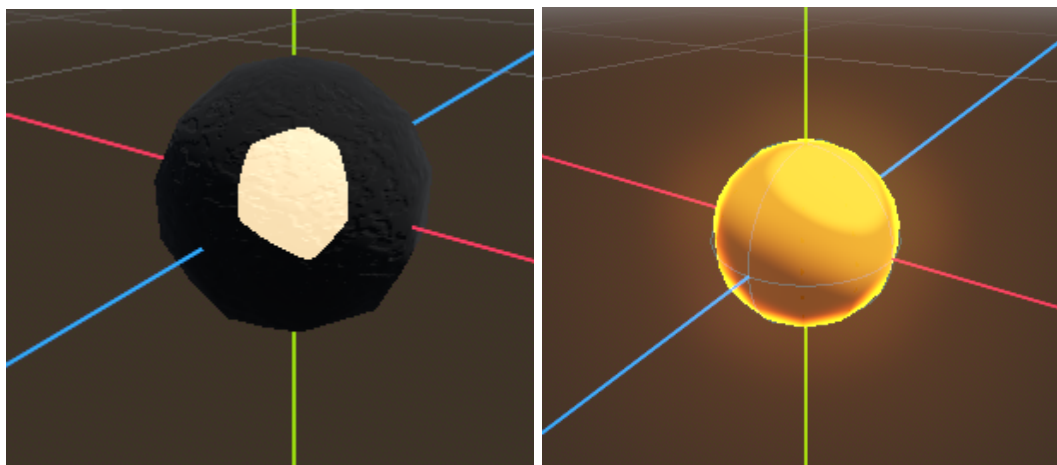
Items

For general raw content, I added a bunch of new items that either gave players new mobility options or a method to outsmart enemies.

Firstly, I added an item called 'pheromones.' Pheromones, when thrown, attract nearby creatures to it. However, players will have to be careful, as they are also drawn to the thrown pheromones.

Additionally, I added a relatively minor item called orbs. These are meant to act as light sources in darker levels – particularly Cold Storage. Other than that, there's not much of a use for them. With these, I want players to consider taking essentially dead weight with them, or bring more useful items with them but struggle to see. Both have pros and cons, and would influence teamwork between players as one player could hold the light while another is on the offense.

Pheromones (left), and Orbs (right)



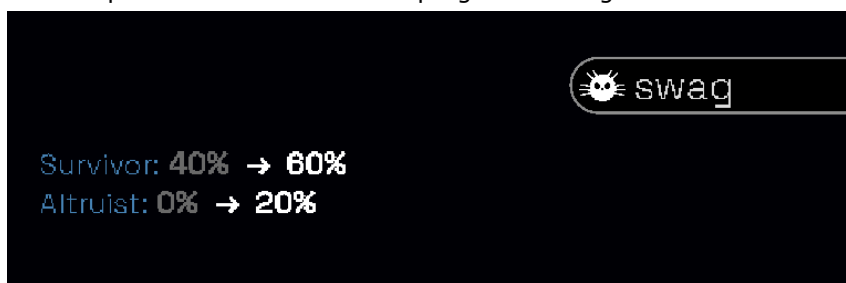
Evolutions

From playtesting, I noticed that, given the game isn't very linear, players weren't feeling like they were progressing. Alongside the main objective of surviving, I added a feature called evolutions. Evolutions are unlockable upgrades for completing a specific challenge. These are mainly based off of Rain World's passages where, on completing a challenge, the player is granted a one-time-use token to immediately teleport to any discovered shelter. In my game, evolutions give the player some kind of boost in a particular attribute, and are usually unlocked by excelling in that field. Currently, the evolutions I have implemented are as follows: Survivor, Altruist, Brutalist, and Pilgrim.

Survivor mainly exists to teach new players how evolutions work – all other evolutions are locked before Survivor is unlocked. When all players in a lobby survive a night (i.e. nobody dies), progress towards unlocking Survivor is increased by 20% – when all players survive for 5 times, it unlocks. Survivor only gives a minor health bonus, but with no downsides – it is a straight upgrade as opposed to using no evolution.

After Survivor is unlocked, Altruist, Brutalist, and Pilgrim can now be progressed. Altruist is attained by refusing to harm any creature – behaving like a pacifist. When unlocked, it gives the player heightened movement abilities (particularly faster walk speed and larger jump speed), but at the downside of significantly worse throwing abilities – meaning that they can only effectively harm something that's literally meters away from them. Brutalist is the exact opposite of this – progress is attained by seeking out and harming creatures. When unlocked, it gives better fighting abilities – with a faster throw velocity and larger throw range – but with worse movement abilities. Care was taken with Brutalist to ensure that key platforming was still possible with the decrease in movement potential – hence jump height and leap velocity was not affected, only walk speed was affected.

Visual representation of evolution progression in-game



Polish

HUD Improvements

One thing that I also noticed was that the game's HUD no longer matched the artstyle I had for the game – it still looked like something from a prototype. As such, I decided to change it.

Before and after



I also added crosshairs to the game that reflected what the player is holding. There isn't a lot to go by in terms of figuring out what an item does. With the crosshairs, I added a crosshair to indicate that an item can be grabbed, and two crosshairs to indicate that the held item is a weapon or food respectively.

'Item is grabbable' crosshair (left), and 'held item is edible' and 'held item is a weapon' respectively on right.



Multiplayer Smoothing

One thing I noticed from playtesting was that player movement between players was jittery, mainly because of how multiplayer works. Let's use a thrown item's position as an example – when new packets from other players are received about its position, the data is immediately applied to it, updating its position. However, this is not smoothly applied, meaning that the item's position constantly snaps to its new position and doesn't look great. This was especially noticeable for non-host players – since the host calculates all of the enemy behavior and game logic and syncs those for all other players, all game items appeared to jitter erratically.

With Godot's high level multiplayer, it handles all of the internal multiplayer logic internally. This makes things simple for newbies, but makes it harder for advanced use-cases. `MultiplayerSynchronizer` nodes don't have an option to smooth new positions, and it doesn't allow me to access incoming data to do it myself. If I wanted to smooth incoming data, I had two options:

1. Replace every `MultiplayerSynchronizer` node with a custom solution using `RPC` functions to make it myself – which is tedious and time consuming.
2. Modify the engine to expose internal `MultiplayerSynchronizer` data – which requires modifying internal engine code and rebuilding the engine, but is significantly easier for my current implementation. Godot is an open source engine, meaning its codebase is publicly accessible and I can modify the code to my heart's content.

I eventually decided to implement option 2 to avoid re-writing a lot of my code. However, this involves understanding some of the engine code – which can be confusing.

After some analysis, I found that `MultiplayerSynchronizer` nodes interface with the internal `MultiplayerAPI`, which handles sending and receiving data. It reads data that is received from other players, and automatically applies the property value to that object – which is all handled by this function:

```
C/C++
// Godot Engine Repository
// https://github.com/godotengine/
// Line 173; multiplayer_synchronizer.cpp

Error MultiplayerSynchronizer::set_state(const List<NodePath> &p_properties, Object
*p_obj, const Vector<Variant> &p_state) {
    ERR_FAIL_NULL_V(p_obj, ERR_INVALID_PARAMETER);
    int i = 0;
    for (const NodePath &prop : p_properties) {
        Object *obj = _get_prop_target(p_obj, prop);
        ERR_FAIL_NULL_V(obj, FAILED);
        obj->set_indexed(prop.get_subnames(), p_state[i]);
        i += 1;
    }
    return OK;
}
```

We want to expose the data that passes through this function. The easiest way I found of doing this is creating a new signal as a part of Godot's base `Object` class, which I called `multiplayer_synchronizer_data_updated`. When new data passes through this function, we call this signal for use in the engine. However, if we're using this signal, we want to set incoming parameters ourselves in order to smooth it out – so we block the `set_indexed` call in order to run it ourselves in code. Otherwise, let the old synchronizer handle it. The final version of this function is below.

```
C/C++

Error MultiplayerSynchronizer::set_state(const List<NodePath> &p_properties, Object
*p_obj, const Vector<Variant> &p_state) {
    ERR_FAIL_NULL_V(p_obj, ERR_INVALID_PARAMETER);
    int i = 0;
    for (const NodePath &prop : p_properties) {
        Object *obj = _get_prop_target(p_obj, prop);
        ERR_FAIL_NULL_V(obj, FAILED);
    }
}
```



```

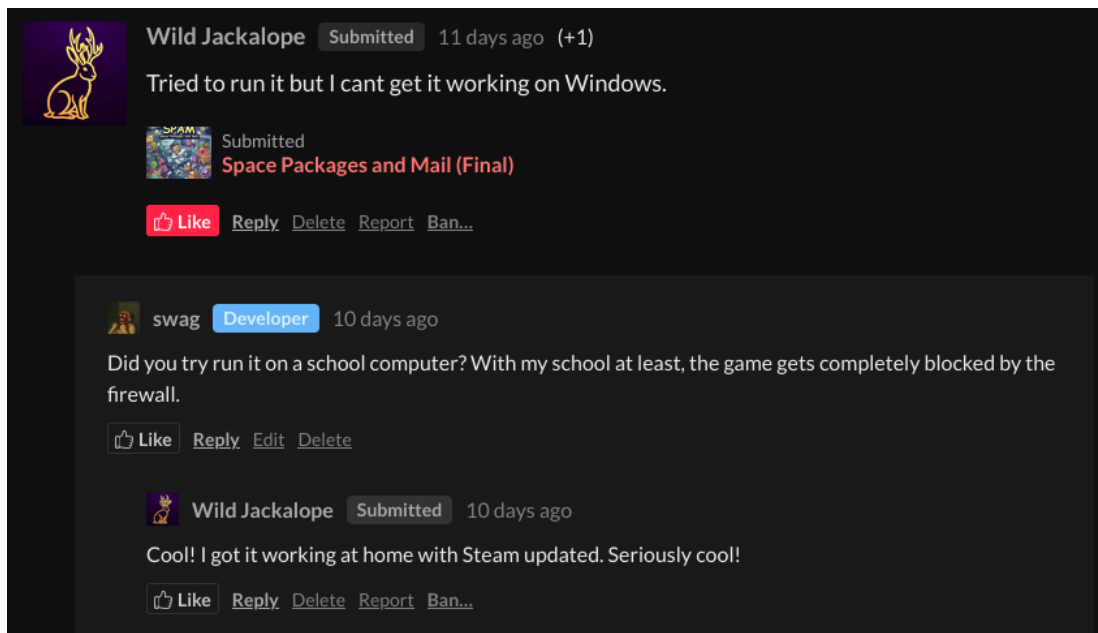
if (!obj->has_connections(
    SceneStringName(multiplayer_synchronizer_data_updated))) {
    // Update automatically if the signal hasn't been connected to.
    obj->set_indexed(prop.get_subnames(), p_state[i]);
}

obj->emit_signal(SceneStringName(multiplayer_synchronizer_data_updated), prop,
p_state[i]);
i += 1;
}
return OK;
}

```

Feedback

With sprint 3, I've run a playtesting session. I was hoping I wouldn't need to do this because of the Techquity Game Jam, but not enough people have played it and given feedback. Additionally, Windows' antivirus does not trust my game – likely because of Godot Engine changing their app certificate³¹ in their most recent update. This means that the game gets completely blocked on school computers, which is where people participating in the jam are likely playing the game.



Sprint 3 Playtesting

³¹ <https://github.com/godotengine/godot/issues/110612>

In terms of feedback from playtesters, I've heard complaints that the audio is shoddy (because of my lack of sound design skills and knowledge), and that the game needs a map in order to keep track of explored areas and player locations.

The game's audio was frankly rushed and was a bit of an after-thought. In some areas, it completely ruined the experience. I tried to learn how to use sound design software and learn how to create audio – which, in places, didn't turn out well. I'll likely swap it out with something else – probably something downloaded from a professional audio website (so it doesn't sound awful). The requested map, however, is a full feature that may take some time.

There were also many bugs that occurred. Some of which include game restarting logic breaking, broken enemy pathfinding, and issues with some of the performance enhancers I created. Some issues and frustrations with gameplay also occurred – like unnecessarily difficult parkour involving crawlways, and players ignoring tutorial instructions. I'm going to address these problems with a hotfix which hopefully fixes them – which will be the final version of the game.

In terms of general gameplay, playtesters said roughly the same things as previous sprints, mainly because there weren't many gameplay changes from the previous sprint – excluding the new regions and items. Because of the current incomplete nature of this sprint, I haven't implemented enough of the features I want to get full feedback on the final game's version.

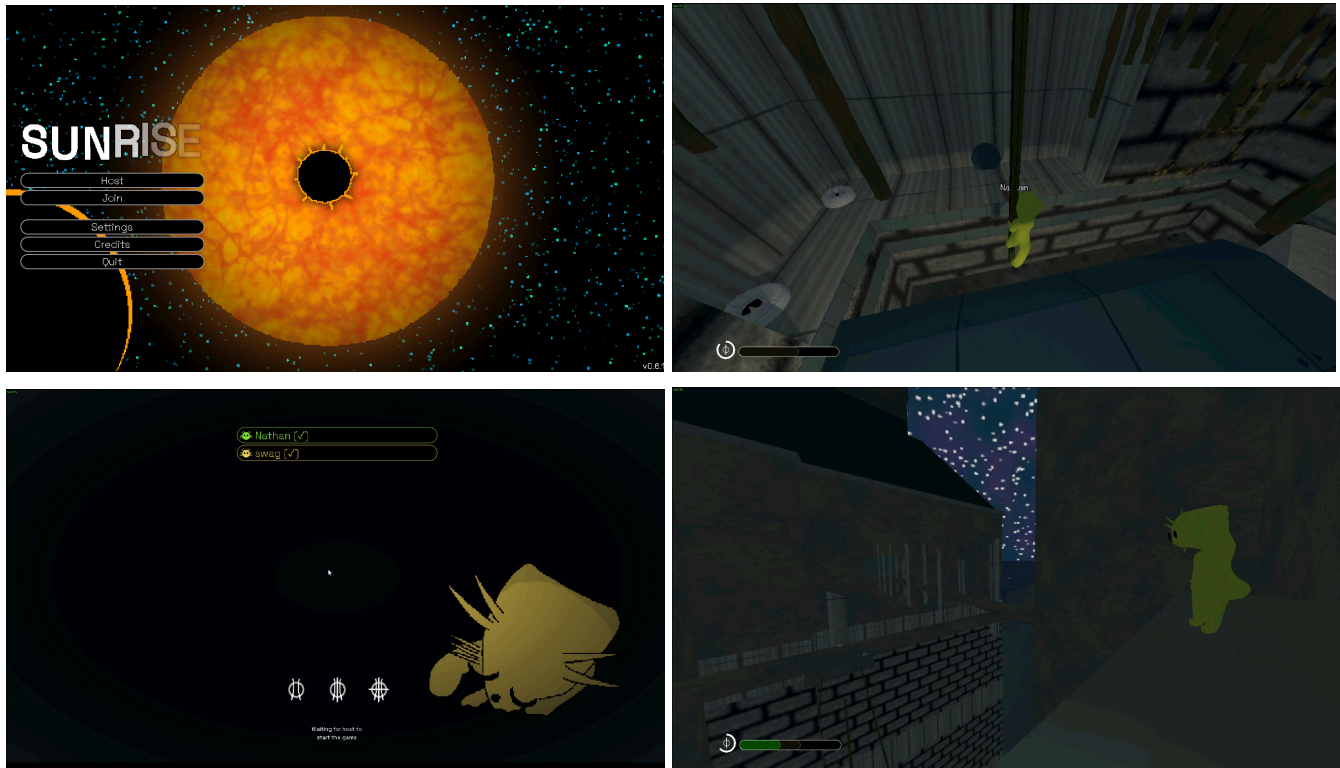
Sprint 3 Reflection

This sprint was unfortunately forced to end due to time constraints. In the game design document, I highlighted that everything should be completed by October, which had arrived at this stage in development. I feel like that this was largely to do with the delays that arose out of sprint 2, as that chewed up so much of my original development time. By the time this sprint came around, I had learned my lesson – but it was too late.

I don't know if I've done enough to call this 'done' – which is a problem, given that this is ideally the final sprint. The time constraints in previous sprints was likely the main reason that I never completely finished everything. Frankly, the game is far from done.

Final Outcome

As of right now, the game is released on itch.io at <https://swag0.itch.io/sunrise>. The game is not finished enough for me to warrant purchasing a Steam store page. Several screenshots of the game in its current state are below.



Original Requirements

In accordance of my original game design, I decided that, in order for my game to be considered complete;

- My game must be a survival/exploration game.
- My game must be cooperative game
- My game must be difficult
- My game must be engaging and fun for my target audience
- My game must be easy to learn
- My game must have intuitive controls and accessibility options
- My game will be released on Steam

From these, I broke each one of these down into attainable measures. In terms of these specifications, my final game fulfilled the following;

- ☒ My game must be a survival/exploration game.
 - ☒ The player must eat food to survive

- ☒ The player must explore to find food
- ☒ The player must avoid danger and survive
- ☒ My game must be cooperative game
 - ☒ The player must cooperate with other players to survive
 - ☒ The players can put themselves at risk to save other players
 - ☒ The game will be made more difficult failing to cooperate (It won't make the game impossible, just minor punishments. Shenanigans are fine)
- ☒ My game must be difficult
 - ☒ The player will be weak and fragile
 - ☒ The player can be easily killed if they are not careful
 - ☒ Cooperation is encouraged because it makes survival easier
- ☒ My game must be engaging and fun for my target audience
 - ☒ I will appeal to my desired target audience
 - ☒ I will playtest often to ensure that my game is fun – if not, make design changes to correct
 - ☒ I will ensure that my game won't be too difficult and frustrate players, causing them to quit
- ☒ My game must be easy to learn
 - ☒ My game will have simple and logical mechanics that are easy to understand
 - ☒ My game's loop will be simple to understand
 - ☒ The player shouldn't require prior context to pick up and play at any point

The following requirements were not fulfilled.

- ☐ My game must have intuitive controls and accessibility options
 - ☒ My game will have controls that make sense
 - ☐ *My game will have several accessibility features such as input rebindings and colour-blind options*
 - ☐ *My game will have several translations to other languages, making the game accessible to non-English speakers*
- ☐ My game will be released on Steam
 - ☐ *The game will be released on Steam*
 - ☒ My game will use Steamworks for multiplayer functionality (for connecting to friends, etc.)
 - ☒ My game will use the player's Steam account for player-specific variables such as player name

In the end, I was so pressed for time that I never had the chance to fully implement many accessibility features – let alone finish the game. This isn't critical for gameplay, but is critical for user experience. Additionally, the game was not released to Steam because I don't consider the project complete enough to release it commercially in its current state.

Relevant Implications

At the beginning of the project, I identified a few relevant implications that are essential for making my project succeed. I'll go over each of them to ensure that they are all addressed correctly like I said I would at the start.

Aesthetics

When developing, I also took great care to ensure that the game looks and feels just the way I want it to from the game design document. The game looks the way it does in the original sketches, and plays the way I specified. I had kept my game idea solidified from the start.

Functionality/Usability

A lot of time was spent ensuring that the game was functional – which is actually the main reason the game is incomplete. A significant chunk of development time was spent debugging and resolving issues, which means that what I have developed is a functional game, but still with not enough content to keep players playing.

Legal/Ethical/Sustainability

Initially, I knew that there would be cyber-security concerns in players. In practice, I mainly tried to keep players in a trusted environment with people that they know, away from bad actors. With Steam's networking, I ensured that players can only connect to a server if they are Steam friends with the host (or at least one player who is already in the lobby). Otherwise, Steam's validation will refuse the connection, keeping the players in the lobby safer. I realise that this means that players cannot play over public lobbies to meet new people (a feature in other multiplayer games that helps solo players find teammates). Public lobbies may be a system that I add later in development.

I also utilised LAN as an alternative method of communication. Steam (owned by Valve) is an external service that I shouldn't have total reliance on. In the event that they are not trustworthy, go out of business, or something else, I used LAN as an alternative method for players to play without using Steam – so I'm not completely reliant on Steam. This method I suspect is less secure, as anyone can connect to the lobby, but only from their local network, meaning they can only be attacked from users on their wi-fi. Players playing on public wi-fi may be at risk but, otherwise, home wi-fi should remain safe.

Overall Reflection

I think that this game idea is good – playtesters all had fun and said positive things – it's just that I could not complete it in the deadline I had set myself. The main reasons for this are mainly due to general game design complexity and development downtime mid-sprint. Multiplayer games have very complex systems – most of which take time to develop, and even longer for it to work flawlessly. For example, sprint 1 (the working prototype) took roughly 6 months to complete (from December 2024 to May 2025) – 6 months for a basic prototype! In the end, almost 800 git commits were created by the end of sprint 3! Additionally, there was some downtime mid-sprint. Like I mentioned in sprint 2's reflection, my family and I went on a long holiday for about a month – which ate up a lot of development time. These two are the main reason the game is not completed by the

final deadline I had set myself. I won't release this project commercially in its current state, due to its unfinished nature.

Learning Outcomes

From this ambitious project, I learned a lot about the importance of playtesting and how multiplayer games work. The initial research at the start helped significantly in understanding how these systems work and how to effectively use them – as opposed to blindly following tutorials.

I need to learn how to better handle complexity in development, and how to keep moving and to focus on key features. All of these are very apparent throughout development – particularly through sprint 2 and 3.

Future

I want to keep developing on this game. With time, I can get to a final outcome that I'm happy with and meets my specifications – I just wasn't able to complete it this year. The main things I'd prioritise for future development would be finishing what I couldn't for sprint 3, and adding the usability and accessibility features I originally had planned. I still want to release this onto Steam at some point, but that could be in the very far future. I'll continue development into the summer and hopefully get a finished product by early 2026, although that might be wishful thinking.