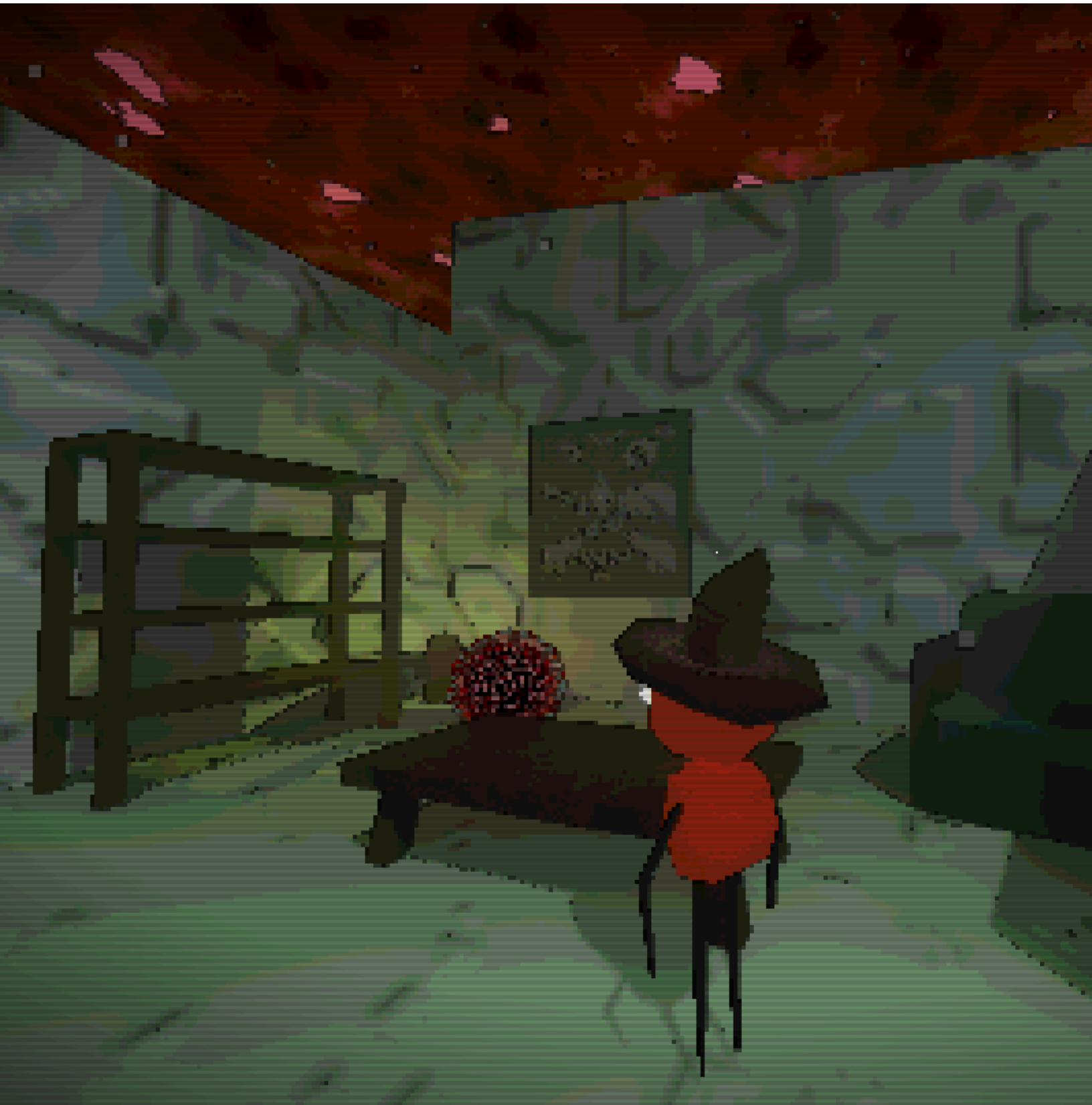


Amelioration

A physics-based, action-adventure game
Scholarship Technology



Introduction:

Hello! Over this year, I've created a game called '*Amelioration*;' an action-adventure, physics-based video game that encourages you to get creative with what remains of the world. The player must find the egg in each level and return it to A-OS, your omnipotent overlord, for incineration. You're destroying these eggs 'to make the world a better place,' or so you are told (which is why the game is titled *Amelioration*. '*Amelioration*' literally means 'the act of making something better'). You can download and play it [here](#) on the itch.io webpage. This document will outline the research behind my ideas and brainstorming, planning, and design process.

Discussion

I've played games for a long time and have always been interested in what makes a good game. In the past, I've often enjoyed action-adventure games like [Red Dead Redemption 2](#), [The Upturned](#), [Half-Life 2](#), and others. I want to know how games are made and how to make them appealing. Although, if I do decide to try game development, I want it to be fun for me to develop. I brainstormed various ideas for games and thought a physics-based video game would be fun for me to develop. I could heavily lean into physics, and make all of my mechanics and enemies revolve around it. That would give me a sort of creative freedom that should hopefully make my development journey enjoyable. This idea became the backbone of my game.

I set myself the initial goal of making a game that will eventually be released as a full indie-type game. This goal may be a bit steep for someone like myself but I've got to get started somehow. I stated earlier that I've enjoyed action-adventure games, so **how can I make an engaging action-adventure game?** To answer this question, let's figure out what goes into making a game.

Analysis:

Before worrying about *how* to make a game, I'll need to figure out what makes games fun in the first place.

I can decide *how* to create my game when that's figured out. There is heavy debate on what game engine is the best for indie game developers like myself.

Next, I need to decide how the player interacts with my game. I must develop a camera and movement scheme for my desired outcome.

Furthermore, I need to figure out how long games typically take to create. I'll need this sort of info because I started this project officially on the 8th of March, and had

until roughly October to finish. I need to ensure I do not over-plan, or fall victim to scope creep.

Finally, when I release my game, what platform should I release it on? Platform means the device on which the game is played (for example; PCs, or consoles like XBOX or PlayStation)

Fun

Before beginning proper ideation and brainstorming, I must figure out what makes games fun.

According to [this article](#)¹ from Plymouth State University, fun is tied to a mental state of 'flow.' The player concentrates on only one thing in this 'flow state'. When this occurs, the player becomes highly productive, and pleasures the player with endorphins released in their brain. The article states that, for a flow state to exist, the player must;

- Be performing a challenging activity that requires skill
- Have a clear goal, and receive immediate feedback
- Have an outcome that is uncertain, but can be vaguely controlled by the player's actions

So, I must create and maintain a flow state within the player. I must keep them challenged, without making them too stressed or too underwhelmed. Due to this, as the player's skill level increases throughout the game, I'll increase the difficulty to keep them engaged.

Additionally, I can use the [MDA \(Mechanics, Dynamics, and Aesthetics\) framework](#)², an academic method to achieve a flow state. MDA is an approach to game design that makes it easier to analyse games. This approach breaks down the components of a game into the 'rules' (mechanics), 'system' (dynamics), and "fun" (aesthetics). MDA will allow me to consider the player's perspective easily while planning. This can be seen in the diagram below;

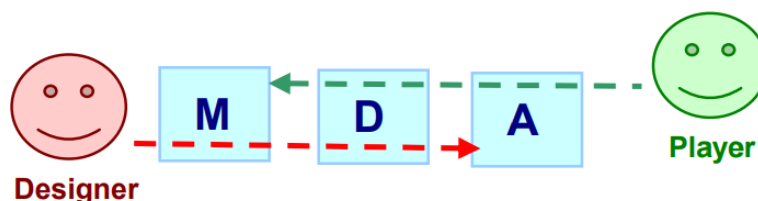


Diagram of how the MDA system works. The designer typically starts with the mechanics, whereas the player experiences the aesthetics (fun) before becoming consciously aware of anything else. Image from Plymouth State University

¹ <https://creatinggames.press.plymouth.edu/chapter/what-makes-a-game-fun/>

² <https://users.cs.northwestern.edu/~hunicke/MDA.pdf>

The player generally experiences the game's aesthetics before becoming consciously aware of the rules and system of that game. Designers work in the opposite direction, working on the mechanics first. By accounting for the player's perspective, I can ensure that my mechanics and dynamics create the desired aesthetics. MDA allows me to have a simple framework that helps me instill a flow state within the player.

Game Engines

Firstly, most games use a game engine, so which one should I use? Because I want to make a physics-based 3D game, I should use a game engine capable of 3D and physics. I decided to look into four possible solutions; Unity, Unreal, Godot, and UPBGE.

Starting with [Unity](https://unity.com/)³, it is described as one of the best game engines. It allows for complex creations. It's also very popular with other developers, hence it has tons of tutorials online. However, the added complexity may boost development time as I have no experience with the engine, and learning the engine may inflate my development time.

[Unreal](https://www.unrealengine.com/en-US)⁴ is similar to Unity but is less popular and easier to learn. Its graphics capabilities are also extremely advanced. However, it suffers from the same issues that Unity does for this project, as I need to gain experience.

[Godot](https://godotengine.org/)⁵ is a game engine I have experience with but is less popular than Unity and Unreal. Regardless, it still has tons of tutorials online. In recent years, Godot has skyrocketed in popularity. Since I already have some experience with it, I don't need to waste time learning how it works like Unity and Unreal.

[UPBGE](https://upbge.org/#/)⁶ is a niche game engine and is odd. It is a modified version of Blender to allow for game development. I experimented with it for fun over the 2023 Christmas holidays and received mixed results. I enjoyed experimenting but don't think I want to use it to make a full game. It has no 2D capabilities, making HUD and UI painful to implement.

With my options above, I decided to use Godot for my game. Defining my choice to use Godot now allows me to plan features that are capable in Godot, instead of trying something impossible and wasting time.

³ <https://unity.com/>

⁴ <https://www.unrealengine.com/en-US>

⁵ <https://godotengine.org/>

⁶ <https://upbge.org/#/>

Movement and Perspective

With my game engine decided upon, I needed to research what movement type my game should use.

I decided to look at other indie games to figure out a solution. I looked at the top-rated indie games on Steam. Steam's store page tracks what games are the most popular on their [Top Rated page](#)⁷. These games included *The Binding of Isaac*, *Hades*, *Undertale*, *Papers Please*, *Don't Starve*. From my analysis, most other indie games use a top-down camera scheme. I decided to also go with this camera style. I'll use a top-down camera style in 3D, where the camera is outside of the level looking in.

This choice will allow me to plan features and levels that work with my wanted camera scheme. For example, I will plan my enemies to remain grounded and on the same level as the player.

Development Time

How long does it usually take to develop indie games? I'll need to know how long games typically take to make, otherwise I can plan too much for what time I have.

According to [this article](#)⁸ by Arc Academy, the length of game development is driven by several factors. They say that an indie game can take anywhere from 6 months to 3 years, depending on the scope of the game. If I manage to keep my game scope small, I should theoretically be able to complete my game in roughly 6 months. However, this is only theoretical and doesn't help much.

I decided to look at another indie game to figure out how much time I may have to develop. Indie developers do not tend to catalogue how long their projects take to develop fully. Due to this, finding a decent estimation is difficult. I have managed to find one source of information though. I looked into how long *Lethal Company* took to develop. I checked the [developer's Patreon page](#)⁹ for the first signs of when they started development on Lethal Company. He first started on Lethal Company on May 12, 2022, and released Lethal Company on October 24, 2023. Although Lethal Company is a multiplayer horror game, the scope is considerably large.

⁷ https://store.steampowered.com/tags/en/Indie?flavor=contenthub_toprated

⁸ <https://arc.academy/how-long-does-it-take-to-make-a-video-game/>

⁹ <https://www.patreon.com/zeekerss/posts>

Since I know I only have roughly 6 months to develop my game, I need to make my game considerably small. To create a game in a tiny amount of time, I need to plan small.

Game Platform

What platform should I eventually release my game on?

I could make my game for PC, or I could release my game for a console, like Xbox or PlayStation. However, releasing a game for a console is a long, tedious process, as detailed by Xbox in [this YouTube video](#)¹⁰. For Xbox, the process involves joining 'ID@Xbox,' which requires applying and getting approved, according to Xbox. This is very unlikely to happen and will take a very long time. Applying for ID@Xbox, for my game, likely isn't worth the trouble.

PlayStation also requires an approval process to become a PlayStation Partner. They address how to do this in their talk at [Unity Unite 2017](#)¹¹. However, they charge money for their DevKit, which they say is 'similar to a high-end PC' (around 2-3 thousand dollars). This, similar to Xbox, is probably not worth it.

With consoles out of the question, this leaves me with PC and Mobile. Both are easier to create a game for than the consoles. However, I do not want to make a mobile game. This is just my personal preference, as I enjoy the freedom and customization that PC gaming offers. The controls are awkward while playing on a tiny phone screen, and the performance would be terrible. Designing a game for this would be painful. Due to this, I'll create and release my game for PC.

With PC, I have two main options for publishing my game: Steam and Itch.io. According to [this article](#)¹² by Steam, which is about joining the Steamworks Distribution Program, uploading to Steam, similar to consoles, requires an application process and a \$100 fee per game you upload. Since I'm broke and am not expecting to make any money off of this, I won't upload my game to Steam.

Instead, I'll upload my game to Itch.io. Uploading to Itch is detailed [here](#)¹³. Uploading my game to Itch is free and doesn't require an application process (although it sort of does if you wish to charge money).

¹⁰ <https://www.youtube.com/watch?v=mhv-C3WP9Cg>

¹¹ https://www.youtube.com/watch?v=-M76_buQY9I

¹² <https://partner.steamgames.com/steamdirect>

¹³ <https://itch.io/developers>

Deciding upon my desired platform now allows me to know what I need to consider in terms of development. For example, I'll need to consider performance when developing for mobile, and I'll need to consider controls for any controller for consoles.

Analysis Conclusion

To summarise everything, I'll make a game that;

- Creates and maintains a mental state of 'flow' within the player
- It is made in Godot
- Uses a Top-Down camera scheme
- Is intentionally planned small in scope
- Gets released on Itch.io for PC

Game Proposal

Game Concept

From my previous research, I will make an engaging top-down, action-adventure game created in Godot and published for PC on Itch.io.

Since I need my game to be simple, I will do one main player mechanic and centre everything around that one mechanic. As stated before, I also want my game to be physics-based, so this mechanic should be able to influence physics. I eventually decided that my main game mechanic would be the player's ability to pick up and throw any object in the level. I gained inspiration for this mechanic from [Half-Life 2](#) and [The Upturned](#), as they both use this mechanic extensively. This should be somewhat simple since Godot handles most of the physics.

This plan may be risky: My overall plan may be too large for the time I have. As stated before in my research, I only have 6 months to develop my game. This, for an indie game, is very short. I will attempt to circumvent this by sticking to my plan and working quickly.

Objectives

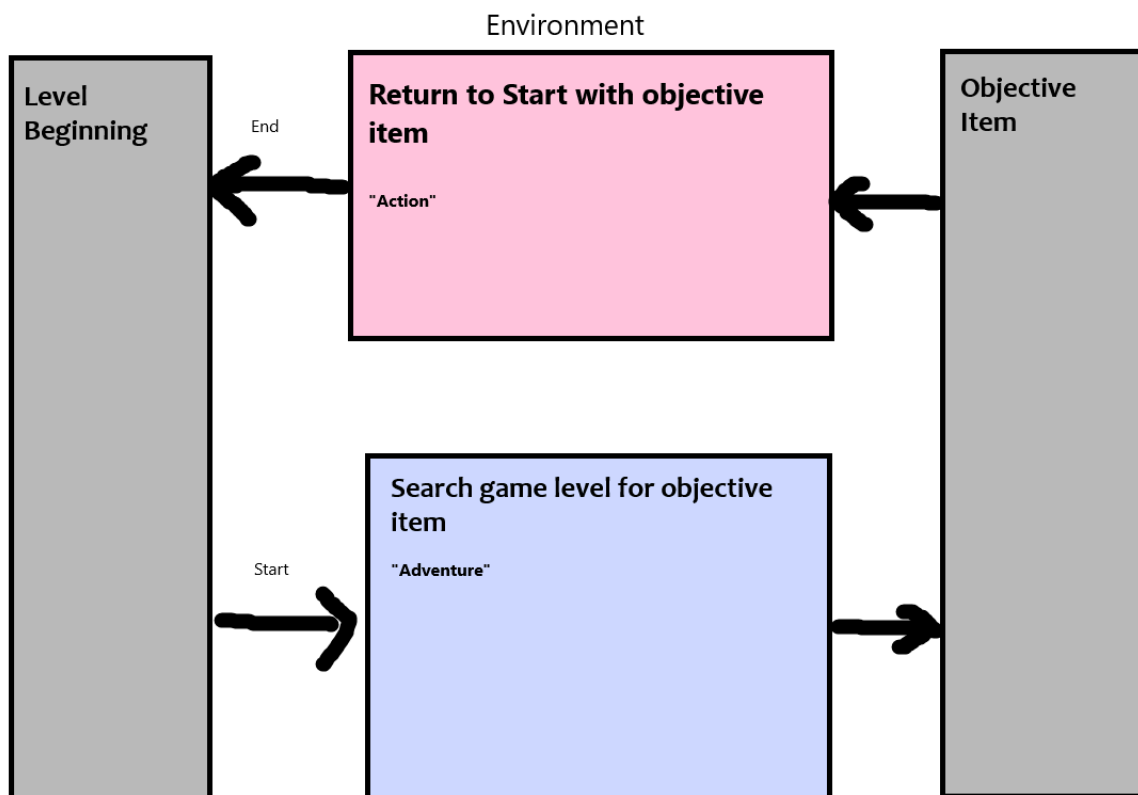
To have a game, I need to give the player a goal. This goal also has to be challenging to prevent the player from getting bored, but also simple to avoid overwhelming them. Since my main game mechanic is the ability to pick up and throw any physics object, I decided that my game's objective would be to find an 'objective item' within the game level and return it to the start of the level. This

objective item would behave like any other object the player can grab and throw. This objective is rather simple and easy to understand (if I manage to teach it well to players)

From this game objective, I decided to create a rough chart of how each level should play out (see diagram below).

I want each level to play out calmly for the first half. After they find the objective item, the level's main challenge will appear. I want most levels to play out like this, as this balances the 'action' and 'calm,' preventing players from feeling overwhelmed by constant action or bored from too little action. I don't want every level to use this framework, because the levels would become too predictable.

Rough Diagram of the framework I want for each level



View full image: [improved-level-chart.png](#)

Target Audience

I want my audience to be mostly older teenagers or adults. This is because, according to [Statistica](#), the majority of the gaming market is made up of teenagers

and adults. This majority also heavily enjoys the Action-adventure genre, which is the genre of game that I'm making.

Appealing to the majority is essential, as I'll have an easier time getting actual people to play my game. That way, I can receive feedback that isn't biased in any way, due to any affiliation with me. Appealing to this age group is also easier because I am also a part of that age group.

Relevant Implications

Usability

Usability means that the game is interacted with simply and understandably for the user.

I need to ensure that people understand the game's control scheme and that the control scheme is not confusing. This also means that my game's UI is usable and simple to navigate.

Failing to ensure my game is usable may cause players to stop playing due to frustration with the game's controls. Any issues with my game's HUD and UI may also cause this same frustration that leads to the player quitting.

To ensure this is not an issue, I will use control schemes commonly used in other games (e.g. WASD for movement, E or F to interact, etc.). This prevents confusion and frustration in players, which could cause the player to stop playing. I will also address [Jacob Nielsen's 10 Usability Heuristics](#). This means including information about the system status, keeping UI efficient, and others.

Functionality

Functionality means that all aspects of the game work as intended. I'll address this with my player controller, including my main game mechanic. Also, I'll need to ensure that my main game objective works as intended. Functionality is essential because, without it, the game would frustrate the player if they continuously ran into bugs and unintended behaviour. This will always have to be in my mind when designing and building the game. When I develop content for the game, I'll always have to be testing for bugs and unexpected behaviour. I must program effectively.

Aesthetics

Aesthetics mainly refers to how the game looks and feels. I'll need to control how the game looks, and how the game feels to play. This affects my entire game, from my level design to UI design. For my levels, I need to create worlds that look visually appealing and immerse the player. For my game UI, I need to make it look, again, visually appealing while simultaneously making it usable and well-designed. Addressing my game's aesthetics is important because nobody would want to play a

game that looks hideous. Aesthetics, in game development, can easily be overlooked.

Before working on my game's art, I'll create appealing concept art. Then, from the concept art, I'll create the game assets. This prevents me from creating a 'temporary solution' that then ends up becoming permanent. As for the game's feel, I'll address this with my game's MDA. I'll use the MDA framework in my development process to control how my game feels to the player.

MDA Analysis

As stated in my analysis, I'll use the MDA framework to control how my game feels to the player.

Mechanics

As stated in the game concept earlier in the proposal, I want my game's main mechanic to be the player's ability to pick up and throw any object on the level. I want every other mechanic the player interacts with to interact with the throwing mechanic in some way.

Dynamics

In terms of how the player uses the main mechanic in gameplay, most game elements should interact with it in different ways. For example, a way for the player to defeat enemies is for them to throw an object at it. Alternatively, I could allow creatures to be picked up and thrown away by the player. A different way the player could use it is to pile objects up to reach somewhere new.

Aesthetics

The overarching feelings that I want the player to feel are a sense of discovery, challenge, and sensation. A main theme I want with this is for the player to feel clever.

My game will utilise discovery because the player will continuously be discovering new game mechanics. Throughout the game, I'll constantly introduce new mechanics to the player. Each level should have a purpose, whether it's a subtle introduction to a mechanic to let the player figure out how it works, or if it's expanding and

I'll make the player feel challenged by constantly forcing them to utilise newly learned mechanics to solve puzzles, fight enemies, etc. Players must use everything they've learned to their advantage– whether it's certain enemy behaviour or properties of certain game elements.

Overall, I want the player to figure things out mostly on their own. The game will push things towards the player for them to interact with it, but there won't be any direct tutorial (aside from teaching them the very basics). My goal here is to allow the player to figure it out themselves, and then a dopamine hit when they do.

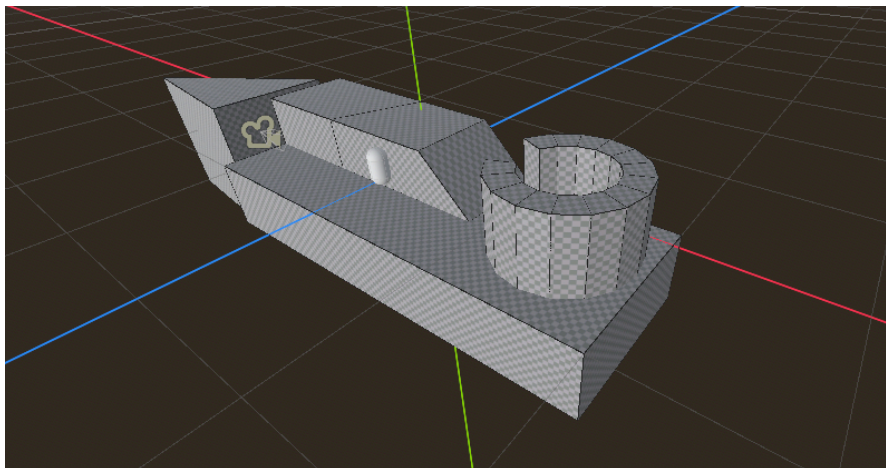
Resources and Tools

From my research, I decided that I'll be using Godot for my game engine. I will also need other tools apart from Godot.

Since my game is 3D, I'll need 3D assets, like models, textures, and levels. I was originally thinking of using assets under the Creative Commons 0 licence from websites like itch.io and TheBaseMesh.com. However, finding assets that all fit the same aesthetic was a struggle. Things may look out of place. Instead, I decided to 3D model the assets myself. These models would be quite simple anyway, and shouldn't take too much time out of development. For the assets that require 3D modelling, like models and levels, I'll use Blender. I decided upon Blender because it's software that I already know. Additionally, these models are easy to import into Godot because of Godot's .blend file importing system, allowing it to recognise and use any .blend files.

For level design and creation, I'll need to use a 3D tool. I was originally thinking of just using Blender for my levels like I'm already doing for my 3D assets. However, my digital technology teacher introduced me to a Godot plugin and community-made tool called Cyclops. Cyclops is a tool designed for level creation within Godot. I trailed with it in a blank project to see if it was worth my time. It worked fine and I made the test level below for demonstration purposes.

Image of a test level I made in Cyclops



However, my game has an isometric camera. With Blender, I can enable backface culling, which makes the backsides of walls invisible. That allows the camera to be outside the map and look into it. Cyclops has no way of doing this. I could use a complex material to hide certain parts of the wall to allow the player to see, but that likely wouldn't be worth the effort. Blender would also allow me to decorate my game's levels much easier. I could directly model decor within Blender, which I can't do with Cyclops. For these reasons, I'll use Blender for level creation and development.

For my game's texturing, I used Godot's Shader Editor. This is because I couldn't use Blender's shader editor. Godot does not import shaders from Blender's shader editor. Blender and Godot use very different shading languages. Godot can only import Blender materials with simple colours or pre-baked image textures. I can still use Blender to make the materials, but textures would need to be baked to appear in Godot, which is a hassle. Because of this, I won't use Blender to create textures and shaders for Godot. Instead, I'll use Godot's shader editor. This solution allows me to create procedural textures like I would be able to with Blender, but

For project management, I decided to use Outlook To-Do. I found this easier on myself since I already used Outlook To-Do for things like managing homework and tasks in general.

In case things go wrong, I'll also need a version control system. I'll use GitHub for my game's version control. GitHub is a cloud-based platform for software development that allows for version control and collaboration. Since I'm working alone, I will not be using the collaboration part of GitHub. I will be using it for its version control. I'll also use it to store my project online, so I can work on my game on multiple different computers (i.e. on my home computer or my school's computers)

Requirements

- My game must be beatable.
- My game must be easy to learn.
- My game must be engaging with the player.
- My game must be a top-down 3D game.
- My game must allow the player to pick up and throw physics objects.

Development:

Game Design Documentation

From my game's proposal, I'll now create a Game Design Document that solidifies my plan. This is essential for my development process, so I don't go off track developing something irrelevant.

Synopsis

Amelioration is a chaotic physics-based, Action-Adventure game, where you can pick up and throw anything.

Game Objective

Your goal is to find the egg and return it to A-OS, your omnipotent overlord, for incineration. You're destroying these eggs 'to make the world a better place,' or so you are told (which is why the game is titled Amelioration. 'Amelioration' literally means 'the act of making something better.').

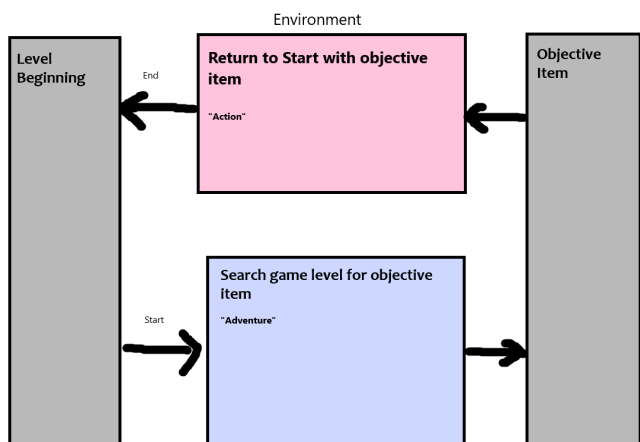
At each level, you explore the area to find the egg. Upon finding it, you return it to the start of the level, to incinerate it. Although, most of the time, there'll be something trying to stop you.

Game Structure

Amelioration's level structure is very linear, meaning after completing a level, you simply move on to the next level.

As for level structure, my game is classed as an action-adventure game, so I need to incorporate both action and adventure. I decided to have a rough level structure of the player beginning each level with an adventure phase. They'll have to find the objective item here. Then, when they find it, an action phase begins as they make their way back to the beginning.

(Right) Level structure diagram, shown earlier
View full image: [improved-level-chart.png](#)



I want each level to introduce or expand upon a new mechanic for the player. I think I could introduce a new mechanic in the adventure phase, and then challenge the player by testing/challenging them with the mechanic in the action phase. Due to this, most levels will be tutorials, but constant tutorials would be annoying for the player, wouldn't it?

Making good tutorials is hard, as it can annoy the player easily, so I decided to figure out what other games do and copy their methods.

Half-Life 2 doesn't have a formal 'tutorial,' it teaches you throughout the gameplay—and it works. Half-Life 2's tutorials mostly use a 'show, don't tell' method. [This clip](#) shows the beginning of the Ravenholm chapter in Half-Life 2. The player finds a zombie's corpse sawn in half by a saw blade. They then have to clear a path to progress and, as soon as they remove a saw-blade from the wall, a zombie walks around the corner, begging to be cut in half. This indirect form of teaching the player appeals to me, as it doesn't break the player's immersion by sitting through a boring tutorial. Half-Life 2's approach mixes the tutorial with the gameplay, thus the player in this scene isn't consciously aware that it is a tutorial.

I'll implement this method throughout the main gameplay. However, Half-Life 2 still uses a direct approach to teaching the player when it comes to basic movement, jumping and simple physics interaction. Those baseline skills are taught at the beginning of the game. Those skills are essential, as everything involves the player understanding those mechanics. I'll also implement this direct tutorial at the very beginning of the game, as everything also revolves around the basic skills in my game.

Game Controls

Amelioration uses a simple WASD control scheme for its movement, with Arrow keys also allowing movement, if that's your preference. Space is used for jumping, and mouse movement is used to look around.

The ability to pick up and throw objects is all bound to left click. To pick up an object, press left click. To continue holding the object, continue holding the left click. When you release the left click, the object is thrown. If you wish to drop an object gently instead of throwing it, look down at the ground and then release the left click.

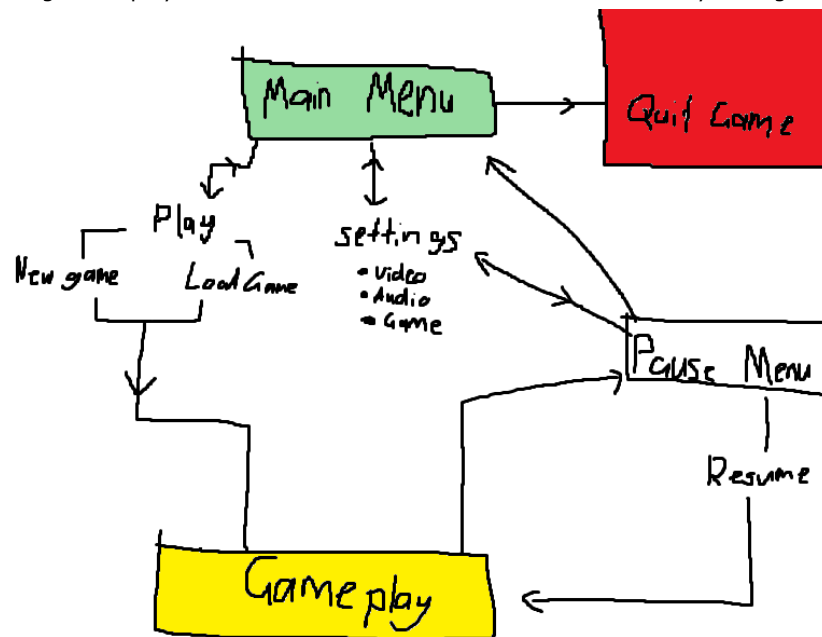
User Interface

For my game's user interface, I intend to keep it minimalistic and out-of-the-way.

My main idea for the user interface is usability. I'll attempt to predict what the player wants to do in given scenarios, and assist them or make the button bright and obvious. For example, if the player just opens the game, the game will have the start button in the centre of the screen, bright and obvious.

In terms of UX, my goal is to keep it as streamlined and simple as possible. I created a chart of how the player interacts with my game's menus to ensure that it is as streamlined as possible before creating it.

UX Design. The player will start at the main menu and work their way through the menus in this structure.



Link to full image: [ideation_ux.png](#)

Game UI Design

<p>Proposed design of game title screen</p> <p>Full Image: ideation_mainmenu.png</p>	<p>Proposed design of Pause menu</p> <p>Full Image: ideation_pause...</p>	<p>Proposed design of game HUD</p> <p>Full Image: ideation_hud.png</p>
--	---	--

Finalised Requirements

For my game to be considered complete;

- My game must be completable
- My game must be easy to learn
- My game must be engaging with the player
- My game must be a top-down 3D game
- My game must allow the player to interact with physics objects

Specifications

To meet each requirement for my game, I'll break each one down further.

For my game to be completable;

- The game will have an ending
- Each level will be completable
- Each level will have some purpose behind it (e.g: introducing a new enemy)

For my game to be easy to learn;

- The game will have an initial tutorial
- The game will have a mini-tutorial for each new element produced
- The game will use a 'show, don't tell' tutorial methodology (similar to how Half-Life 2 teaches the player.)

For my game to feel engaging with the player;

- The player will feel clever and resourceful while playing (from my MDA aesthetics)
- The player doesn't get bored while playing, or get too overwhelmed while playing.
- The player doesn't get frustrated at my game, from either challenges that are too difficult, or the game doesn't function correctly at points.

For my game to be a top-down 3D game;

- The game will have an isometric, top-down camera scheme
- The game will be made in 3D
- The player can see what is happening clearly

For my game to allow the player to interact with physics objects

- The player will have the ability to pick up physics objects
- The player will have the ability to throw these physics objects
- The game will base other game mechanics based on the player's physics interaction

Development Cycles

AGILE Development Process Overview

For the development of Amelioration, I will use the Agile development workflow.

Development workflows are important because they ensure clarity and efficiency throughout my process. There are several different development workflows, like Waterfall. For this project, I've chosen the Agile workflow.

The agile workflow involves 5 stages; define project goals, break down tasks and milestones, assign tasks and responsibilities, create a timeline and schedule, monitor progress and adjust. Since I'm working alone, I won't assign tasks and responsibilities, since everything is dependent on me anyway. This workflow is great because it is very flexible. For example, in the last stage, I can make necessary adjustments to keep my project on track.

I was considering using Waterfall for my project. I never ended up using it because I found Agile better in terms of flexibility. Waterfall development is very linear and rigid. Therefore, it's not very flexible, and it's difficult to make necessary changes to your project later in development. Because Agile has better flexibility, I was drawn to it more and ended up using it in my development.

Each stage of my development process is going to be broken down into 3 'sprints.' Each sprint is a major step in development with a set of goals. After each sprint, I'll playtest what I have after each sprint to figure out what needs changing.

I want sprint 1 to be my 'main prototype.' I'll create a main block-out for my game, with functioning mechanics with simple levels and enemies. There will be no art done in this sprint because it's recommended that my game should be fun without game art.

I will implement basic art and some storyline in sprint 2, along with any feedback I receive from sprint 1. This sprint is about coming up with a short demonstration of what I want my final game to look like.

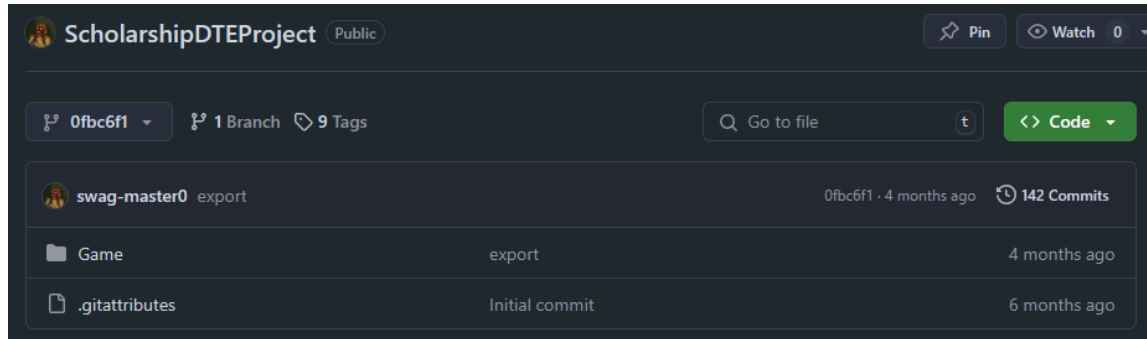
Sprint 3 is all about finalising my ideas into a finished game. I'll add whatever feedback I receive about sprint 2. Although, I'll mostly be adding more content to my game, polishing what content I have, and bug fixing.

I have to finish my game by October. I have roughly 4 months to develop, due to lost time with planning my game. I've decided that sprint 1 and sprint 2 should last 1 month, and sprint 3 is the rest of the time I have (i.e. roughly 2 months).

Sprint 1

All of the progress made in this sprint was saved onto GitHub for version control and online file storage. The GitHub repository at the end of this sprint can be found [here](#). There were 142 commits made in this sprint.

Overview of GitHub repository at this stage.



GitHub commits made throughout the sprint, detailing changes made within them.



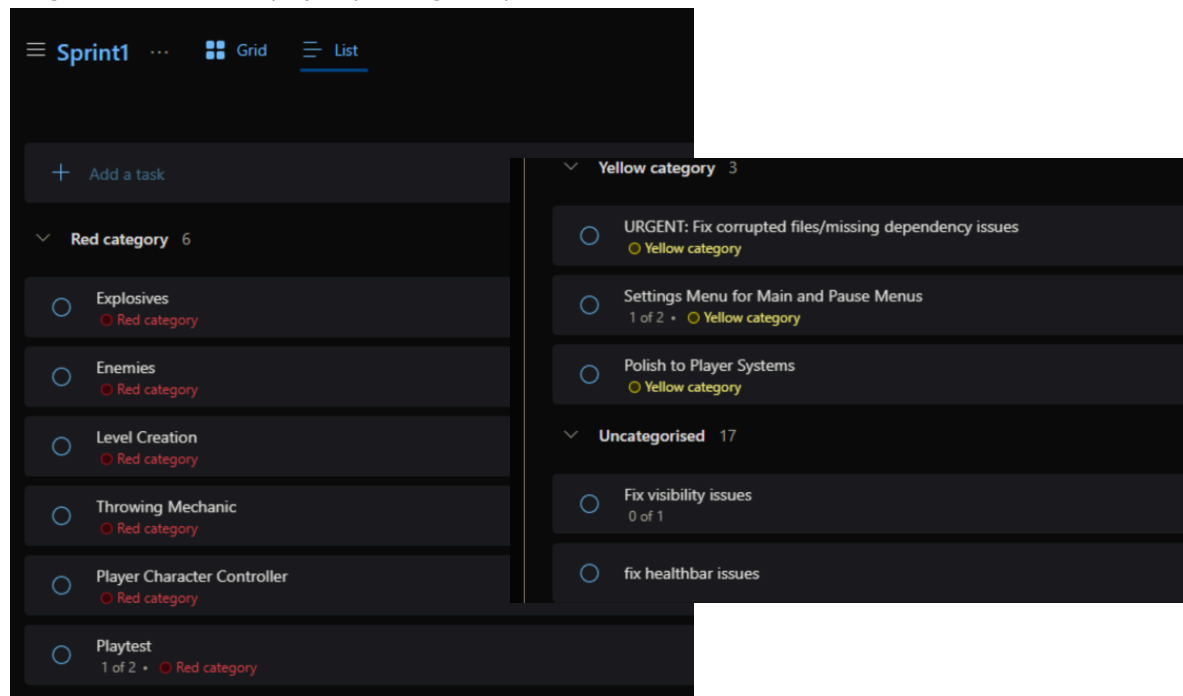
Goals

With sprint 1, I aim to complete a basic prototype of my game. Overall, for my first sprint, I will;

- Create the character controller.
- Create the main game mechanic (i.e. the ability to pick up and throw objects).
- Create other game mechanics.
- Create the game's basic tutorials.
- Create the game's levels.
- Create the game's enemies and challenges.

As stated prior, I have roughly one month to complete this sprint.

Image of Outlook To-Do project planning for Sprint 1



Development Towards Goals

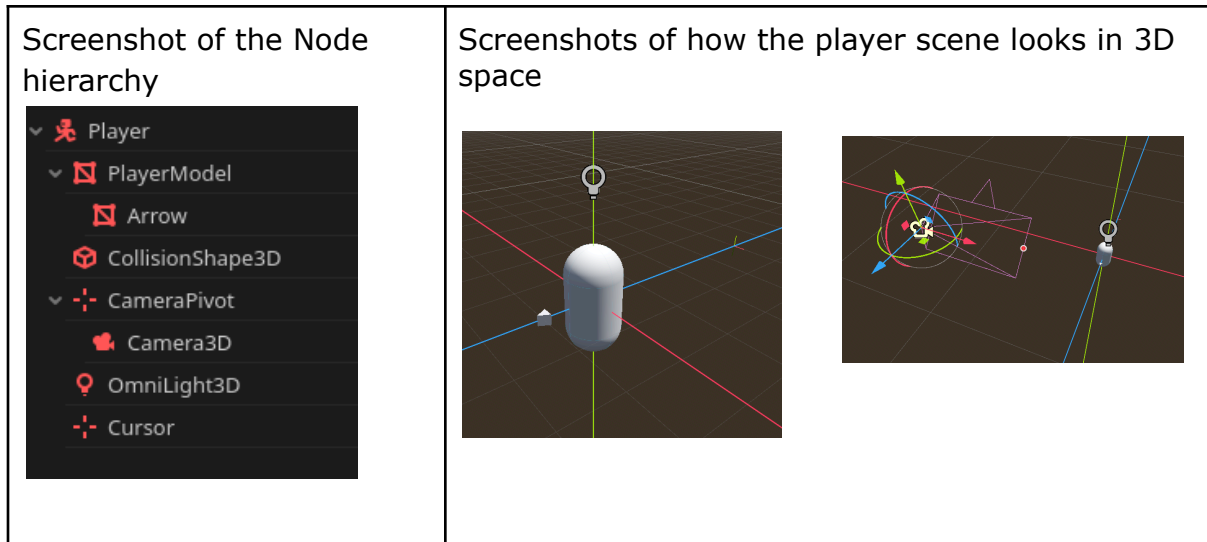
Character Controller + Player Throwing Mechanic:

I'm including these both together because they heavily relate to each other.

I started this sprint by first creating the player's main movement system. I needed to finish this first, as every other game mechanic will revolve around how the player interacts with them.

Godot uses a node-based system, where everything is made from nodes, the 'game's smallest building blocks.' You then use these nodes to create scenes. I created my player scene using the **CharacterBody3D** node as the root node (i.e. the central node).

From that root node, I attached a **CollisionShape3D** node (so the player collides with the world), and a **MeshInstance3D** node (so the player can see themselves), which was renamed to 'PlayerModel,' a **Camera3D** node, attached to a pivot point with the player, and an invisible Cursor node (i.e. a **Marker3D**, which isn't visible in-game). The cursor exists as a way for the player to pick up and throw objects (more on this later, when discussing the main throwing mechanic)



On the root note (i.e. the CharacterBody3D), I have a script attached to run all of the player movement.

I created the player's code by starting with the player template script that comes built into Godot. This GDScript code has very basic player movement, allowing me to build upon this script without wasting time doing this from scratch. GDScript is Godot's built-in scripting language, designed for use in Godot.

The template Godot movement script:

```
Python
# Portion of player.gd;
extends CharacterBody3D
...
# Runs every physics tick (should be 60 times per second)
func _physics_process(delta):
    if not is_on_floor(): # Handles player falling
        velocity.y -= gravity * delta * JUMP_FALLMULTIPLIER
    if Input.is_action_pressed("jump") and is_on_floor():
        velocity.y = JUMP_VELOCITY

    var input_dir = Input.get_vector("left", "right", "forward", "backward")
    var direction = (transform.basis * Vector3(input_dir.x, 0,
input_dir.y)).normalized()

    if direction:
        velocity.x = direction.x * SPEED
        velocity.z = direction.z * SPEED
    else:
        velocity.x = move_toward(velocity.x, 0, SPEED)
```



```

        velocity.z = move_toward(velocity.z, 0, SPEED)
    move_and_slide()
    ...

```

Sidenote with this code block and future code blocks: This isn't written in Python, it's written in Godot's native scripting language- GDScript. However, Google Docs doesn't allow me to show that and forces me to show it as Python. Sorry for any confusion.

This script handles simple movement for the player, using 'left,' 'right,' 'forward,' and 'backward' button presses to determine movement. These names are set up in Godot's input map, within its Project Settings menu. It is constantly checking if the player is pressing a button that corresponds to a direction the player should travel, and then move the player in that direction.

Link to a video of the player's movement:  trial-movement.mkv

I implemented cursor movement in this script. The cursor is constantly placed where the player's mouse is in 3D space. To do this, I wrote a simple function that calculates where the mouse is pointing in 3D space and returns the location.

The function that fetches the player's mouse in 3D space. It fires a ray cast from the camera to the location where the ray is colliding.

```

Python
# Portion of player.gd;
...
func MousePosition():
    if ready and canPause:
        # Creates Raycast
        var mousePos = get_viewport().get_mouse_position()
        var spaceState = get_world_3d().direct_space_state

        var rayOrigin = camera.project_ray_origin(mousePos)
        var rayEnd = rayOrigin + camera.project_ray_normal(mousePos) * 2000
        var query = PhysicsRayQueryParameters3D.create(rayOrigin, rayEnd)

        var rayArray = spaceState.intersect_ray(query)

        # If the raycast hit something
        if rayArray.has("position"):
            cursor.global_position = rayArray["position"]
    ...

```

The function is then used for the throwing mechanic, to allow the player to pick up a specific object, and throw it in an aimed direction. The mechanic will allow the player to pick up any RigidBody3D object in the level, which are bodies that handle physics. I created the code for the player throwing within the `_process()` function, which runs every frame. It checks if the player is close enough to the object, the cursor is on the object, and the player isn't already holding an object. If all of these are true, the script waits for the player's click to pick up the object.

Upon picking up the object, it quickly reparents the object to the player, so it is bound to the player. It also disables the object's physics and disables its collisions. Upon throwing the object, it quickly undoes these changes and applies a force to the object so it is thrown towards the player's cursor.

Video of the player throwing mechanic:  [trail-throwing-2.mp4](#)

The portion of code that handles the player's throwing mechanic

```
Python
# Portion of player.gd;
func _process(_delta):

#     NearestObject() returns the closest object to the cursor
    if (NearestObject() and NearestObject() is RigidBody3D and
        global_position.distance_to(NearestObject().global_position) <= PICKUP_RANGE) and
        isHolding == false:
#         Prevents the player from picking up objects I mark as ungrabbable
        if NearestObject().is_in_group("ungrabbable"):
            return

#         Detects clicks, and then picks up the object
        if Input.is_action_just_pressed("click"):
            isHolding = true
            object = NearestObject()
            # saves the old parent for when the player lets go
            oldparent = object.get_parent()

            holdpoint.position = Vector3(0, 1, 2)
            object.global_transform = holdpoint.global_transform
            object.reparent(character) # reparents
            object.set_freeze_enabled(true) # disables physics
            object.set_collision_layer_value(1, false) # disables collisions
            object.set_collision_mask_value(1, false)

#         Checks if the player is no longer holding click, and then throw the object
        if !Input.is_action_pressed("click"):
#             Checks if the object exists still, or else the game will crash
            if !is_instance_valid(object):
```

```

        object = null
        isHolding = false

#         Throws the object
        if object and isHolding == true and object is RigidBody3D:
            isHolding = false
            object.set_freeze_enabled(false) # reenables physics
            object.reparent(oldparent) # reparents the object

            object.set_collision_layer_value(1, true) # reenables collisions
            object.set_collision_mask_value(1, true)

            # applies a force to the object, in order to throw it
            var force = (cursor.global_position - position).normalized()
            object.apply_force(force * THROW_FORCE)

#         Resets the object variable
        object = null

```

Enemies and the Info Component:

To make the development of future enemies simpler, I decided to make a simple 'baseline' enemy that did the basics of what I wanted out of an enemy. These basics include following the player when it sees them and taking damage from thrown objects since both actions will be common with other enemies.

The basic enemy will simply follow the player when they see them, and take damage when hit with a rigid body. The 'take damage' part is tricky since I need to make it consistent across all enemies.

What I did instead was create a separate component scene that handles the health and damage of enemies, and the player. I called it 'info,' which, in hindsight, was a terrible name choice. It is essentially an external hitbox for each enemy and the player and handles the health and damage for me.

When the hitbox is collided with by the enemy, it triggers the following signal;

```

Python
# Portion of info.gd;
...
func _on_hitbox_body_entered(body):
#     Checks if the colliding object is a physics object
    if body is RigidBody3D and TakeDamageFromRigidBodies == true:

#         Checks that itself still exists, and checks if it's the player

```

```

        if self.get_parent() and self.get_parent().is_in_group("player"):
#             if it's the player or it doesn't exist, then ignore damage
                return
        else:
#             The Damage() func mainly handles the damage
            Damage(calculateDamageBasedOnVelocity(body))
...

```

Essentially, this code checks if the colliding object is a rigid body and the info node does not belong to the player. If so, it will deal damage to the enemy depending on the velocity of the object. When dealing damage, it runs the amount to take damage through the `Damage()` function. This is important because this function handles the owner's death and also runs the signals that communicate with the owner of the component.

The `calculateDamageBasedOnVelocity()` function calculates the damage that should be dealt to the component owner. It calculates the magnitude of the object's velocity when it collides with the info component hitbox and returns that value as the damage. This means that the faster that object is travelling, the more damage is dealt.

The damage function

```

Python
# Portion of info.gd;
signal takeDamage
signal death
...

func Damage(damage: float):
#     This timer acts as some sort of 'invincibility frames,' preventing damage
    if timer.is_stopped():
#         The 'takeDamage' signal is called when this occurs, and is used to
        communicate with the enemy that this info component belongs to
            takeDamage.emit()

            health -= damage

            timer.start() # Restarts the invincibility frames

# Checks if the health is less than or equal to 0, and calls the death signal if
so
    if health <= 0:

```

```

        death.emit() # Called when this occurs, and communicates with the enemy
        that this info component belongs to
    ...

```

The calculateDamageBasedOnVelocity function, which calculates the magnitude of the velocity and returns it

```

Python
# Portion of info.gd;
...
func calculateDamageBasedOnVelocity(body):
    if body is RigidBody3D:
        #         abs() is used to ensure that this value is a positive number, otherwise
        #         it'll return Not A Number when we square root it later
        var vel = abs(body.get_linear_velocity())

        #         calculate magnitude of the force
        var magnitude = sqrt(vel.x + vel.y + vel.z)

        #         do not damage if the magnitude is very small and shouldn't deal damage

        if magnitude > MagnitudeThreshold:
            return magnitude
        else:
            return 0
    ...

```

The info component is used for all killable enemies and the player.

As stated, the info component communicates with the owner through signals, which I can connect to actions in the enemy's script.

The portion of the enemy's script that detects when the info component emits the given signals

```

Python
# Portion of enemy.gd;
...
# Activates when the info node emits the signal 'takeDamage'
func _on_info_take_damage():
    #         plays a brief visual animation to show that the enemy got hurt
    $AnimationPlayer.play("hurt")

# Activates when the info node emits the signal 'death'
func _on_info_death():
    self.queue_free() # deletes itself

```

This is how I manage enemy health for the player and every enemy in the game.

As for how the enemy moves, it starts by using a Raycast3D node in the centre of the enemy. The target position of this ray is constantly set to the player's current position. If the ray collides with a wall before reaching the player, then the enemy cannot see the player and does not move. However, if the ray does collide with the player, the enemy starts moving towards the player.

It moves towards the player using Godot's built-in navigation system. For the level where I want my enemy to be, I create a navigation mesh using a NavigationRegion3D node. I can then utilise the navigation system using a NavigationAgent3D node I put within the enemy scene. The NavigationAgent3D node allows me to access methods to navigate my enemy towards the player.

The portion of code that searches for the player in the scene tree, sets the target position to their position and moves towards them if the ray cast collides with the player.

Python

```
# Portion of enemy.gd
func _physics_process(delta):

    var direction = Vector3()

    # Constantly searching for the player (using the world's worst code to do so), and
    # setting the raycast target position to the player's position
    # There's likely a better solution for this
    for i in self.get_parent_node_3d().get_children():
        if i is CharacterBody3D and i.is_in_group("player"):

            var player = i

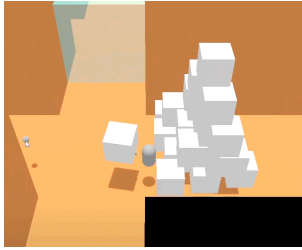
            raycast.target_position = player.global_position - global_position

            if raycast.get_collider() == player:
                # nav is a variable that accesses the NavigationAgent3D node
                nav.target_position = player.position
                # the navigation is trying to reach the player's position

            raycast.global_position = global_position

    # follows the path calculated by the NavigationAgent3D node to reach the player
    direction = nav.get_next_path_position() - global_position
    direction = direction.normalized()

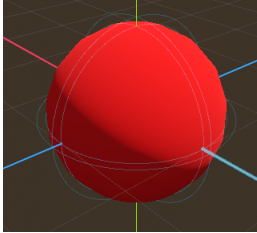
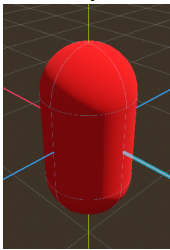
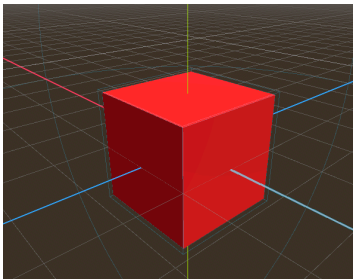
    move_and_slide()
```



Video of basic enemy in action: [trial-enemy.mkv](#)

Off this, I created a whole bunch of other enemy ideas. There are more working prototypes since this is a 'create a general prototype' sprint.

Other enemies I created include:

<p>'Bomber'</p> 	<p>An enemy fundamentally similar to the basic enemy, but explodes upon hitting the player and on death.</p> <p>I wanted to make the player tread carefully when around it, and maybe kill it when other enemies are around to kill them too– playing into the MDA by trying to make the player feel clever by finding clever workarounds to it.</p>
<p>'Sentry'</p> 	<p>An enemy that stays completely still, but throws grenades at the player. These grenades are Rigidbody3Ds, so the player can pick them up and throw them back. I'm trying to encourage the player to use these creatively.</p>
<p>'Copycat'</p> 	<p>An enemy that pretends to be a physics object that the player can pick up. This is a Rigidbody itself but is coded so it can't be picked up by the player. It'll fling itself at the player to deal damage when they get close. I'm turning the player's tools on the player to see what happens– like I said, I'm prototyping to see what works.</p>

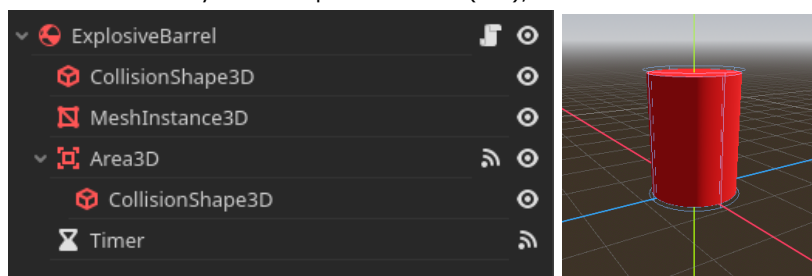
Explosions and Explosive Barrels

To have an interesting and engaging game, I need to have some other game mechanics. I want these to give the base throwing mechanic some depth and complexity.

Firstly, I wanted to make an explosive barrel. The idea was that it was a one-time use but deadly to large groups of enemies, forcing the player to use it carefully. It's also deadly to the player, so they need to carefully use it so they don't damage themselves. There are two components behind this; the barrel that creates the explosion, and the explosion itself (a separate component, for use by other nodes that I want to explode).

The explosive barrel object behaves like every other object in the world, except for the fact I want them to explode upon contact. The root node of the explosive barrel is a RigidBody3D node, so it obeys regular physics; the same as every other object. To detect collisions, I used an area 3D node that's slightly bigger than the object itself and uses a signal to check when something enters that area. When a collision is detected, it checks the speed that the explosive barrel is travelling at that moment and explodes if it is greater than a threshold I dictate.

The Node hierarchy of the explosive barrel (left), and the 3D view of the explosive barrel (right)



The explosive barrel script

```
Python
# The entire explosive_barrel.gd script;
extends RigidBody3D

@export var explosion: PackedScene=
var minimum = 3.5 # the minimum magnitude threshold

func _on_area_3d_body_entered(body):
    var vel = self.get_linear_velocity()
    # convert negative numbers to positive, or else square root returns 'not a number'
    vel = abs(vel)

    var magnitude = sqrt(vel.x + vel.y + vel.z) # calculate magnitude of the force
    # if the magnitude is greater than the required threshold, trigger a timer for a
    # delayed explosion
    if magnitude > minimum:
        $Timer.start(0.1)
```



```

func _on_timer_timeout():
    explode()

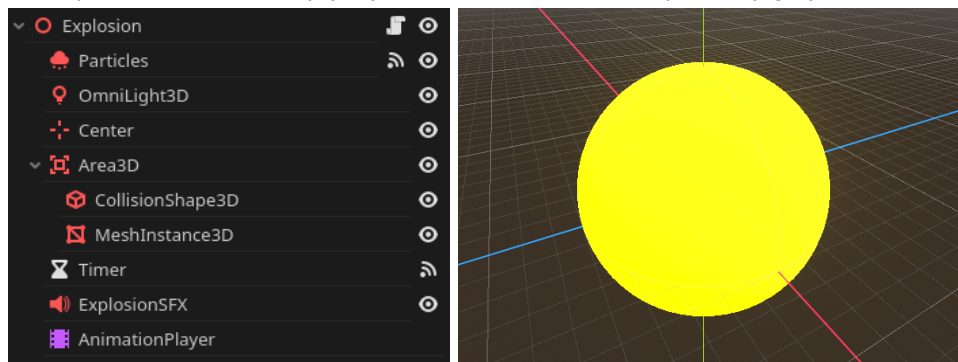
func explode():
    # Creates the explosion scene on the barrel's current position, then delete itself
    var created_explosion = explosion.instantiate()
    self.get_parent().add_child(created_explosion)
    created_explosion.position = position

    self.queue_free()

```

When the explosive barrel explodes, it creates a new scene of the explosion component. This explosion is a temporary yellow ball that damages anything within it and deletes itself after 0.5 seconds. It also flings any objects in the explosion radius away. The script keeps track of what has already been affected by the explosion, so it doesn't damage one object too many times (which would instantly kill the player).

The explosion node hierarchy (left) and the 3D view of the explosion (right)



The explosion.gd script, that handles the code behind the explosion

```

Python
# The entire explosion.gd script;
extends Node3D

# References to nodes
@onready var timer = $Timer
@onready var area = $Area3D
@onready var center = $Center

@export var force : float = 1000
@export var damage : float = 8

var triggered = false

```

```

# This is to track what has been damaged by the explosion, to prevent damaging something
multiple times when it shouldn't
var affected = []

func _process(_delta):
    for i in $Area3D.get_overlapping_bodies():
        if i is RigidBody3D and !(affected.has(i)):
            i.apply_impulse(
                (i.global_position - center.global_position) .normalized()
                * force
            )
            i.set_freeze_enabled(false)
            affected.append(i)

        if (i.is_in_group("player") or i.is_in_group("enemy")) and
        !(affected.has(i)):
            affected.append(i)
# Finds the info component and damages it
    for x in i.get_children():
        if x.is_in_group("info"):
            x.Damage(damage)

    if !triggered:
# Runs the visual components of the explosion
        triggered = true
        $Particles.emitting = true
        $ExplosionSFX.pitch_scale = randf_range(75, 125) / 100
        $ExplosionSFX.play()
        # Reparents the audio, so the explosion sound doesn't suddenly get cut off
        # when it gets deleted
        $ExplosionSFX.reparent(self.get_parent())

# Starts timer for how long the explosion lasts
        timer.start()

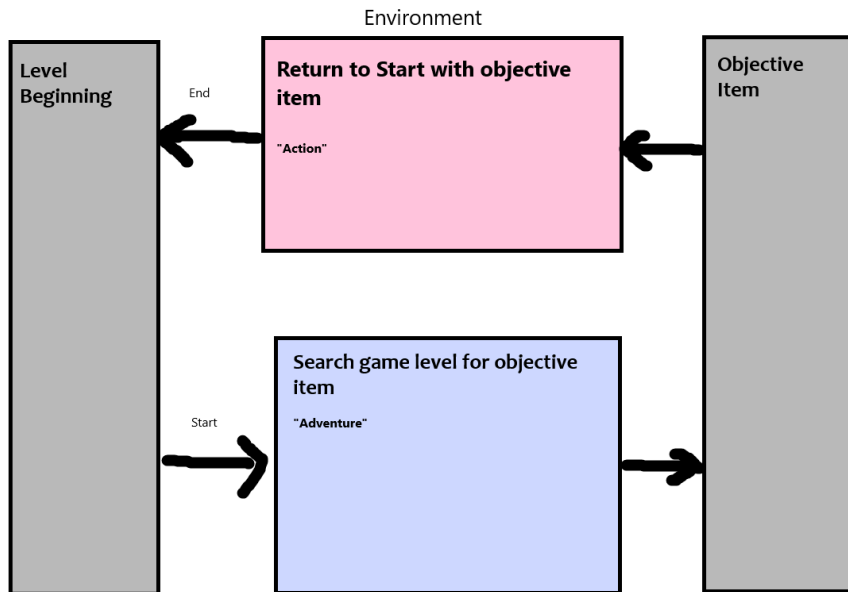
# Deletes the explosion when it has persisted long enough
func _on_timer_timeout():
    self.queue_free()

```

Game Levels:

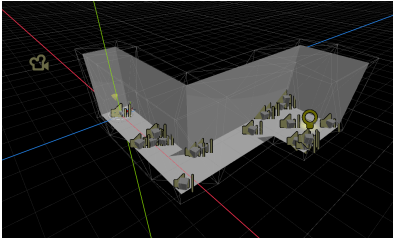
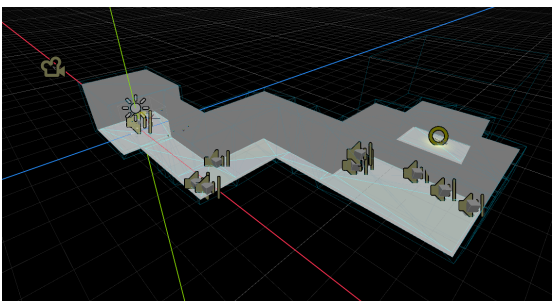
Since I've already decided upon my game's objective– to find the objective item and return it to the level start– I will design each level with that gameplay loop in mind. From before, I've also decided upon the framework I want to follow (see below).

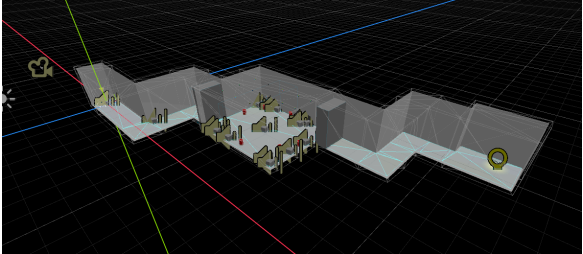
Rough Diagram of the framework I want for each level



View full image: [improved-level-chart.png](#)

I won't go over all of the levels, so I'll only go over the notable ones. There are 8 levels total.

<p>Level 1</p> 	<p>This is one of the only levels that doesn't follow the guidelines above. I want this to be a safe area for the player to learn the main game objective.</p> <p>There's no threats here, just a simple explanation of the game tutorial</p>
<p>Level 2</p> 	<p>This is where I let the game open up more. I introduce the first puzzle here, where the player must stack some boxes to get the objective item.</p> <p>I also introduce the basic enemy here when they return. This is the first level that follows the framework above because of it.</p>

<p>Level 4</p> 	<p>Level 4 is the first sort of 'challenge room.' The player gets the objective item without a hassle but, as they return, their path gets blocked and 25 enemies spawn one by one. The player can use the explosive barrels in the room to make the battle easier.</p>
--	---

Testing and Feedback

Upon finishing this sprint, my digital technology teacher got us to upload our games to the [Techquity NZ High Schools Game Jam](#). This allowed other people outside of our school to play our game and give feedback from an unbiased perspective. We all attached a Google Form to our submission so people who play our games can give us feedback anonymously. I submitted my game at this stage and received 3 responses. This was problematic for 2 reasons;




1. This isn't enough evidence to draw any conclusions from
2. The responses themselves weren't taking it too seriously

Form Responses. Not the best

Questions	Did you complete the game fully?	What did you like?	What did you NOT like?	Did you encounter bugs? If so, what were they?	How long did you play?	General feedback and comments
Answers	Yes;No	The picking up was very cool especially throwing the barrels at the bad balls, also the movement and item physics are quite nice	the picking up was bit janky	the objective got absolutely yjetted at one point across the room	Roughly 20 mins	Great potential
	Yes;No	the boxes mechanics	me not reward :(nop	Roughly 5 mins	make wall backrooms add an basketball rim for an easter egg
	No	the mechanics	nothing	the blocks went through the wall when carried	Roughly 30 mins or longer	10000000000000/10

Instead, I had some friends playtest it. I got them to share their screens over Discord as they played so I could see their gameplay. I recorded this for later use.

The recordings can be found here;

- First playtest with Reid:  reid-1.mkv (My microphone didn't record for some reason.)
- Second playtest with Reid:  reid-2.mp4 (He encountered a bug that didn't let him progress. I fixed it and I let him pick up where he left off)
- Playtest with Thomas:  thomas.mp4 (I let him play the game on his own)

- Playtest with Louis: 📺 louis.mp4

Overall, the major things I found from the playtesting were;

- The main game mechanic and game objective were poorly taught to the player (these were the only things that weren't communicated through the tutorial method I had devised)
- Some levels were too difficult (these are levels that weren't mentioned above)
- Major bugs (e.g. the player could walk through some walls, random explosive barrels spawned for no real reason, etc.)

Reflection

Overall, I think the first sprint went fairly well. Everything I had planned at the start was finished. I only finished a few days after I had scheduled, mostly due to difficulties with the playtesting builds and a file corruption issue.

The playtesting suggested some things I wasn't expecting, like the difficulty of areas and unclear initial tutorials. I've planned to fix these issues in Sprint 2 anyway.

However, I noticed that the game's isometric camera scheme causes issues with gameplay. Firstly, I had to make each level with the game's camera scheme in mind, ensuring that I minimise how many walls block vision. Making levels this way is tedious and annoying. Additionally, the gameplay also seemed tedious with the isometric camera scheme. The platforming was made annoying because of this. Aiming thrown objects also made it difficult to accurately aim. The distance from the player to the camera also makes the player somewhat disconnected. This is a fundamental design flaw that I need to address immediately.

Due to this, I considered changing the game's camera scheme to something else. Most 3D games are either first-person or third-person. I could change my game to a camera scheme players are familiar with. I chose to try a third-person camera to see how it compared with the original isometric- to see if it was worth changing.

Camera Changes


To ensure I keep the old player, in case I wish to revert this change, I did this on a copy of the player. This change involved adding the `_input()` function to the player's script that rotates the camera around the player. This code triggers every time the game registers an input from the movement (whether it's a keyboard press or a mouse moving), and checks if it was a mouse moving.

```
Python
# Portion of player.gd
func _input(event):
    if event is InputEventMouseMotion:
        rotate_y(deg_to_rad(-event.relative.x * SENSITIVITY))
        if !velocity and !isHolding:
            character.rotate_y(deg_to_rad(event.relative.x * SENSITIVITY))

        pivot.rotate_x(deg_to_rad(-event.relative.y * SENSITIVITY))
# Cap the camera's rotation to prevent the player from flipping the camera
upside down
        pivot.rotation.x = clamp(pivot.rotation.x, deg_to_rad(MIN_LOOK),
deg_to_rad(MAX_LOOK))
```

Other than this change, not much else needed to be changed to ensure that this worked. The rest of the player script will remain untouched.

To check if this change was better, I ran another wave of playtesting. I found that the game was easier when the players were using a camera scheme more familiar to them.

- Video to playtest:  louis-3rdperson.mp4 (I'm not too sure why but my microphone is glitchy in this recording)

From the playtesting, I think the gameplay is a lot better for the player. The player's ability to aim is significantly improved, which allows them to utilise the main throwing mechanic more. The player can aim up to arc their throws and hit targets further away.

I think I will replace my current player with this third-person player because of the significant improvements it provides. This means I may have to revisit my game design documentation and correct information due to this change.

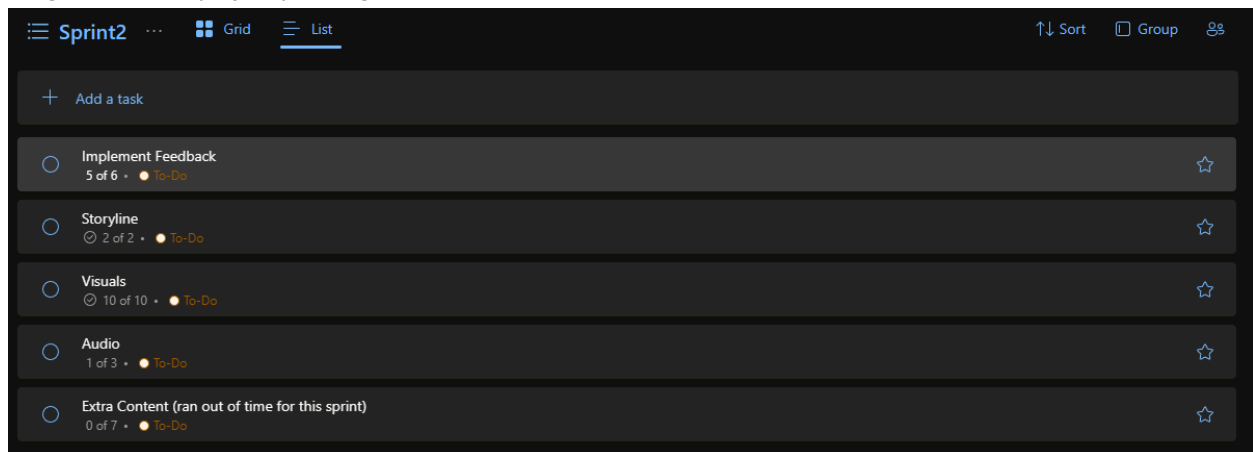
Sprint 2

Goals

In this sprint, I want to make a demo of what I wanted from my final product. To accomplish this, I:

- Implement feedback from the previous sprint
- Create a game storyline
- Design an art style for my game

Image of Outlook project planning



Development Towards Goals

Feedback from Previous Sprint

Several bugs and issues were found in the previous sprint's playtests.

Firstly, some game mechanics weren't taught to the player properly, as playtesters were clueless and required me to intervene and tell them what things did.

Playtesters mainly failed to understand the main game objective of returning the objective item to the start of every level. To fix this, I will make the first level focus on teaching the game's main objective.

Another major issue is that some walls could be walked through. I had the isometric camera scheme in mind when making levels, so I made some walls transparent.

This involved manually flipping the wall's face in Blender, which unfortunately resulted in the wall losing collisions and allowing players to walk through and fall into the void. I indirectly fixed this when I changed my camera scheme to a third-person perspective since I could then revert all walls to face the correct direction, fixing the issue.

Writing / Game Storyline

I was struggling with coming up with a game storyline that I liked. I eventually decided that I wasn't going to come up with something unique, so I took very large inspiration from a short story titled *I Have No Mouth and I Must Scream* by Harlan Ellison (which I'll acronym to IHNMaIMS). IHNMaIMS is about a sentient AI created for war, that destroys the world but leaves 5 humans alive to torture forever for its sick enjoyment. I decided to go a similar path with post-apocalyptic themes and sentient AI.

Unlike IHNMaIMS, I don't want my main character who pushes the player forward to be blatantly evil (like AM). Instead, I want him to be more clueless. I eventually decided to make my character like another character from a different game called [KinitoPET](#). KinitoPET is a psychological horror game I've played starring Kinito, a sentient desktop assistant based on BonziBUDDY who wants to be friends forever. I won't get specific, but he's very clueless about how the world works and how friendship works, and I want to try to replicate that. I eventually decided that my main character would sort of play on this idea that they're lonely and want someone to stay with them. I had the idea that he created the player just to have someone to talk to but, while trying, also made a bunch of 'rejects' (who are the monsters the player fights). Like AM (which stands for Allied Mastercomputer), I decided to name this character A-OS (pronounced like 'chaos,' it doesn't stand for anything intentional, I just like the sound of it.)

In terms of story structure, I decided to use the 3-act-structure. I'll treat Act 1 and Act 3 as normal, but I'll use Act 2 a bit differently since this is a game. Act 2 will be the game's general gameplay. That will take up most of the story. Act 1 will be used as an introduction to the game world and as a tutorial for the player gameplay-wise. Act 3 will simply be an ending to the game.

Diagram of the 3-act-structure (diagram from [Wikipedia](#))



Art Style

My game's art and visuals should support the story since the visuals will be one of the main ways my story will be communicated to the player.

Originally, I was thinking of a sort of 'technological' art style, which is mostly dark with neon colours for contrast (see below for examples). I eventually decided against this. I figured visibility would be difficult since most of the screen would be dark and it may be hard to differentiate and recognise game elements like enemies. Instead, I decided to go for a retro-cartoony look, similar to Lethal Company or The Upturned (see below for examples). The visuals in these games are stylised and unique, which I sort of want to replicate. This still fits with a sort of retro art style I was originally thinking of, and the player could recognise game elements more easily than with the other technological art style I was thinking of. Since my game is 3D, I'll make my 3D models low-poly, adding to the cartoony look.

The technological art style I was considering.

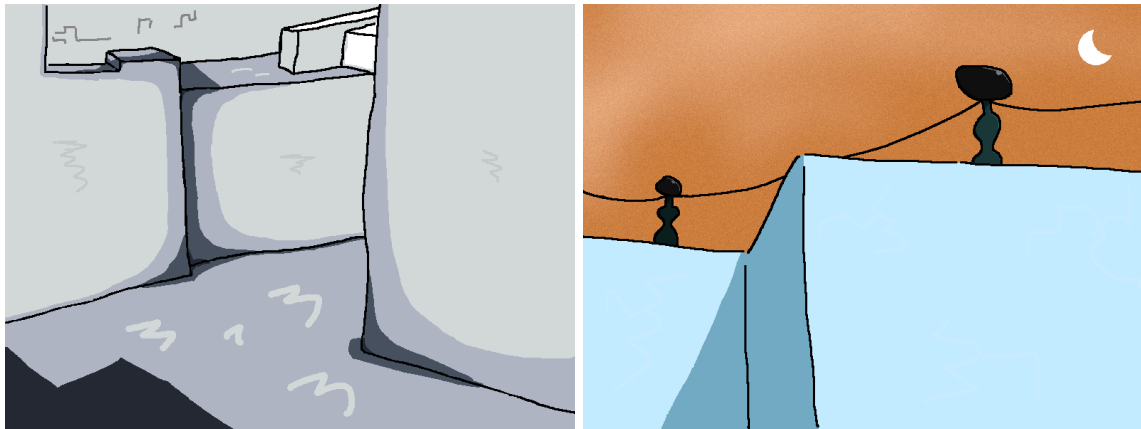


That retro/cartoony art style that I decided on. Images from The Upturned (left) and Lethal Company (right)
These images are stylistically pixelated, but you can't tell when the images are small like this.



Before doing anything concrete, I decided to experiment and ensure everything looked right. To do this, I made concept art. For how the world looks, I decided to do some sort of weird abstract collection of boxes. I wasn't inspired by anything.

Concept art of the game's world

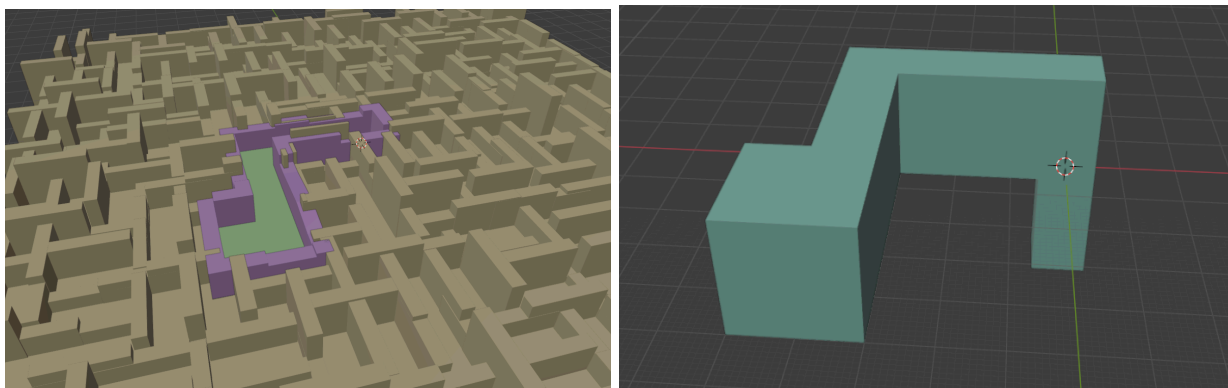


These images aren't great. I'm not good at art, but they communicate my ideas just fine.

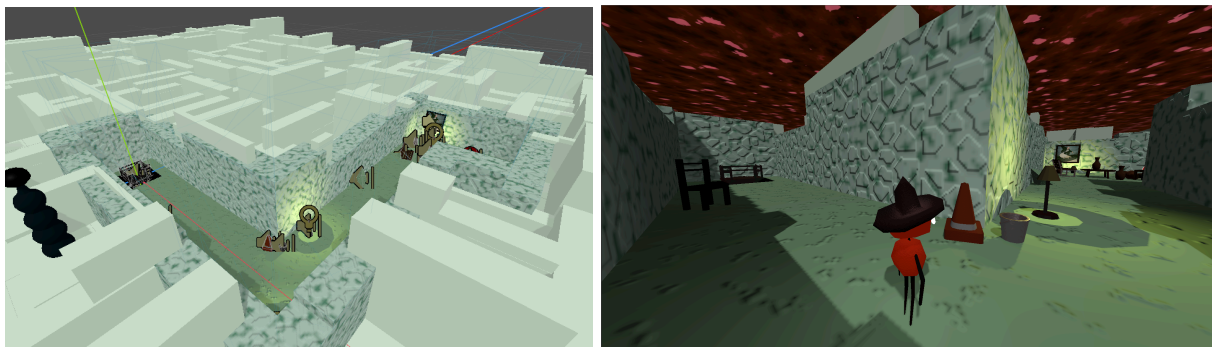
View full images: [concept_art-environment1.png](#) and [concept_art-environment2.png](#)

I based each level on these images. I took the current levels I have and made them look like my concept art. Within Blender, each level consists of an invisible collision mesh (which was the level's block out), a visible level mesh that looks like the concept art above, and some terrain outside the game level that only exists for visual purposes (so the player doesn't see void when they look outside the map).

The visible map (left) and block out used for collision mesh (right)



View in Godot, with textures and assets (left), and in-game view, with final textures, assets and colours (right)



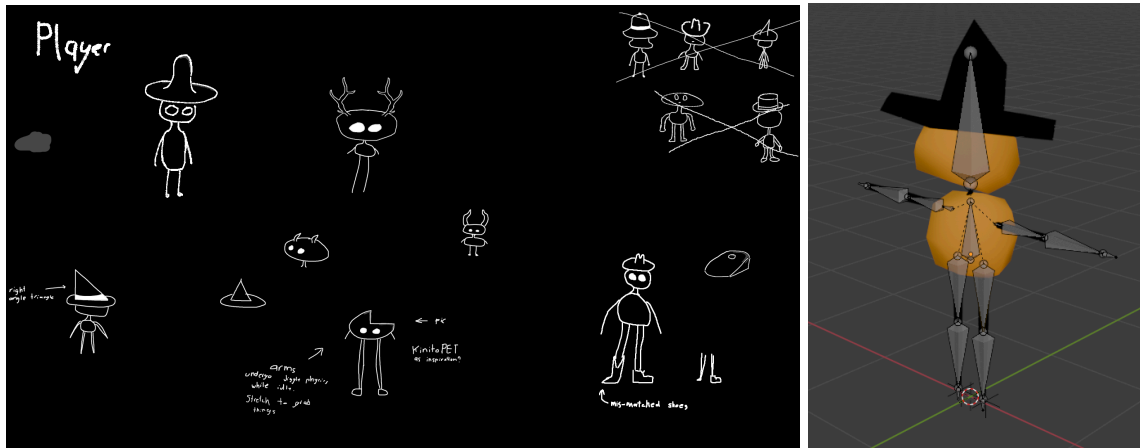
View full images: [development_level-studio.png](#) and [development_level-ingame.png](#)



In terms of colour scheme, I decided to choose a complementary colour scheme. A complementary colour scheme consists of two colours used on opposite sides of the colour wheel. These two colours contrast against each other. I chose a red-green colour scheme. I was originally thinking of an orange-blue colour scheme, which can be seen in the concept art above. I decided to change it since I figured the red-green colour scheme fit the theme of the story better.

For my player design, I wanted it to look simple but distinct. So, I decided on a simple shape with some kind of head accessory. I was thinking of either some kind of hat or some animalistic accessory like horns or antlers. I played around with some ideas on MS Paint with the concept art. I figured that the hat looked better.

Concept art of the player-characters design (left) and Final model with visible skeleton (right)



View full concept art: [concept_art-player.png](#)

With an idea of what I wanted the player character to look like, I modelled it in Blender. To get a shape, I used a skin and subdivision surface modifier within Blender. This setup creates geometry around single vertices. I can resize the created geometry to quickly make a general shape with only a few vertices. After that, I added a decimate modifier, which removes geometry from the model. That creates a more jagged and low-poly look. With the model created, I made a simple skeleton that will be used for animating. The skeleton consists of 11 bones total; 3 for the torso and head, and 2 for each arm and leg. This was then animated for the game's animations. The player's animations can be found [here](#). The model will then be textured within Godot using its shader editor.

Now, for the stylisation in-game. This is done with post-processing shaders made with Godot's shading language. For the stylisation, there are 3 parts: outlines, colour quantisation, and pixelation.

The outlines are made with a [Sobel](#) filter. This is an algorithm that runs on an image, which is commonly used in image processing for edge detection. I applied this filter to the game's screen, which produces results as seen below. However, this produces outlines with neon colours. So, I made every colour that wasn't black white. After that, the final image is overlaid onto the original frame. This is then run and applied to every frame drawn.

```
Unset
sobel\_outlines.gdshader
shader_type canvas_item;
render_mode unshaded;

uniform sampler2D SCREEN_TEXTURE: hint_screen_texture, repeat_disable, filter_nearest;

const float outline_alpha = 0.5;
const float outline_threshold = 0.25;

vec3 convolution(sampler2D tex, vec2 uv, vec2 pixel_size) {
    vec3 conv = vec3(0.0);
    for (int row = 0; row < 1; row++) {
        for (int col = 0; col < 1; col++) {
            conv += texture(tex, uv + vec2(float(col - 1)) * pixel_size).rgb;
        }
    }
    return conv;
}

void fragment() {
    vec3 pixels[9];
    // This forms a Matrix
    // [0, 1, 2]
    // [3, 4, 5]
    // [6, 7, 8]

    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++){
            vec2 uv = SCREEN_UV + vec2(float(col - 1), float(row - 1)) *
SCREEN_PIXEL_SIZE;
            pixels[row * 3 + col] = convolution(SCREEN_TEXTURE, uv,
SCREEN_PIXEL_SIZE);
        }
    }

    vec3 gx = (
```

```

        pixels[0] * -1.0 + pixels[3] * -2.0 + pixels[6] * -1.0
        + pixels[2] * 1.0 + pixels[5] * 2.0 + pixels[8] * 1.0
    );
    vec3 gy = (
        pixels[0] * -1.0 + pixels[1] * -2.0 + pixels[2] * -1.0
        + pixels[6] * 1.0 + pixels[7] * 2.0 + pixels[8] * 1.0
    );

    // Raw sobel colour
    vec3 sobel = sqrt(gx * gx + gy * gy);

    // Turning the outlines into a single colour
    float grey = (sobel.r + sobel.g + sobel.b) / 3.0;
    vec4 sobel_color = vec4(grey, grey, grey, 1.0);
    // Turning black values into transparent values
    vec4 outlines = sobel_color;
    if (outlines.x <= outline_threshold) {
        outlines.a = 0.0;
    }

    // Turn white values into dark outlines
    if (outlines.a > 0.0) {
        outlines.r = 0.0;
        outlines.g = 0.0;
        outlines.b = 0.0;
        outlines.a = outline_alpha;
    }
    COLOR = outlines;
}

```

Sobel filter with unfiltered neon colours (left) and black and white version (right)



View full images: [development_sobel.png](#) , [development_sobel-greyscale.png](#)

Output overlaid onto the original frame, creating the outlines.



View full image: [development_sobel-outlines.png](#)

The colour quantisation is very simplistic. Colour quantisation is a process that reduces the amount of colours in an image. It's usually done in image compression, but I'm intentionally doing it to create a stylised look. With every pixel, it takes the RGB components and rounds each component to the nearest multiple of 0.15. With colour in Godot, each component is a value between 0 and 1, instead of 0 to 255.

```
Unset
pixelisation.gdshader
shader_type canvas_item;

uniform sampler2D SCREEN_TEXTURE: hint_screen_texture, repeat_disable, filter_nearest;
const float scale_factor = 15.0;

void fragment() {
    vec4 pic = texture(SCREEN_TEXTURE, UV);

    pic.r = round(pic.r * scale_factor) / scale_factor;
    pic.g = round(pic.g * scale_factor) / scale_factor;
    pic.b = round(pic.b * scale_factor) / scale_factor;

    COLOR = pic;
}
```

Result from quantisation



View full image: [development_quantisation.png](#)

For the pixelation in-game, I was originally thinking of literally rendering the game at a lower resolution. However, that would also affect the quality of HUD and UI elements, making any text hard to read. Instead, I applied another shader effect to the rendered frame. This shader makes surrounding pixels the same colour to create a pixelated effect without actually rendering at a low resolution.

Unset

[pixellation.qdshader](#)

```
shader_type canvas_item;  
render_mode unshaded;
```

```
uniform sampler2D SCREEN_TEXTURE: hint_screen_texture, repeat_disable, filter_nearest;  
const int pixel_size = 4;
```

```
void fragment() {  
    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;  
  
    COLOR = texture(SCREEN_TEXTURE, vec2(x, y) / vec2(1.0 / SCREEN_PIXEL_SIZE.x, 1.0 /  
SCREEN_PIXEL_SIZE.y));  
}
```


Result from pixelation. It may be hard to see due to the image size. I advise viewing the full image.



View full image: [development_pixelisation.png](#)

Testing and Feedback

Again, I submitted my game to the Techquity NZ High School Game Jam. This time, I made my feedback form take the emails of responders. This was an effort to try to prevent people from responding poorly like last time. This did not work– they did it anyway.

Form Responses

Question	Responder's Emails (censored)	Rate the game on a scale of 1 to 10	Explain reason for answer above	Report bugs here	General Comments / Feedback
Answer	-	10	very well made with few bugs. Nice story, sounds and graphical style. Generally very good	a few things clipped through walls. I also had an egg duplicate when i came down the lift	i liked a lot. One of the better ones (:
	-	10	i like the funny man move		i believe the game could be improved if there was a speedrun mode
	-	10	wow so good so cool. the art style is so cool lethal company-esque. The story was so cool and interesting and wow. you should make it so the guy who created you is evil >:)	ladder disappeared through the wall on first level	can u make a drop option aswell for the objects so i dont have to kobe these ladders

However, this time the feedback isn't fully essential. There was no gameplay added in this sprint that'd require iteration. This sprint was just art, and these responses suggest that they liked it a lot. They also said physics bugs were a massive issue, mainly objects clipping through walls.

Reflection

This sprint took much longer than I was expecting it to. I originally thought this should take one month, but ended up taking two.

I realise that I could have ended this sprint much sooner if I had planned properly. I'll admit, I spent some of my time doing things that weren't on my plan- like over-engineering a save system or settings system. I should have only done that if I had extra time.

I realise that I've barely looked at my project management tool (Outlook To-do) at all. I need to refer to that more. I also now notice that Outlook To-do is **not** designed for project management. It's very black-and-white, where things are either marked as 'to-do' or 'done.' It also barely allows me to add additional details to any task. Due to this, I will use a different tool for project management in Sprint 3.

For Sprint 3, I will use Trello because I've already used it before for previous projects for my digital technology class- I already know how it works. Unlike Outlook, Trello is designed for project management.

Sprint 3

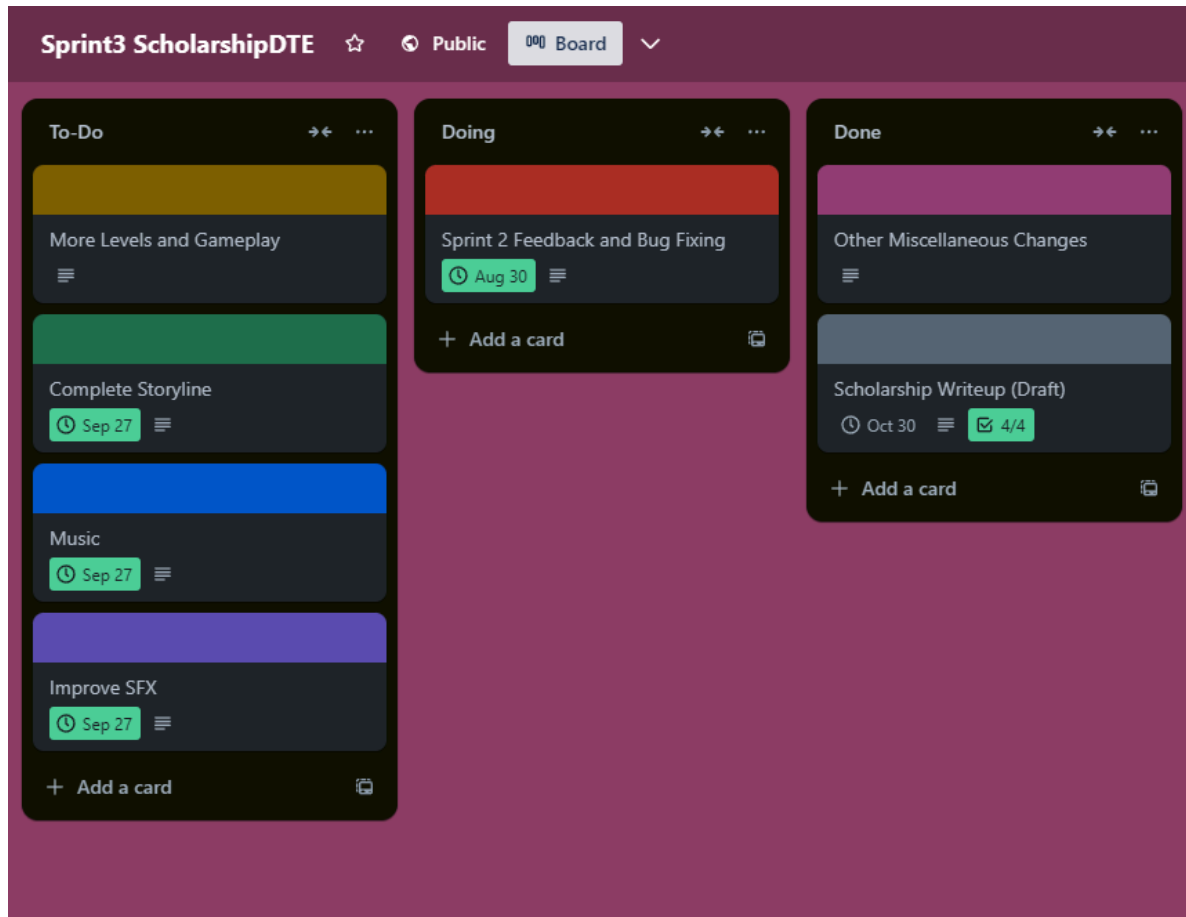
Goals

Sprint 3 will be my final sprint, so I need to aim to have my game finished. To do this, I will:

- Implement fixes and changes from feedback
- Add more game content
- Develop a saving system
- Update sound effects and create game music
- Create a game-ending

This sprint is generally brief. There are not many new mechanics or changes that need documenting, unlike the previous two.

Image of my Trello board



Development Towards Goals

Feedback

From my game's feedback, there were a lot of physics bugs reported. I did some investigating and found two main problems: thin/small objects could phase through walls, and objects would glitch out when the player stands on the edge of one.

To fix the objects phasing through the walls, I know Godot is typically bad when it comes to collisions with small objects. So, for smaller objects, I manually increased the size of their collision shape. This helped, but wasn't fixed fully. To ensure that this never happens again, I doubled the physics ticks that occur per second, which doubles the number of physics calculations per second. This may result in poorer performance, especially on poor hardware. I'll monitor game performance and listen to feedback to see if I need to revert this.

The second issue is much more difficult to fix. I've seen this throughout development but never attempted to fix it because it hasn't been important for

gameplay reasons. That changed this sprint, as a level added in this sprint requires the player to platform on objects, which may [get flung out from under the player's feet](#) due to this problem. To try to fix this, I increased the safe margin on the player- which is an extra margin for collision recovery in case of physics issues. I also set the contact speculative distance to 0 within the physics engine's settings- which increases the accuracy of the physics engine at the cost of performance. These changes seem to have done the trick, but I'll continue to monitor in case these physics issues return.

Game Content

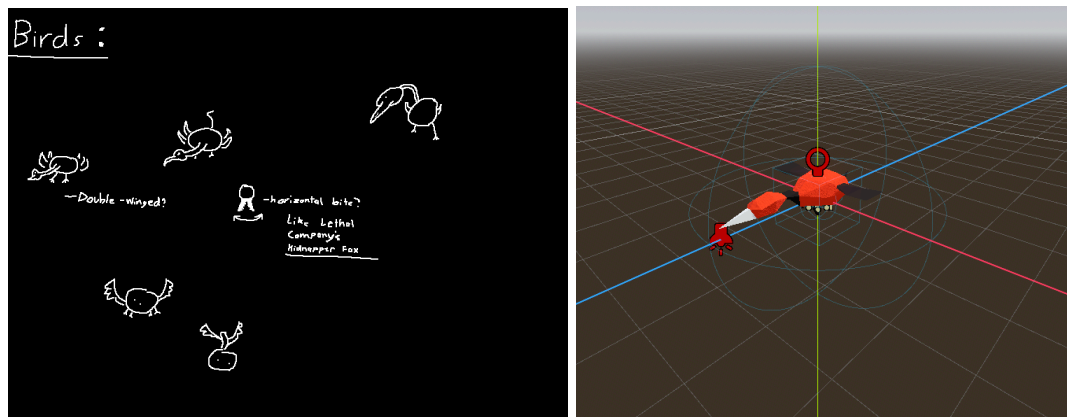
Right now, my demonstration from Sprint 2 has 4 levels, which isn't enough. Sprint 1 had 8 levels that I could reuse. But more levels won't be enough to make a good game. I want to add one more enemy to my game to try to increase the difficulty.

I wanted to make an airborne enemy that is hard to hit and hard to dodge. Thus, the bird was created.

I started with an idea of how I wanted it to behave. I wanted it to fly quite relaxed towards the player. When they get close enough, their flying steadies and they move quickly straight towards the player to sort of 'lunge' at the player. Then, when they touch the player, they deal damage to the player and quickly retreat away to rinse and repeat.

I made a working prototype to ensure that the bird works as expected and that there isn't any additional behaviour that I need to work on. Now, they needed art- some kind of visual design. At this stage, I figured that some players might find them annoying, so I decided to play with that idea and decided to make them look like geese (which can be seen in my concept art below).

Concept art of the bird's visual design (left) and editor 3D view of the final enemy (right)

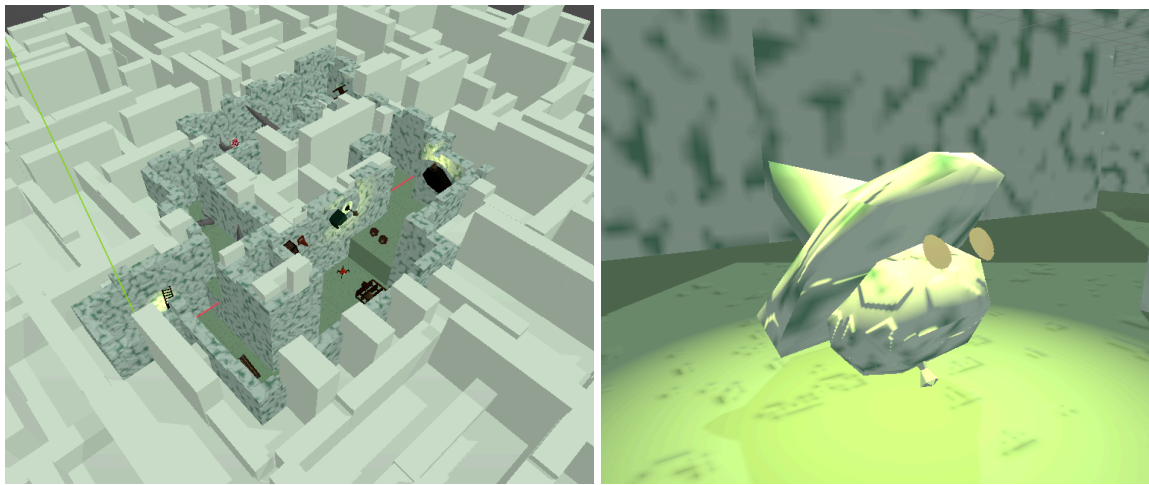


View full concept art: [concept_art-birds.png](#)

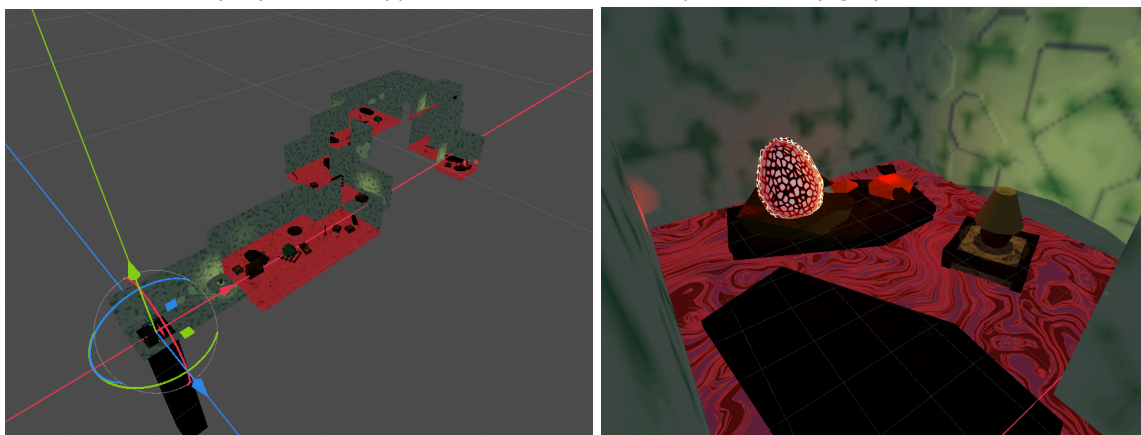
From this, I needed levels for the new enemy. I decided to include it in the last 3 levels: 1 to softly introduce the bird and the other 2 to show its regular gameplay. From the original 4 I had in Sprint 2, I added 3 more levels, along with the 3 for the bird, to add to a total of 10 levels.

With level 5, I intentionally gave the player only one object and they must carefully hang onto it. They'll need it to kill enemies and jump a ledge. Level 6 is about a different enemy that was Sprint 1. I reworked the 'bomber' enemy to instead be a physics object that rolls towards the player and starts a fuse to explode when they get near. I wanted an enemy that the player could pick up and throw around. This would then be used in the level to blast objects out of the way and clear a path. It's also used to blast the objective item to a place where the player can grab it. I then have the 7th, 8th and 9th levels for the bird enemy, and the 10th level will be the game's ending.

Overview of level 5 (left) and image of the single object provided on level 5 (right)



Overview of level 7 (left) and first appearance of the bird enemy on level 7 (right)



Saving System

After adding the new levels, I realised that the game may not be played by players in one sitting. To allow this, I decided to add an autosave system that would save the player's progress.

Whenever the player reaches a new level, the game extends the script below and calls the `save_game()` function, with the new level path as a parameter. This function sets the export variable named `level` to the given level path. It then saves the script itself with the export variable included as `save.res` to the user folder.

When it comes to retrieving this data (only done in the main menu to return your progress), the game fetches the save file and retrieves the level path within it. That is done in the `load_game()` function, which returns the save file resource. The game then uses that to return your progress. In the case that the file cannot be found (or the data is damaged/corrupted), the game will create a new game and overwrite your save.

Python

```
# The entire save_level.gd script
extends Resource
class_name SaveGame
# To use this script anywhere, use :
# var save = SaveGame.new()

# This is the path where the save file is saved to
const save_progression_path = "user://save.res"

# must be a string of the file path to go to, default level initially
@export_group("Current Level")
@export var level : String # The contents of this variable is what gets saved

func save_game(data : String):
    # Saves the script itself, with the variables in it
    level = data

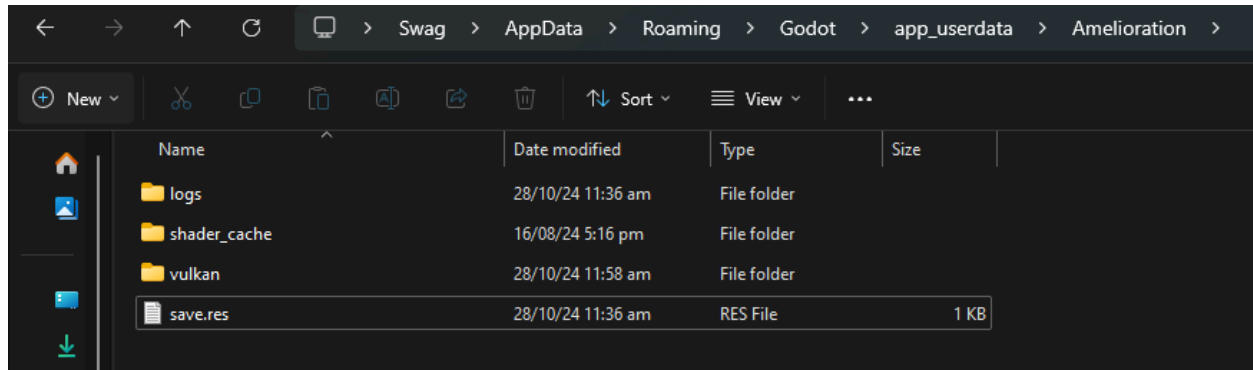
    ResourceSaver.save(self, save_progression_path)

func load_game():
    if ResourceLoader.exists(save_progression_path):
        var retrieved_save = ResourceLoader.load(save_progression_path, "")
        level = retrieved_save.level
        return retrieved_save # Returns the resource where everything is held

    else:
        return null # If the resource loader fails for some reason, returns null
```

The save file is saved to the user's folder. This is usually `C:\Users\<current user>\AppData\Roaming\Godot\app_userdata\Amelioration` on Windows computers. Godot uses a different location depending on the user's operating system, and may not be consistent between different operating systems (like Windows and MacOS).

The save file is located in the directory above, as seen in File Explorer.



Soundtrack

For the game, I ended up making 3 music tracks made in LMMS. Bear in mind that I do not know any music theory and this was my first time making music- ever. I'd rather learn how to make music instead of letting AI do it for me. I don't have too much to say about these tracks. There wasn't any research or inspiration behind them- I just wanted them to sound retro-esq to fit the game's art style. Frankly, I just did what I thought 'sounded right.' I didn't know what to name them, so I just named them in the order that they appeared in the game.

The first track ([1.wav](#)) is the first track I made. It's used for the introduction of A-OS and a slowed-down version is used for the title screen.

Both the second ([2.wav](#)) and third ([3.wav](#)) tracks play during gameplay in level 3 and level 5 respectively.

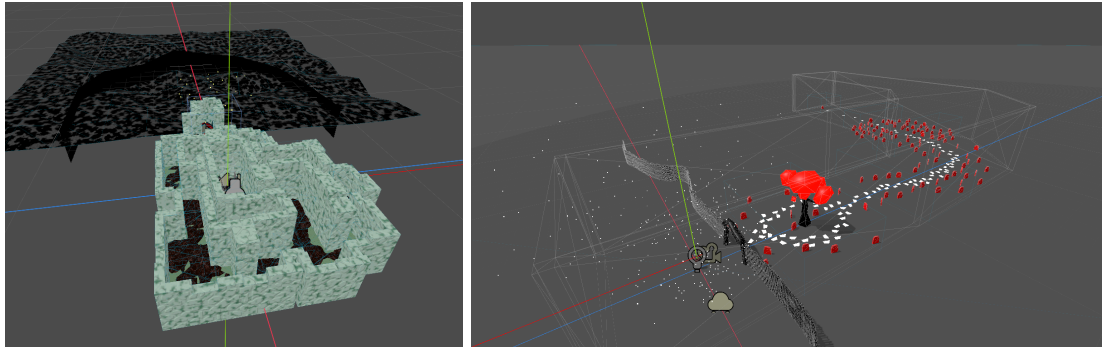
Game Ending

I have the very last level devoted to conveying an ending to the game. It then goes off to a very final 'heaven' scene.

The game's ending tries to convey a backstory to the game's world. With my game's ending, I tried to make it about how humans destroy each other with warfare, endangering everything else with it. I'm not too proud of this ending. I could have done this infinitely better.

I feel it was way too direct with the way it's communicated. It likely would have been more powerful if I implied this and let the player figure it out. There's also a major plot hole left that I only see in hindsight. The game never addresses how A-OS is communicating with you, where he is, or how he's even alive. I was going to imply that he was in some complex very far from the player's location, but couldn't find a way to communicate that.

Level 10 (left) and the ending scene (right)



Testing and Feedback

At the end of this sprint, I submitted it to the Techquity NZ High Schools game jam again. I asked one person to playtest it for me. I didn't observe them, I just left them to it and asked them to give me feedback when they were finished. They told me that the game was really good, but gave me information on minor issues. He said that sometimes the jumping mechanic wouldn't work, which would be easily fixed by adding coyote time to my player's jump mechanic. He also said that you can run past most enemies and ignore them.

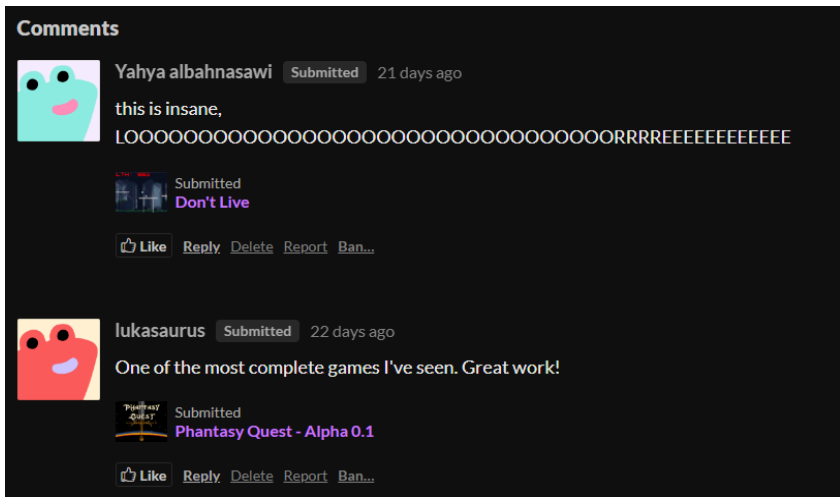
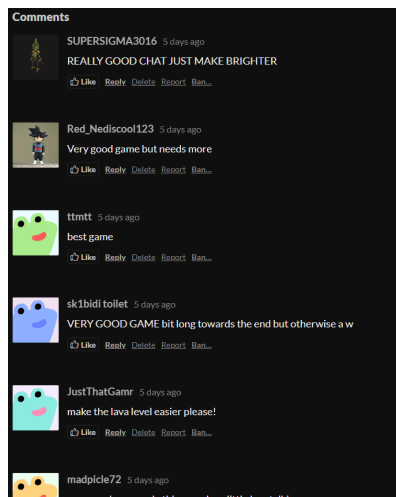
My game was received insanely well. Unlike the previous versions of this jam, the final product is rated by others who participated in the jam and a winner will be decided. My game was decided as the winner. Amelioration won 1st place in all categories, including overall. Techquity, the jam organisers, also declared winners by their judges, where my game won the Best Aesthetics category.

Game Jam ratings for my game ([link](#)), where my game came first in every category

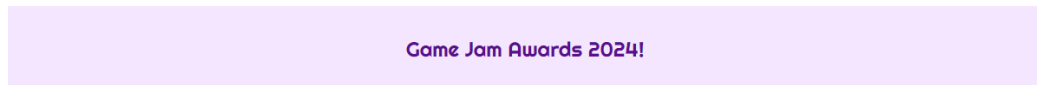
Results			
Criteria	Rank	Score*	Raw Score
Gameplay	#1	4.889	4.889
Use of Theme	#1	4.778	4.778
Juiciness	#1	4.889	4.889
Aesthetic	#1	4.889	4.889
Overall	#1	4.861	4.861

Ranked from **9 ratings**. Score is adjusted from raw score by the median number of ratings per game in the jam.

Comments on my game jam page



Game jam official judging, where my game won Best Aesthetics ([link](#))



Congratulations to all those involved in the game jam! There were 169 entries in the competition and the quality of the entries made the judges jobs really hard. Everyone deserves an award, but here are the top picks from the judges. Please play and enjoy!



swag0.itch.io/amelioration



yisen.itch.io/the-grey-room-final-release

Reflection

Development-wise, this sprint went very well. This felt like Sprint 2 but was insanely more productive. I managed to get everything I wanted done.

I felt that the switch to Trello had a change for the better, as it allowed me to better manage my time. I stuck to my plan. There wasn't a moment where I didn't know what I needed to do. I was also making a conscious effort to use it often and constantly refer to it, which helped me keep on task.

Evaluation:

Evaluation again Requirements

My game must be completable

The game progresses linearly. Every level in my game can be completed by the player, with varying levels of difficulty. After all 10 of the game's levels, there is an endpoint that serves as the end of the gameplay and the game's story.

My game must be easy to learn

My game overall uses 6 buttons for gameplay: WASD for basic movement, Space for jumping, and Left click for picking up and throwing objects. This simplistic control scheme keeps things simple for the player, to not overwhelm them.

My game approaches tutorials differently. I used a 'show, don't tell' method to teach new game mechanics to the player, without interrupting gameplay with a tutorial. The only exception to this is at the very beginning of the game. This is where basic movement, jumping, and the main throwing mechanics are directly told to the player. These are the most basic skills that need to be taught directly, as every other mechanic relies on these.

My game must be engaging for the player

I designed my game so that the game was engaging by constantly addressing MDA. I instil a sense of discovery and challenge in the player by incorporating them into the game's level design. Each level uses a rough level framework that allows the player to explore and fight (see [here](#)).

To ensure that my game was engaging, I ran playtests and received feedback from peers and strangers from other schools. This helped ensure that I was on the right track, or indicated if I needed to make changes.

~~My game must be a 3D top-down 3D game~~

This was changed throughout development. I changed to a 3D third-person camera scheme, instead of top-down. This was because the camera scheme made gameplay tedious. It made platforming annoying, made thrown objects difficult to aim, and made the player somewhat disconnected from the gameplay. This decision was a fundamental design flaw. It needed to be addressed ASAP. At this stage, I trialled a simple third-person camera to see how it compared. It performed significantly better– it fixed all the issues that the isometric camera had. From that point on, I used that third-person camera.

My game must allow the player to interact with physics objects

My main game mechanic that the player is constantly interacting with is the ability to pick up and throw any physics object in the level. Every other game mechanic interacts with this in some way. For example, enemies are defeated by throwing objects at them to deal damage. A special type of enemy that explodes can also be grabbed and thrown away, or redirected by the player for their advantage.

Evaluation against Implications

Usability

The game has an easy-to-use control scheme. 6 buttons are used in gameplay (WASD for basic movement, Space for jumping, and Left Click for picking up and throwing physics objects). These buttons are chosen because of their common usage in other games. Playtesting after this decision allowed me to verify my decision, to see if players agreed.

Functionality

My game works as intended. There are no bugs or errors that affect gameplay (that I know of). I playtested after sprints to ensure that the game works as intended. It also allowed me to see what players understand what the mechanics do and how they work.

Aesthetics

My game looks appealing to players. Before I modelled anything, I made concept art. That allowed me to ensure my game's visuals look good before setting anything in stone. I also took inspiration from Lethal Company and The Upturned for my visual aesthetic, which gave me a visual direction to go so I wasn't winging it (which would end horribly).

Reflection

What went well

I am very pleased with how my final product looks. I'm very happy with the aesthetics and visuals I chose. I feel that it was a good idea to do the concept art. The concept art allowed me to keep my game's art consistent, even though it's badly drawn.

I am also happy with how my gameplay turned out. My idea to centralise everything around 1 mechanic was probably the best move, given the allocated time of roughly 6 months.

What didn't go well

My storyline creation wasn't great. It was very time-consuming due to my lack of ideas. That caused time constraints due to the amount of time spent on it. Even then, I'm still not very happy with it. There are many mistakes with it. The method of storytelling is very direct, especially with the ending. It would have been a lot better if it was more indirect, and let the player imagine and interpret it on their own. There was also a major plot hole I failed to address. The player isn't told where A-OS is, what A-OS is, or how he's even alive. There's also no character development for A-OS. The story overall was a disaster.

Because I spent so much time figuring out the story, several time constraints arose. I had other ideas for mechanics I could implement if I had more time. For example, I was thinking I could allow the player to directly interact with held objects with a right click to light a fuse, toggle lights, etc. That would allow me to increase game complexity.

My thoughts

Although I'm not happy with some parts of my game, players found that my game is really good. I am overwhelmed with the amount of positive reception I got from players. The feedback on the Itch.io game jams was super positive.

Overall, I think my ideas were good, but some of my implementations were terrible. For instance, I had the ingenious idea of having everything to do with my player in one script– including player movement, hud, level changes, etc. The way my game's dialogue is handled is also terrible. I never addressed this problem in development because I didn't think it mattered– as long as it worked, it was fine. That ultimately made development harder because the code wasn't designed to be iterated and added upon in the future.

Future

I don't think I'll continue the development of Amelioration. I've learned a lot from this project– which I'll bring with me to future projects– but I won't continue this one. I have other game ideas that I want to pursue. Also, I've learned my lesson with some ways my mechanics are implemented. This experience will benefit me for future projects, regardless of whether they're games or other such software.

If I were to develop it further, I would add the idea I mentioned earlier about the player being able to directly interact with held objects. Earlier in development, I also thought of an object that teleports the player to where it lands after being thrown– like the Ender Pearls in Minecraft. I would implement that concept and several others that involve the central throwing mechanic. The entire game revolves around that one mechanic, and I like it that way.

Final Product

You can download and play the final version of Amelioration on its Itch.io page [here](#).

There is also a trailer that I made, which can be found here:

▶ Amelioration Gameplay Trailer . Alternatively, it can be watched as a file within Google Drive here: 📁 AmeliorationTrailer.mp4

You can find a full game playthrough here: 📁 full-playthrough.mp4 . This was recorded when recording footage for the trailer.

Amelioration's source code can be found publicly on its GitHub repository [here](#). Builds for older versions of Amelioration can also be found on GitHub in the [releases](#) section.

Bibliography

Research Sources:

Research

- <https://unity.com/>
- <https://www.unrealengine.com/en-US>
- <https://godotengine.org/>
- <https://upbge.org/#/>
- <https://creatinggames.press.plymouth.edu/chapter/what-makes-a-game-fun/>
- <https://users.cs.northwestern.edu/~hunicke/MDA.pdf>
- https://store.steampowered.com/tags/en/Indie?flavor=contenthub_toprated
- <https://www.patreon.com/zeekerss/posts>
- <https://arc.academy/how-long-does-it-take-to-make-a-video-game/>.
- <https://www.youtube.com/watch?v=mhv-C3WP9Cg>
- https://www.youtube.com/watch?v=-M76_buQY9I
- <https://partner.steamgames.com/steamdirect>
- <https://itch.io/developers>

Design

- <https://www.statista.com/statistics/1263585/top-video-game-genres-worldwide-by-age/>
- <https://dataprot.net/statistics/gamer-demographics/#:~:text=Video%20games%20are%20most%20popular,for%20male%20and%20female%20gamers>
- <https://www.nngroup.com/articles/ten-usability-heuristics/>
- <https://www.atlassian.com/agile/project-management/workflow>
- <https://www.atlassian.com/agile/project-management/waterfall-methodology>

Game Jam submissions

- <https://www.techquity.co.nz/>
- <https://itch.io/jam/techquity-sprint1-2024/rate/2854529>
- <https://itch.io/jam/techquity-sprint2-2024/rate/2854529>
- <https://itch.io/jam/techquity-final-2024/rate/2854529>