Inertia – A 3D Unity videogame

Jonathan Berkeley

N00181859

Supervisor: Catherine Noonan

Second Reader: Louise Glynn

Year 3 2020-21

DL836 BSc (Hons) in Creative Computing

Abstract

This application is a 3D videogame called Inertia, its purpose to provide entertainment. It has a focus on multiplayer, which has been developed open-source and from scratch to provide thorough support for users to make their own modifications and servers to play with friends.

The actual gameplay is focused around using momentum and movement mechanics to achieve a goal. There are different game modes in the game, such as PvP (Player Vs Player) where players can battle against each-other, PvE (Player Vs Environment) where players can battle computer-controlled bots, and finally race mode, where players race to get to a location the fastest to win. The game is designed to have a cartoonish playful appearance, similar to well-known and popular games such as Team Fortress 2.

Figure 1 - Title screen.

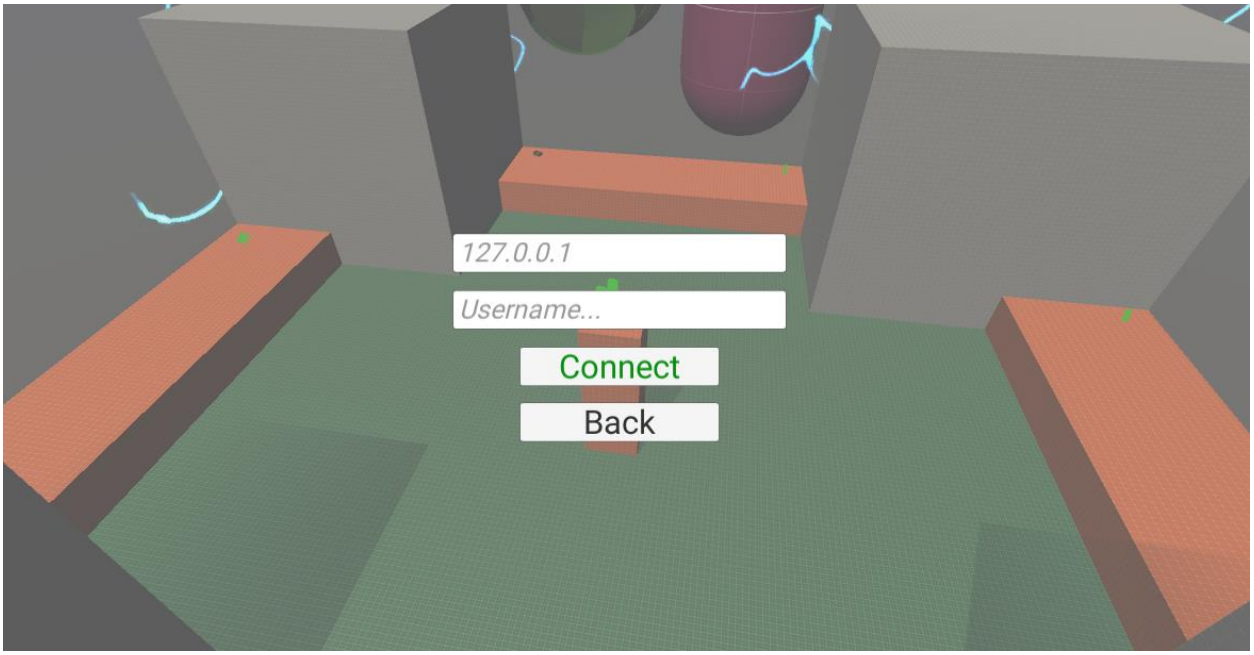


Figure 2 – Multiplayer connection screen.

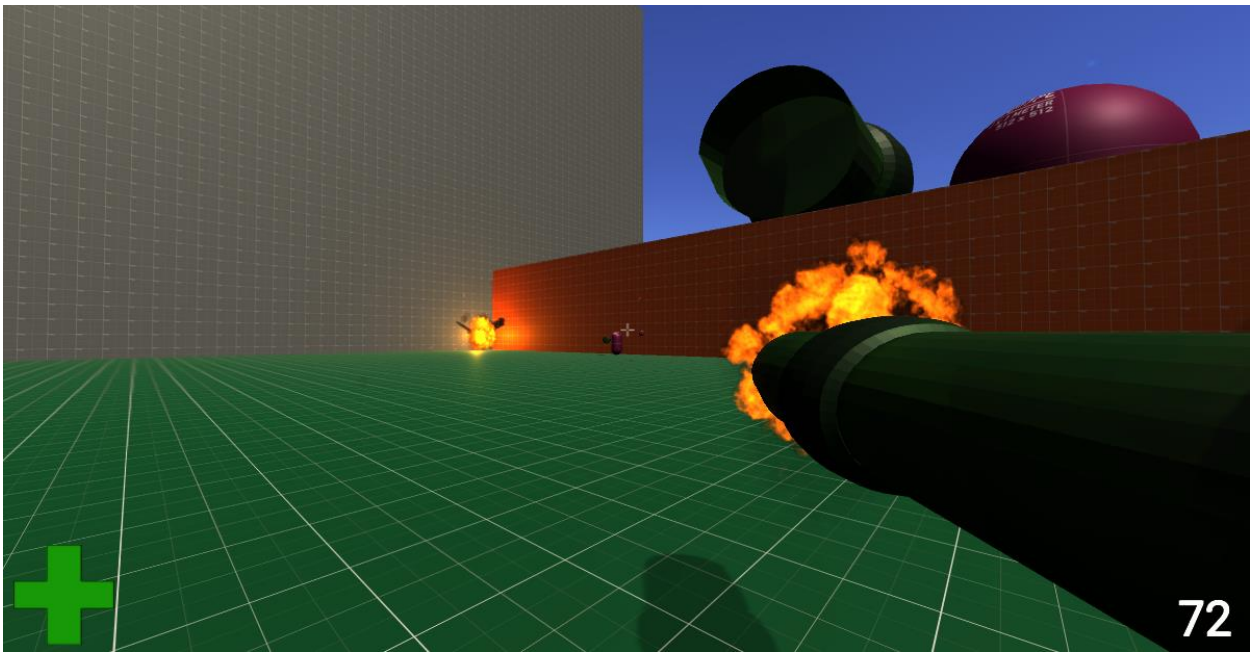


Figure 3 - Player Vs Bots mode (Shooting a rocket launcher at computer-controlled enemies)

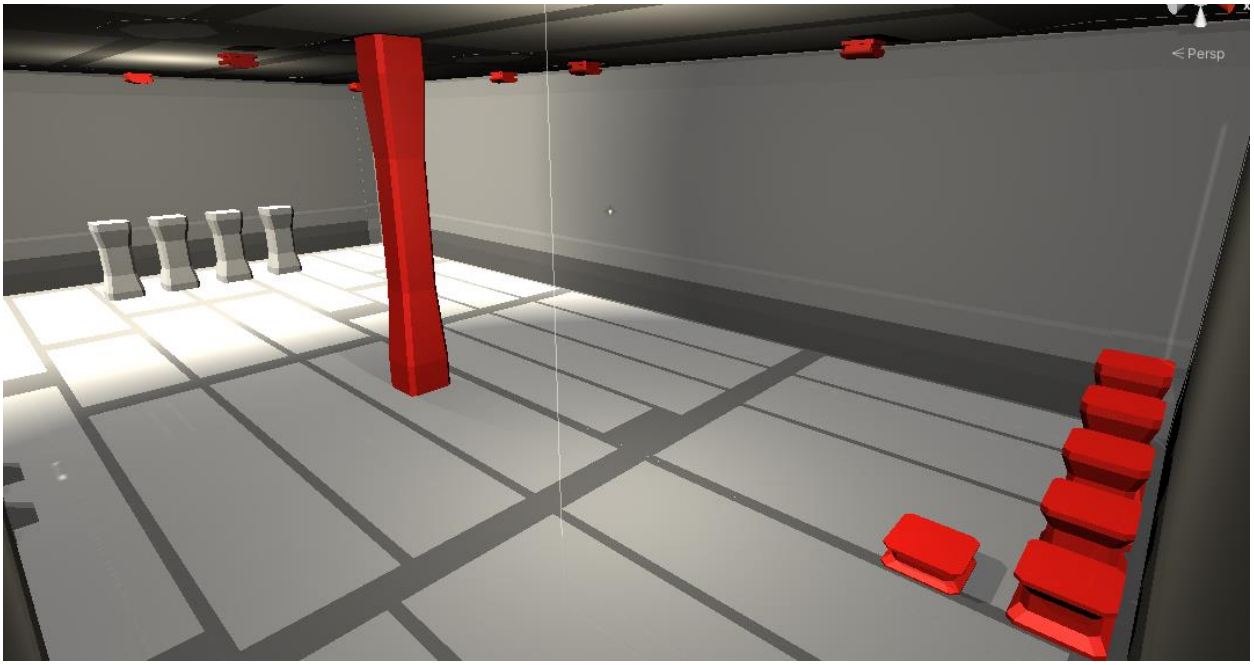


Figure 4 - Grapple hook level

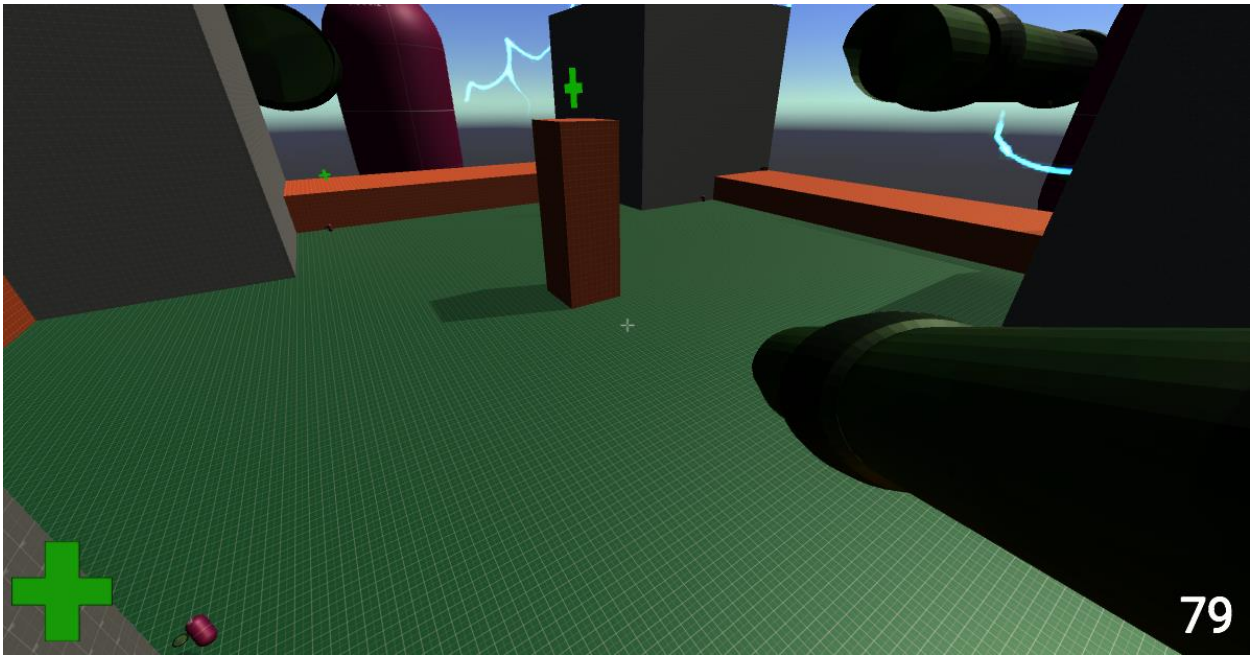


Figure 5 - Player Vs Player arena

Acknowledgements

First and foremost, I would like to give my sincerest thanks to Catherine Noonan for her continued support and advice on the project throughout, from its conception to its completion, it would not be possible to have brought the project to place it is today without that continuous support.

I also want to extend my warmest thanks to Louise Glynn, the networking information I learned from the Computer Networks module proved invaluable in the creation of the server software used at the heart of this project.

I would like to thank Joachim Pietsch for the information learned in the Game Development module which rapidly improved by ability to develop games in Unity, which was crucial for this project.

And finally, I would like to thank my partner in this project, Mark Hurley. Mark Hurley has been a great help, and I have enjoyed my game development experience alongside him in this project.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student: Jonathan Berkeley (N00181859)

Signed: Jonathan Berkeley

Table of Contents

2 Requirements

Introduction

The product being developed is a single player 3D movement-based physics game, where the player assumes control of an unnamed test subject who is armed only with a grappling gun. This unnamed test subject has the objective of escaping the laboratory they are trapped in by solving various puzzles using the grappling gun and their wits.

The game is very beginner friendly but can also be a challenge for those who have are interested in beating the game in the fastest time possible. The game requires the player to make their way from one end of the level to the other while jumping across and around platforms. It will be a relatively short game with the main focus being on the unique gameplay and level design.

This game will be made in Unity and will also be using Unity collaborate and Github to share the project between developers as they work on it. The game will be coded in C sharp (C#).

2.2 Requirements gathering

Similar applications

Portal 2: https://store.steampowered.com/app/620/Portal_2/

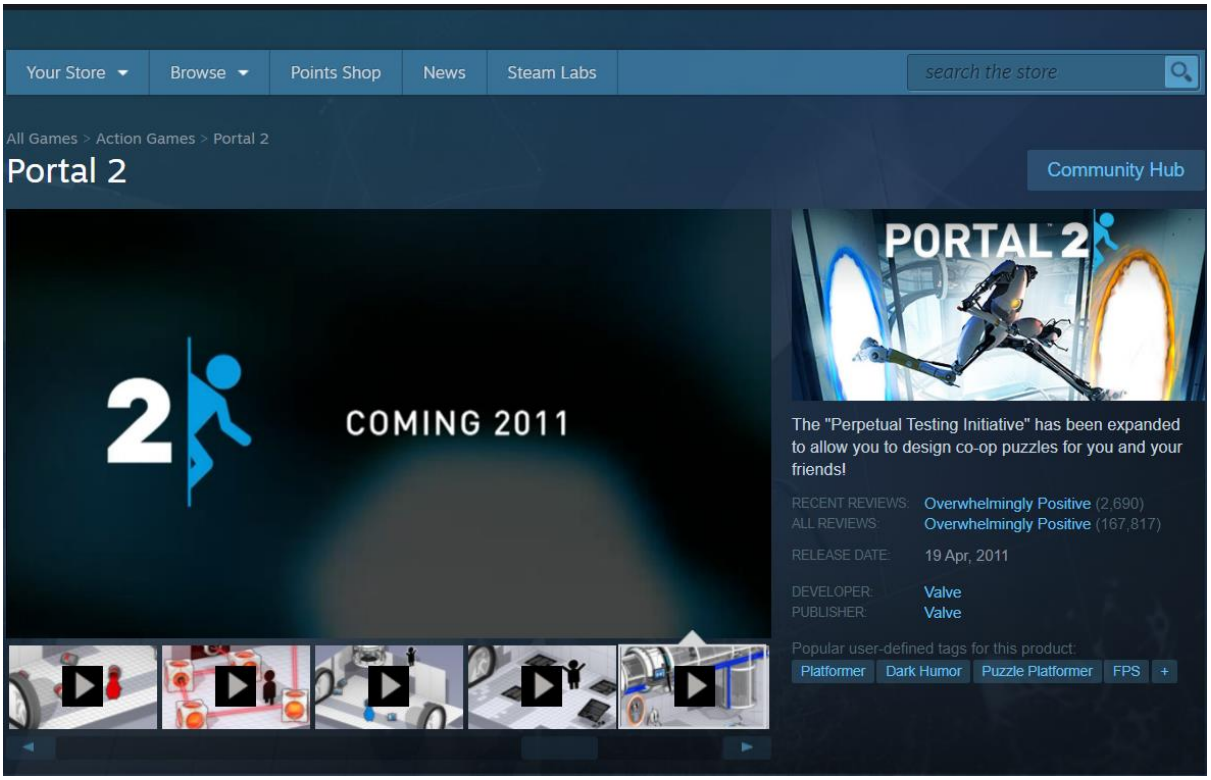


Figure 6 - Portal 2 home page on steam

Portal 2 is an Action Adventure, First Person, Puzzle Game that can be played in both single player and co – operative multiplayer. The single player game portion of the game takes place in a laboratory where the player takes control of a test subject trying to escape the lab while solving various puzzles along the way using a portal gun which used by placing an orange-coloured portal on one and a blue coloured portal on a different surface as seen in the example below.

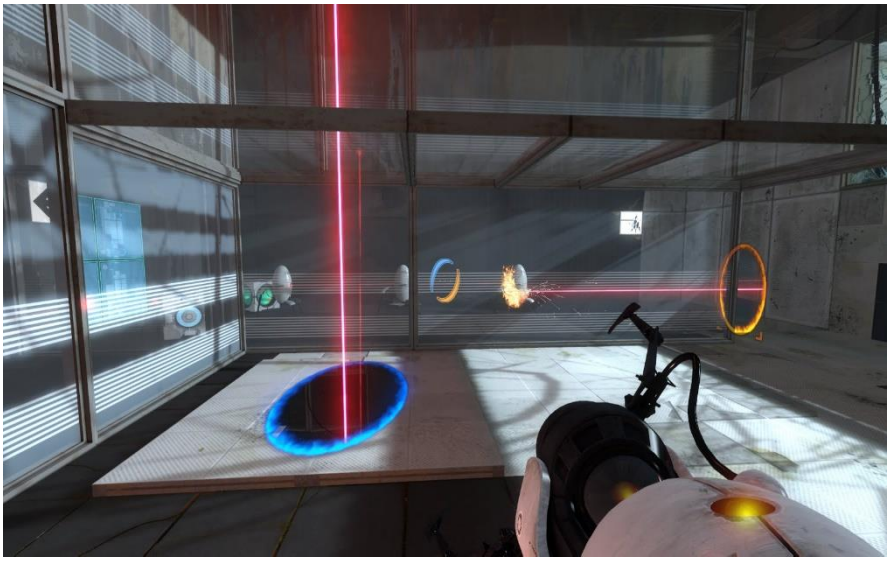


Figure 7 - example of the portal concept

As for the multiplayer side two players take control of two robots using portal mechanics as described above to solve various puzzles together with a narrative of freeing frozen test subjects in the lab. An example is shown below.

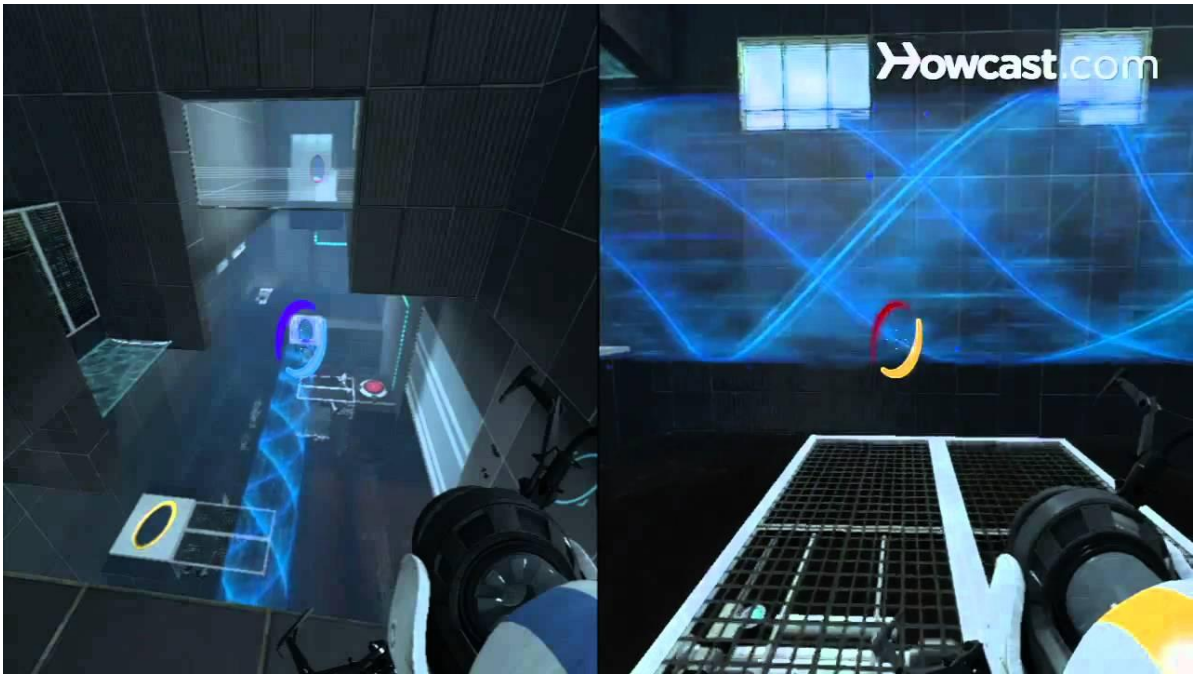


Figure 8 - example of co-op play

Advantages:

- Provides the user to experience the game alone or with a friend.
- Has a strong narrative in both single player and co-op.
- Has great characters and great voice acting performances.
- Has great and unique gameplay.

Figure 9 - Actors Ellen McLain and Stephen Merchant next to the characters they voice

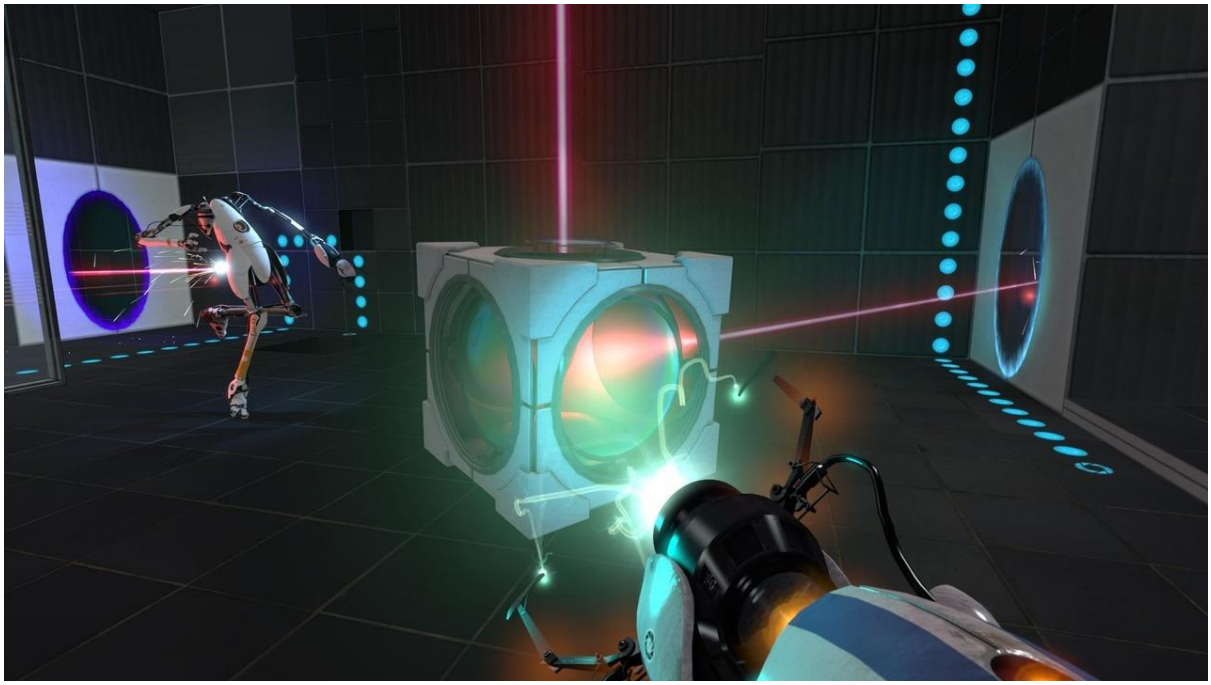


Figure 10 Example of unique mechanics

Disadvantages:

- Has aged graphics.
- Is considered too easy for some.
- It is a relatively short game.

Refunct: https://store.steampowered.com/app/406150/Refunct/

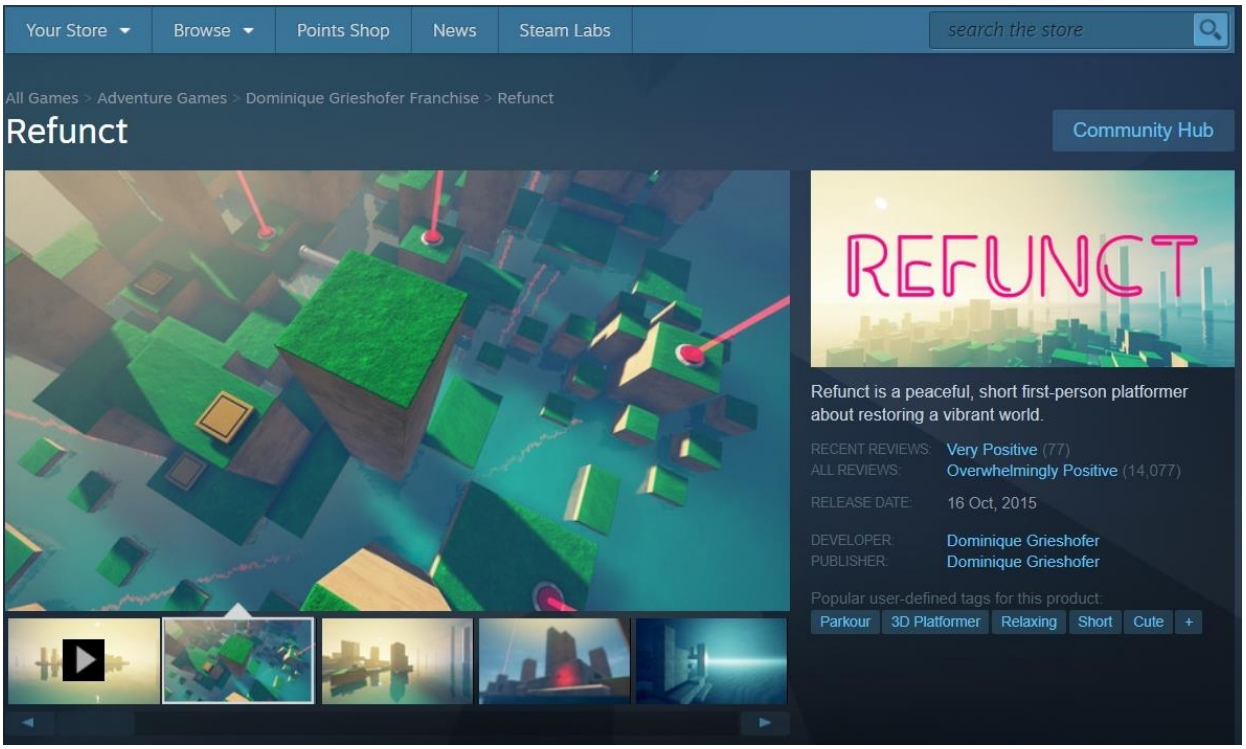


Figure 11 Refunct's home page on steam

Refunct is a small and calm Single Player, First Person, Puzzle, Platformer game where the player is tasked with getting from point A to B with no real narrative, but focuses more on the

visual aspects of the game with nice soothing colours and great ambience. An example is provided below.

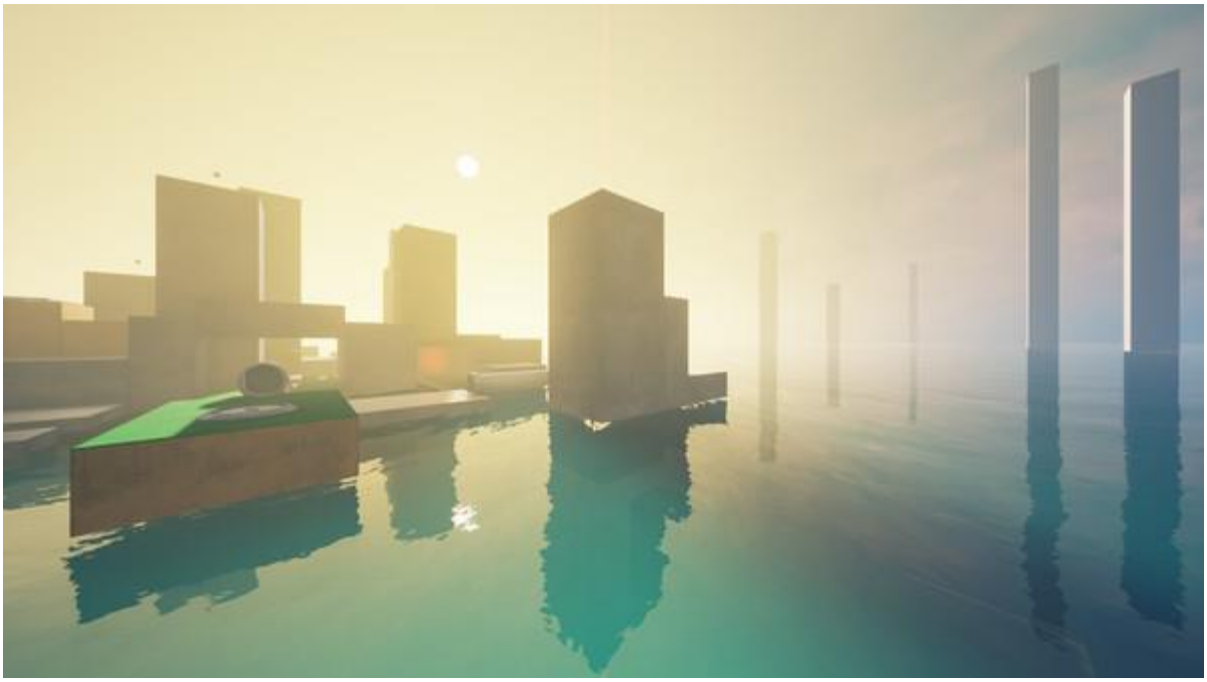


Figure 12 an example of Refunct's beautiful world

Advantages:

- Amazing visuals and music.
- Relaxing setting and tone.
- Easy to learn and hard to master puzzles.
- Great gameplay.

Figure 13 an example of Refunct's gameplay

Disadvantages:

- Might be considered too short for hardcore puzzle game fans.
- No story if the user is looking for a story driven experience.
- Not much replayability.

The Stanley Parable:
https://store.steampowered.com/app/221910/The_Stanley_Parable/

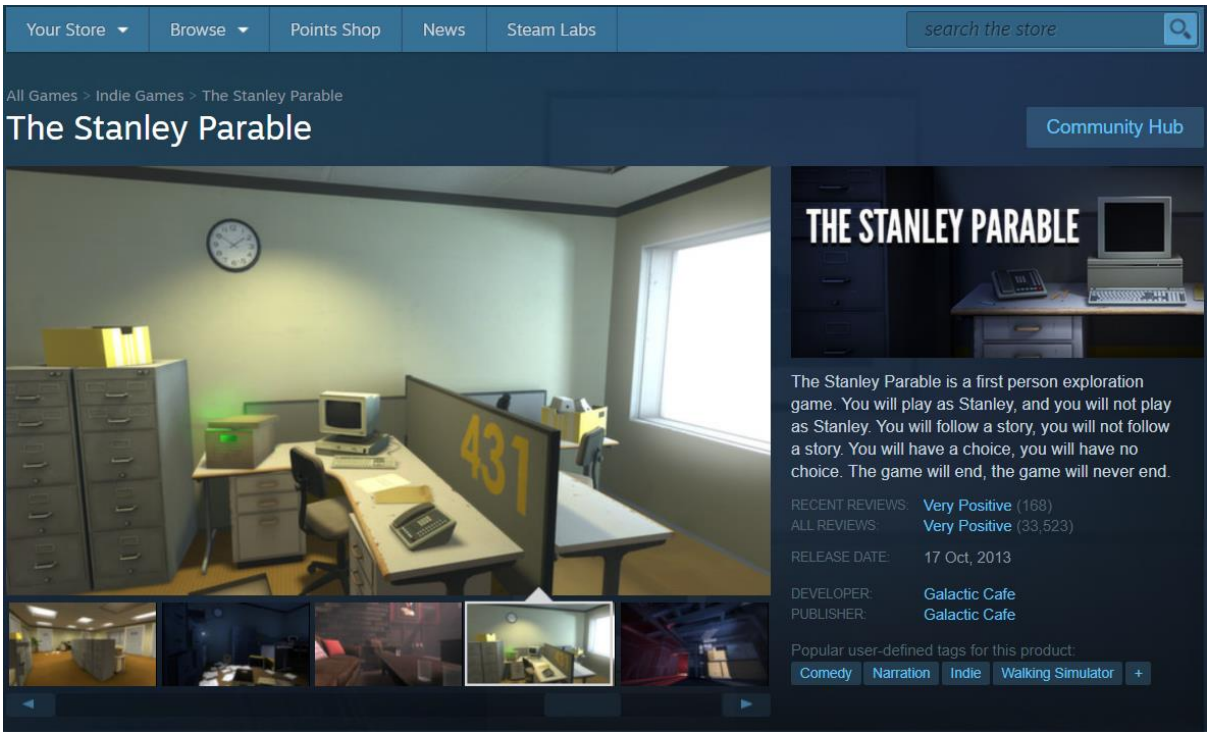


Figure 14 The Stanley Parable home page on Steam

The Stanley Parable is a Single Player, Story Driven, Comedy, First Person, Puzzle game where you assume the role of a man named Stanley whose goal is to do as the narrator says or do something else entirely, the choice is left up to the player.

Figure 15 an example of the kind of gameplay in this game

Advantages:

- A unique and clever game.
- Has some great and funny writing/dialogue.
- Has tonnes of replayability.
- Interesting story.
- Multiple pathways and endings.

Disadvantages:

- Limited Gameplay.
- Can be considered too easy.
- Not for the impatient.

Personas

Courtney Rose

"I always keep my head down and get my work done"

Age: 23

Work: Student
Family: Single
Location: Dublin
Character: Determined, focused, perfectionist

Personality

Introvert	Extrovert
Thinking	Feeling
Sensing	Intuition
Judging	Perceiving

OCD | Intelligent | Stubborn | Hard Worker

Goals

- Plans on getting a really great high paying job
- Wants to get the highest grades possible
- Hopes to overcome her social awkwerdness

Frustrations

- Dislikes waiting around
- Dislikes socializing
- Dislikes messy and disorganized things

Bio

Courtney Rose is a 23-year-old who enjoys helping old ladies across the road, meditation and running. She is intelligent and careful, but can also be very pessimistic and a bit impatient. She is an Irish Christian who defines herself as straight. She is currently in college. She is a vegetarian and is obsessed with Frozen. Physically, Courtney is in pretty good shape. She is very short with tanned skin, brown hair and brown eyes.
She lives in a middle class neighbourhood. After her father died when she was young, she was raised by her mother.
Courtney goes to IADT, where she is studying Applied Digital Business. She would not consider herself much of a gamer but does enjoy the occasional game for around an hour in total on a given week and usually on her mobile. She primarily plays Casual Indie games. She has never played any first person puzzle games, however after seeing the game refunct she thought the game looked visually appealing but was also confused as to what was happening in the game. upon being asked if she would play a game like that she said 'maybe'. She seemed to favour aesthetics, quantity of levels and information when asked about important features of a game.

Motivation

Incentive
Fear
Growth
Power
Social

Preferred Channels

Traditional Ads
Online & Social Media
Referral
Guerrilla Efforts & PR

Figure 16 Courtney Persona

Brendan O Doyle

"It'll be done when it's done"

Age: 21

Work: Student
Family: In a relationship
Location: Wicklow
Character: Lazy, Easy-Going and Smart

Personality

Introvert	Extrovert
Thinking	Feeling
Sensing	Intuition
Judging	Perceiving

Lazy | Smart | Easy-Going | Social

Goals

- Plans on becoming a game developer
- Wants to get decent grades
- Wants to be able to concentrate more in his college work if possible

Frustrations

- Dislikes complicated problems
- Dislikes being alone
- Dislikes drama and gossip

Bio

Brendan O Doyle is a 21-year-old who enjoys hanging out with friends and playing video games regularly. He can be smart when he chooses to be but generally prefers to just chill out and take things easy, loves being optimistic and making any situation fun anyway he can. He is an Irish Atheist who defines himself as straight. He is currently in college. He is not a picky eater and he is physically in decent shape. He is average height with white skin, brown hair and blue eyes. He lives in a middle class neighborhood near a shopping center. His parents are happily married and he has 1 brother and 1 sister.
Brendan goes to IADT, where he is studying Creative Computing. He would consider himself a massive gamer and is proud of that, he would easily spend 10+ hours in total on a given week playing games on his PC and occasionally on his PlayStation console. He primarily plays RPG's and FPS's. he has played his fair share of first person puzzle games and after seeing the game refunct he thought the game looked nice and calm and it appeared to have caught his attention, when asked if he would play a game like that he said 'sure I'd give it a go, it looks calming'. To him the important features of a game are gameplay, level design and physics.

Motivation

Incentive
Fear
Growth
Power
Social

Preferred Channels

Traditional Ads
Online & Social Media
Referral
Guerrilla Efforts & PR

Figure 17 Brendan Persona

Interviews

In the interviews conducted, participants were asked the same set of questions about gaming in general. From this, it was discovered what the participants liked and disliked about gaming. Some shared similar opinions while others had vastly different ones.

Everyone interviewed each said that they enjoy playing games and they would all consider themselves 'Gamers', which is the demographic the product is aimed towards so that the survey will get only the most relevant answers. When asked how often they would spend playing video games in each week the results ranged from 1 or 2 hours (only during very limited free time) to playing daily for upwards of 5-6 hours daily. This shows that the project developers need to be mindful of those who do not have enough free time to play games but also of those who spend all their free time playing games. The developers of this game need to strike a good middle ground for the game between hardcore gaming and casual gaming.

Everyone interviewed also seemed to primarily play on the same platform (PC) alongside some secondary console of preference. This is good for us as the game is being developed to be played on PC. When asked about favourite game genre, a variety of responses was received. The majority of responses listed RPGS as their favourite game genre. Everyone interviewed said that they have all played portal 2 which is one of the games listed above and that they thoroughly enjoyed it. This is great news for development as Inertia is heavily inspired by portal 2. When asked about the most important aspect of a game it appeared to be a 50/50 split among the people interviewed between story and gameplay which is quite interesting.

Survey

The survey created for this project consists of several questions aimed at finding out how much interest there would be for this type of game. It aims to find out how much awareness and interest there is for the game genre that the project is targeting, and to find out what people thought would be most important aspects/mechanics of a game in this genre. The survey was sent to all college students in IADT Creative Computing Year 3, and to some other students from other courses, all of whom are over 18 years old. The survey was not sent to anyone under 18 years old for ethical reasons. The survey received 12 responses since it was published on 16th Oct 2020 to 30th October 2020.

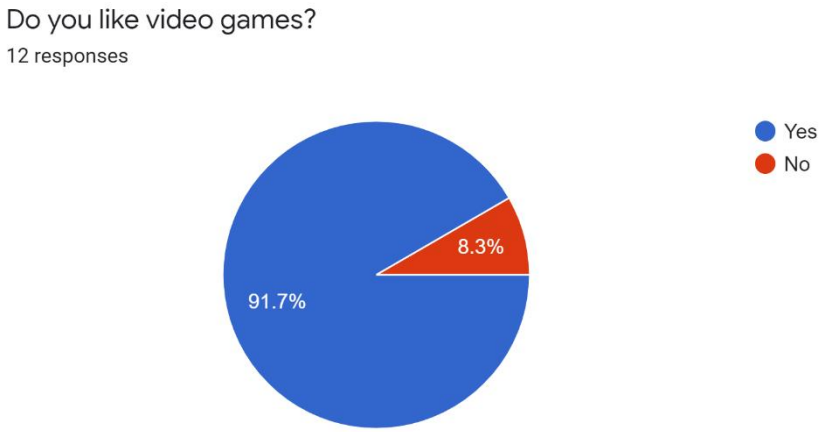


Figure 183 - Percentage of participants that like video games

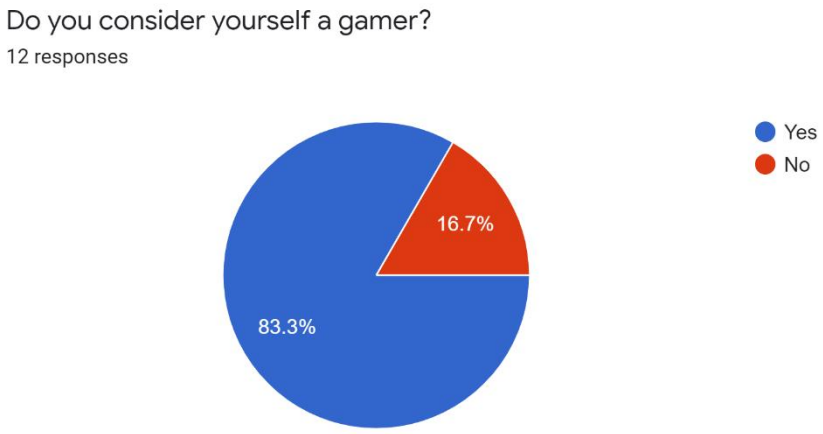


Figure 14 - Percentage of participants that consider themselves a gamer

These initial questions were asked to understand what background the participants were coming from, most of the participants were people that play video games.

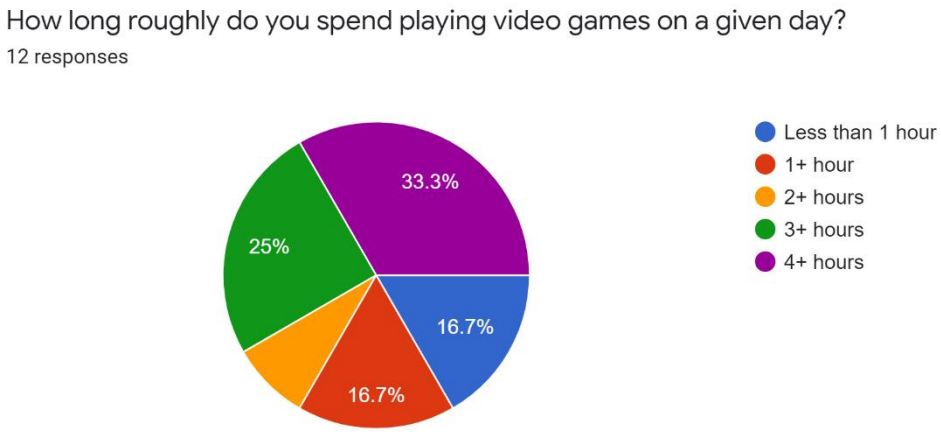


Figure 15 - Percentages of approximate play time on a given day

This question was asked to get approximate information about how much the survey participant plays games on a given day. The response indicated most participants spend more than 3 hours on a given day playing video games. This is good for the later survey questions as it shows most of the participants have a strong interest in video games.

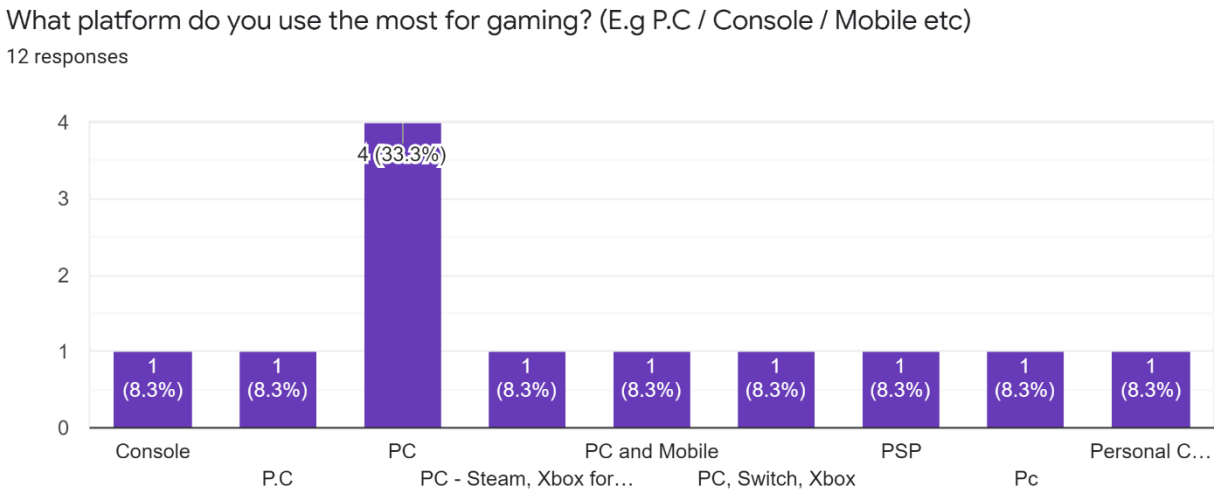


Figure 16 - Write-in information on participants choice for gaming platforms

This question in the survey allowed the participants to write their answer in as text. This is why there were multiple entries for the same thing (Pc, PC, P.C, Personal Computer). The survey allowed for write in answers as there are so many possible platforms to game on. Participants could write in answers as to not restrict the participants to choosing from a few. Despite this, 10 out of the 12 participants included P.C gaming in their answer, this is encouraging information as it is the platform that the game is targeting.

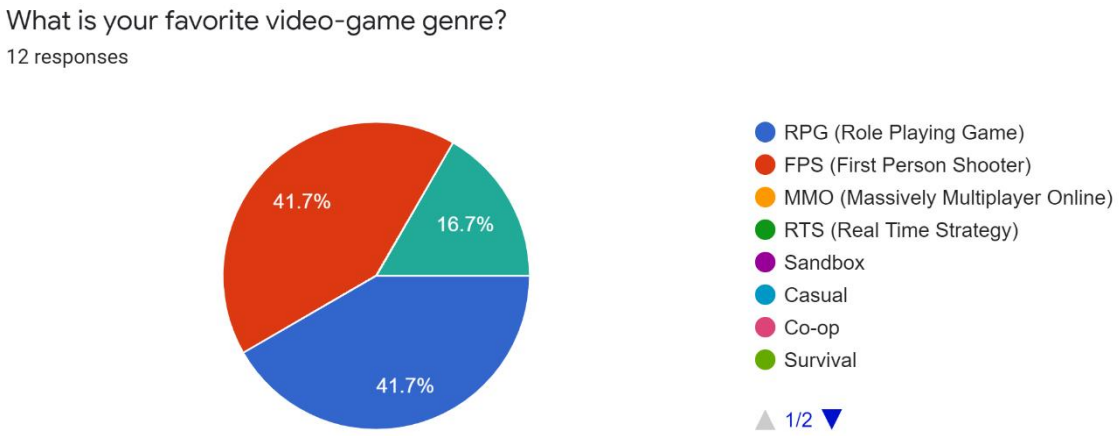


Figure 17 - Participant percentage answers for favourite video game genre (Page 1)

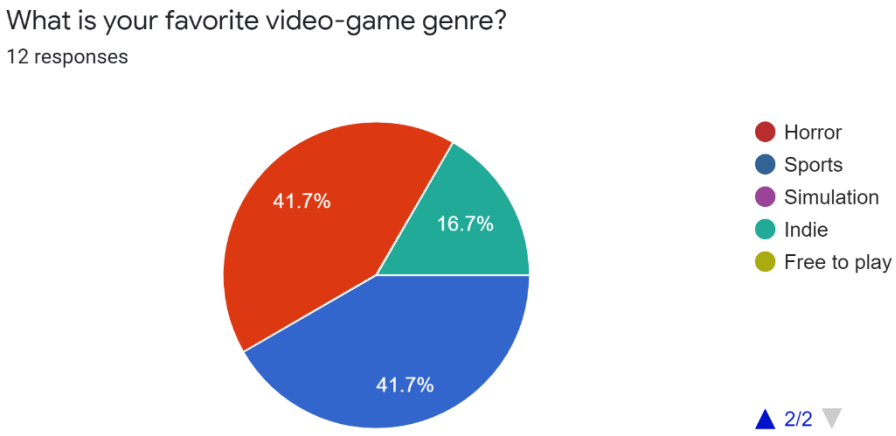


Figure 18 - Participant percentage answers for favourite video game genre (Page 2)

5 of the responses (41.7%) were for the FPS (First Person Shooter) genre as a favourite which is displayed in the pie chart as red. This genre is similar to Inertia's, although Inertia would not be considered a First-Person Shooter, it does follow a similar theme and gameplay feel. It has similar movement and view mechanics; Inertia is also from the first-person view. 5 other responses were for the RPG (Role Playing Game) genre, this does not have any particular relevance to Inertia, as it strays far away from the RPG genre. The 2 final responses listed Indie games as their favourite video game genre, this genre refers to games that are self-published, created by independent developers rather than professional game studios, this directly coincides with Inertia, which would fall into the Indie genre category as it has two developers and no publisher.

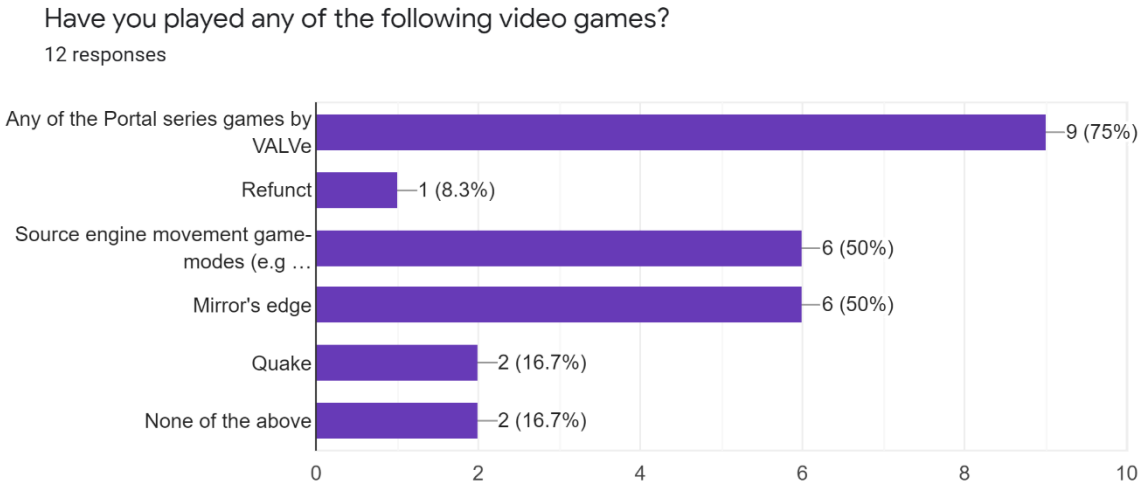


Figure 19 - Judging participant awareness to existing games similar to Inertia

This question was asked to find out what awareness and engagement there was for games that are similar to Inertia. 10 out of the 12 participants had played a similar game. All the games listed their cost money to play except for “Quake”, so this also indicates monetary interest in this type of game.

Participants were then given an optional section in the survey to answer, this contained footage of the video game ‘Refunct’. Refunct is an indie game developed mostly by a single developer, it is a simple movement game yet highly rated on the Steam platform for P.C. It was important to link this in the survey as it is an inspiration for Inertia’s development, 95% of the 14,077 reviews on steam for this game are positive as of 29th Oct 2020. The gameplay linked in the survey is located at this link: https://www.youtube.com/watch?v=l1ypINE-igE

Participants were asked to view a short portion of the clip and write in what they thought about it. The following images contain the text responses of the participants to the video clip. 11 out of 12 chose to respond to this part of the survey.

Briefly answer what you think of the gameplay in this game "Refunct". (Likes/dislikes)

11 responses

I like the simplicity and ambience

Nice environment with good lighting and how the blocks change adds complexity make it look fun. I don't think you die in this game? You just step on many red buttons as you want, which makes the game a bit boring afterwards. But having different levels will will make it more fun or if you touch the water and you die.

Boring, too repetitive

looks booty

I like the style it seems immersive, im not a big fan of the graphic style

Fun game to play while chilling out if you're into kz/bhop

Looks interesting

Like the speed and pacing, but I don't know what is going on

Gameplay seems ok, but that's not a game I'd play. I'd rather spend my time on a game with more content

The game looks interesting but probably not fun for long periods of time.

its interesting and unique aswel as being simple, i think that it could use some more satisfying animations in order to make it feel more grounded

Figure 20 - Participant write in responses for the gameplay of Refunct

Would you be interested in a skill based movement game similar to Refunct?

12 responses

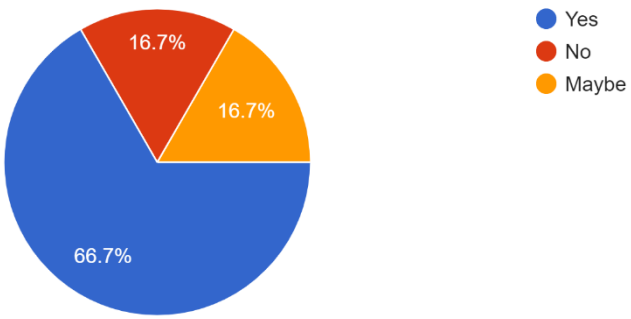


Figure 21 - Percentage of interest for a similar game to Refunct

This question was asked to determine the level of interest for a game like Refunct. 8 out of 12 participants (66.7%) indicated they would be interested, 2 indicated they might be interested, with the remaining 2 indicating they would not be interested. This is encouraging as it seems from this survey that there is some interest for a new game of this type.

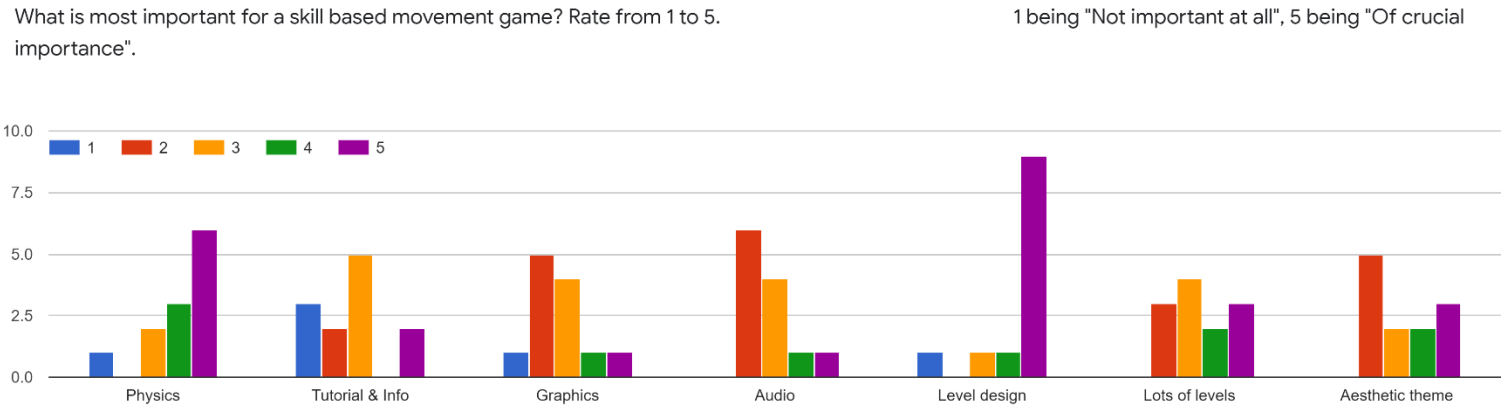


Figure 22 - Participant responses for importance of various aspects in Inertia's game genre

This question gave crucial feedback into what participants thought was important for a game of this nature. Participants were asked to rate 1 – 5 the importance of various aspects of game design, 1 signifying "Not important at all" and 5 signifying "Of crucial importance".

Summarizing the average results of the answers to this question, level design is the most crucial part of the game design for this genre (with 9 participants responding that it was "Of crucial importance"), followed by physics mechanics. Tutorial & informational mechanics, and aesthetic theme were of average importance to the participants. Audio and graphics were of low importance to most participants. There was dispute on the answer for the importance of the number of levels needed in the game, though most answered that it was important.

From the collected responses it can be concluded that the game will need a strong focus on level design and physics mechanics. There is interest in the type of game that aligns with Inertia's theme, and general interest in the genre it resides in. The majority of participants use P.C for gaming, so Inertia should continue to focus on targeting P.C gaming.

Functional requirements

1. Full player-controlled movement around the play area.
2. Provide game levels adequate for player to interact with.
3. Adequate fundamental physics mechanisms (gravity, etc.).
4. Additional physics elements core to gameplay (grapple, etc.).
5. Level end conditions, a set objective which ends the level (Level can conclude).
6. Start screen / home screen (A way to launch game).
7. End state / end screen (Game can conclude).
8. Resetting on failure, preventing 'softlocking' player (Player can always progress).

Non-functional requirements

1. Optimizing scripts so that they do not cause unnecessary lag / performance issues.
2. Creating levels of adequate difficulty, so they are not too easy or too hard.
3. Ensuring there are as few bugs as possible which could hinder gameplay.
4. Make game and menus as intuitive as possible, so players do not need to spend lots of time learning how to play or use the menus.
5. The graphics and models should be adequate.
6. Sound effects and music.
7. Experience enhancing graphical features such as field of view change on acceleration.
8. Making the game fun, above all else the game must be fun for a player to want to play it.

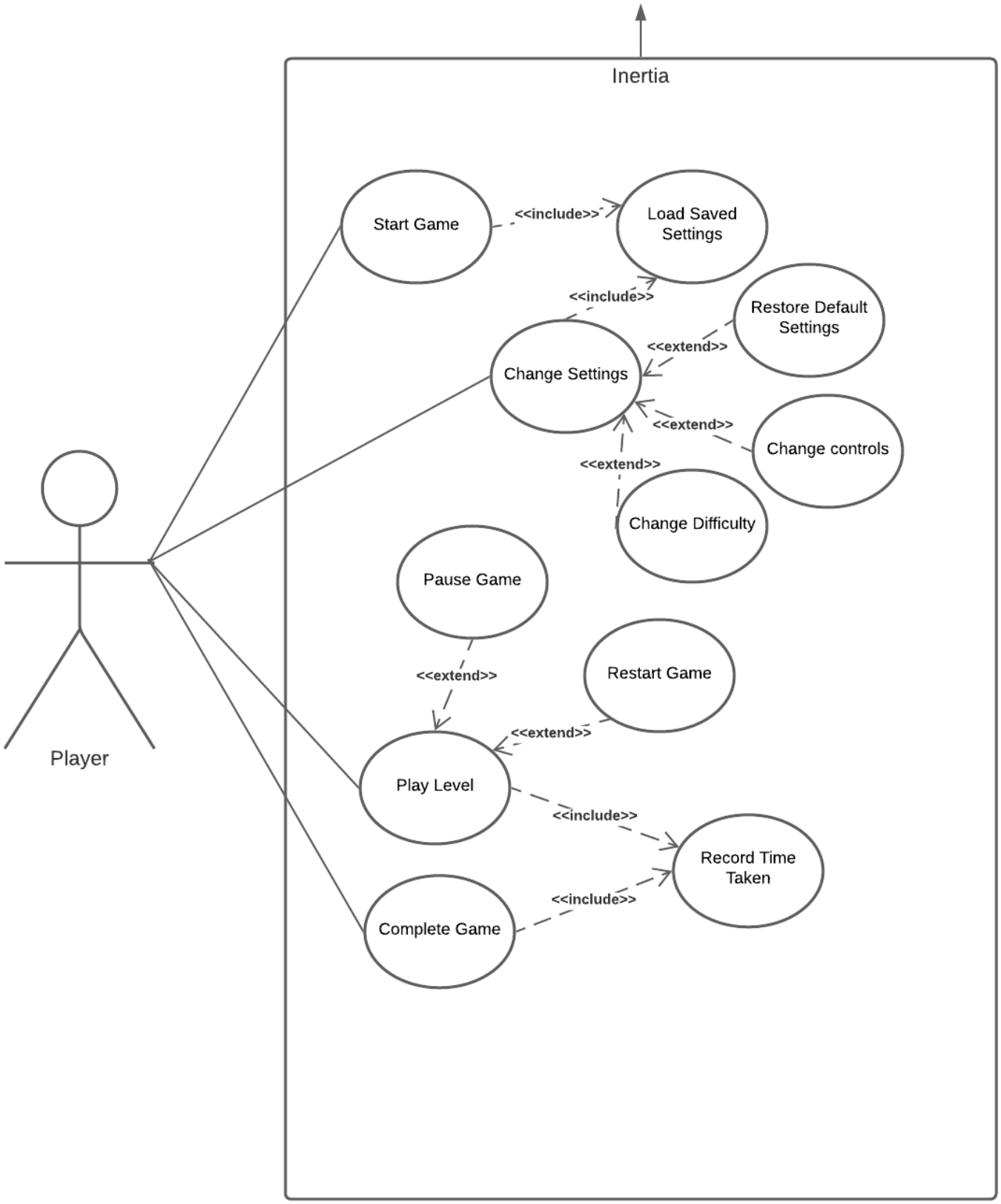


Figure 23 - Use Case Diagram

Use Case Diagram

In the following explanations, “Player”, refers to the Player actor, which is the user. For this use case diagram, there is only one actor. There is only the Player interacting with this game, as it is a single-player game. The rectangle represents the system, which in this use case diagram is the game, titled Inertia.

The first base use case is “Start Game”. This describes when the Player launches the game and arrives at the main menu. Functionality must be there to boot the game and load all the assets and game code into memory, ensuring the game is launchable. One of the included use cases for this is to “Load Saved Settings”, this will load the settings from the local storage on the computer, so that settings preferences would not have to be re-entered every time the game is launched. If this is the first time the Player actor is playing the game, the game will load the default settings.

Next base use case is “Change Settings”. This includes the use case “Load Saved Settings” also, ensuring that the settings the Player is changing is up to date with their previous changes (if any). The use case “Restore Default Settings” is extended, this functionality is needed if the Player has accidentally messed up their settings and wishes to restore the original settings the game came with, or as a way to restore if the Players settings somehow became corrupted. “Change Controls” is extended, this allows the Player to change the input controls that allow them to move around the world and interact with the game. “Change Difficulty” is extended, allowing the Player to modify the difficulty of the game, the exact implementation of this functionality is still in early development phases and planning, but will make the game more difficult or easier depending on the setting.

Third base use case is “Play Level”. This is the use case that describes all the gameplay of an individual level in the game. The Player will have a way to access this functionality through the main menu (represented by the “Start Game” use case). This is the main use case of the game, the Player will navigate the world and will (likely) attempt to complete the objective of the level, which would be to reach a certain point in the level. This base use case extends “Pause Game” use case, allowing the player to pause the game in time, useful if they need to take a momentary break from playing but do not wish for their progress in the game to reset. Also extended is the “Restart Game” use case, this allows the Player to restart the game if they want to abandon their current progress and return to the start of the game. “Record Time Taken” is included here, which will record the time it takes the Player to complete a level.

The fourth base use case is “Complete Game”. This is the functionality the Player invokes when they complete all objectives of all levels. The exact implementation of this use case is also still in development. It may run a routine script of displaying credits and returning to the main menu, saving the time if it beat all previous completion times. “Record Time Taken” is also included here, this is the functionality that will record the time it took to complete all levels of the game, and save it to local storage if it beats all previous Player records.

Feasibility

This game is going to be developed in the Unity game engine, in the 3D game creator. The game developers decided to keep a common unity version and not update it during this project (Unity version 2020.1.6f1). For sharing individual progress with each other, they will use the in-built unity collaborate tool. They both signed up with their student accounts to get Unity Pro for free, which makes sharing the project easier. The project is periodically pushed to a private GitHub repository, this is to keep a version history, keep track of updates and as a backup in case something goes wrong. The Unity asset store will be used for getting some free assets for the game, this can be a source of problems as the target version of the assets on the store do not always match up with the version in use for the game. This can be circumvented by either manually updating the assets or ensuring not to use out of date assets.

For model and texture creation, 3D computer graphics software called Blender will be used. For images and icons, Photoshop will be used. The website Trello (https://trello.com/en) is being used for planning and development. Instant messaging and VoIP software, Discord, is being used to coordinate team effort. There are periodic meetings on discord to discuss progress and ideas, as well as problems and brainstorm solutions to bugs.

Conclusion

This requirements chapter helped solidify direction and purpose with the game. The game will be developed with the results of the survey in mind, as well as the existing similar games of competitors. Due to this requirements chapter, it was possible to map out goals for the game more clearly, it is now more obvious which aspects of the game's development will need the most focus and time. The goals for the game are now more clearly defined. Ideas and inspiration have been drawn from existing successful games in this field, to be implemented within the game.

The information from the surveys conducted was particularly helpful. It is now planned to continue developing for the P.C platform, mobile game development was briefly considered. The most important areas of the game are more clearly understood, for example the survey results suggest the developers should focus development more on level design than on audio for this genre of game. Other feedback from the survey suggests that existing games in the genre can lack an exciting edge, this has prompted further investigation into more exciting game concepts such as grapple hook movement.

Outlining the exact requirements allows the developers to plan, the workload is now easier to judge. Knowing exactly what is needed for this game to be finished makes it much simpler to gauge progress and set deadlines. Game features which were previously in limbo have now been either scrapped or kept based on this segment of surveying and viewing competitor games, which massively solidifies the direction of this project.

Tools and information

Use case diagram creator: https://lucid.app/lucidchart/

Use case diagram for a game info: https://stackoverflow.com/questions/58031632/am-i-heading-in-the-right-direction-with-my-use-case-diagram

Use case diagram creation information: https://www.youtube.com/watch?v=zid-MVo7M-E

Refunct reviews: https://store.steampowered.com/app/406150/Refunct/

Game survey: https://forms.gle/SbSP3fNUuBd14RkA7

Trello (used for coordination): https://trello.com/en

Discord (used for development chat): https://discord.com/

3 Design

Introduction

The product is a 3D movement-based physics video game. The player navigates through levels using a grappling hook projectile gun to reach a location within the level. The player assumes control of an unnamed test subject who is armed only with a grappling gun. The grappling gun shoots a projectile which pulls the player towards the surface the hook strikes. The player needs to use momentum gained from this action to progress through the level. This unnamed test subject has the objective of escaping the laboratory they are trapped in by solving various physics-based puzzles using the grappling hook gun.

The game is beginner-friendly but can also be a challenge. It is designed to be easy to pick up but difficult to master. A timer is added into the game for those skilled players who are interested in beating the game in the fastest time possible. The game requires the player to make their way from one end of the level to the other while jumping across and around platforms. It will be a relatively short game with the core focus being on gameplay and level design.

This game is being developed in the Unity game engine. The Pro version of Unity is available to students free of charge, so the Pro version is being used. Unity comes with a useful tool called Unity Collaborate, a version control system, which syncs the project as changes are made, making development as a team much easier. Github is used periodically to log major changes to the project, Github is also for logging major issues that arise. The game scripts are coded in the C# programming language.

Program Design

Technologies

Unity:

Unity is a game engine used to massively simplify video game creation, and it is free. It is very accessible and easy to learn as there is a lot of documentation for it available free online. Unity will be used to create and edit the game's assets and setting up various scenes the game will have.

Unity is the core tool that handles rendering, game memory, input/outputs, physics calculations, and much more for the game. Unity also has a collaboration version

control system feature allowing development to sync changes seamlessly within the program without having to do any major downloads.

Affinity:

Affinity Photo is one of the best programs available for creating graphics and textures for the game. This is used in Inertia to create textures and graphics for the game such as the Main Menu screen, Game Over screen, and textures/colours for the various assets.

Visual Studio Code:

Visual Studio Code is used to edit and create the C# scripts that make up the code for the game. Visual Studio Code is useful as it has plenty of plugins and extensions as well as having a very modern interface and it is easy to use and easy to manage.

Visual Studio 2019:

Visual Studio 2019 is also used for creating and editing C# scripts. It is similar to Visual Studio Code but is a lot more heavyweight, containing a lot more functionality out of the box.

Github:

Unity Collaborate is the main tool used for version control in this project, though Github is used for periodic versions to store a history of changes in case something goes wrong during development. Github is where issues are logged and addressed as well, in the issues tab of the project.

Trello:

Trello is a list application used for planning development through a series of cards. It is very useful for coordinating efforts of development on the game, especially for planning what needs to be done for sprints. As well as logging what features have been finished and what still needs to be done.

Discord:

Discord is a program for communications originally designed for gamers to talk to each other. It is evolved into a multipurpose chatting program with support for embedded code with syntax highlighting. This is used for communicating and coordinating on the project.

Alternatives:

Alternatives to these were considered. Blender was considered for level and model development, though Probuilder plugin in Unity took over in this area, as it was easier to learn and faster to develop in for Unity. Photoshop was considered for graphic development as well but was less suited for video game asset development than other software. Developing a mobile game was also considered, as Unity has support for creating mobile games, though this idea was scrapped as it was not practical to make or play this game on a mobile device.

Structure of Unity

A game created in Unity has a specific number of folders and files in it that all exist in 2 specific but different hierarchies.

The first one is known as the Project hierarchy, this contains all the assets and plugins ready for use and to be placed in the game's scene, an example is seen in figure 1 below.

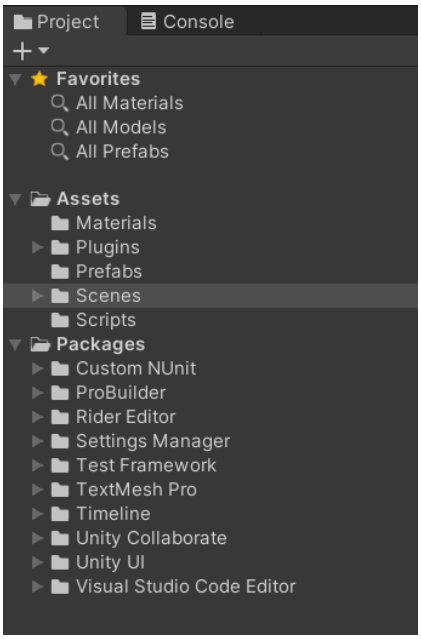


Figure 19 - Unity Project Hierarchy

There are 2 main folders here The Assets folder and the Packages folder. Within the Assets folder, there are the individual files such as scripts, game assets, and textures which we keep here and drag into the game scene as needed. It is also easy to manage to organise as you can put folders within folders to micromanage everything. E.g. In the Scripts folder, there will be the scripts created for this project. See Figure 2 below.

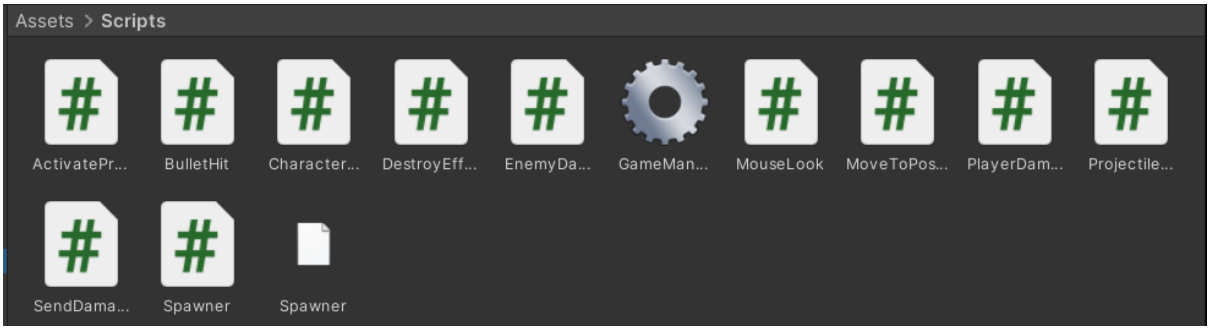


Figure 20 - Scripts in the Scripts folder

The second main folder in Unity is the Packages folder. This folder contains various extensions that the user may need for their game, such as the Probuilder extension. This package allows the user to create a level in a matter of minutes. See the layout in figure 3 below.

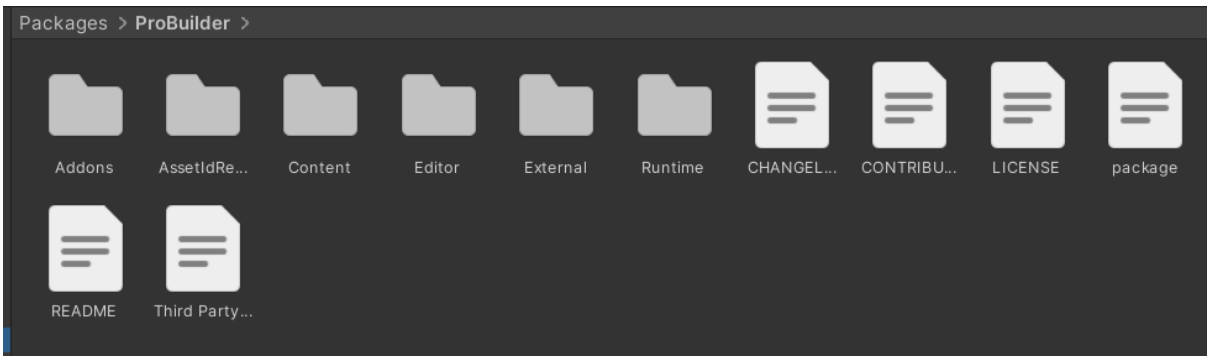


Figure 21 - Files in Probuilder Folder

The other hierarchy in Unity is in the scene, this hierarchy is everything that is currently in the game scene that appears on the screen. To add something to this scene, you simply drag the file you want to add from the assets folder into the scene hierarchy. Everything in here is from the previous hierarchy that contains the assets and packages. E.g. Figure 4 below.

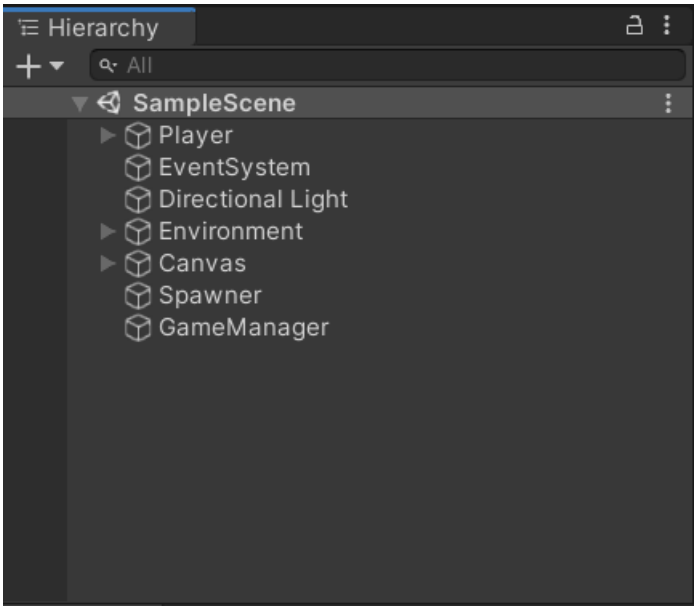


Figure 22 - Scene Hierarchy

Those are the main structures available in Unity. It makes it easier to track assets and the various components of the game.

Design Patterns

Unity does not have a set software design pattern. However, this project aims to be consistent across scripts, with stylistic consistency with Unity's coding conventions. In general, the coding style should be consistent with C#'s outlined coding conventions, only differing when Unity's specific style also deviates (https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions).

Application architecture

The game's architecture is heavily reliant on the Unity game engine. The code for the game is compiled when exported. Unity makes use of a technology called IL2CPP to convert C# code into C++ code to generate code that works on more platforms. This process can also improve game performance.

The application has an initial main menu that greats the user when they start the game, this menu gives them the opportunity to modify settings such as graphical settings, controls, and volume. On the main menu there is a button to begin playing the game. When the user clicks this, they will be brought into the game. The main menu acts as a control area for the player, it obeys the typical convention for video games that experienced gamers will expect. Similarly, there is a pause menu that can be brought up during the game, this menu has similar functionality to the main menu. It allows the player to modify their settings but also allows the player to leave the game.

All specific architecture aspects for this project are controlled by the Unity game engine. Such as input, calculations, game memory, physics, rendering, etc. An issue that arises with this for other games developed in Unity is that cheats can rapidly be developed, as existing video game cheating software already exists for other Unity games. This is less of a problem for this project though, as cheating only really causes damage to multiplayer games. This is one of the drawbacks of Unity though, it will be easier for a hacker to crack the game and release it for piracy.

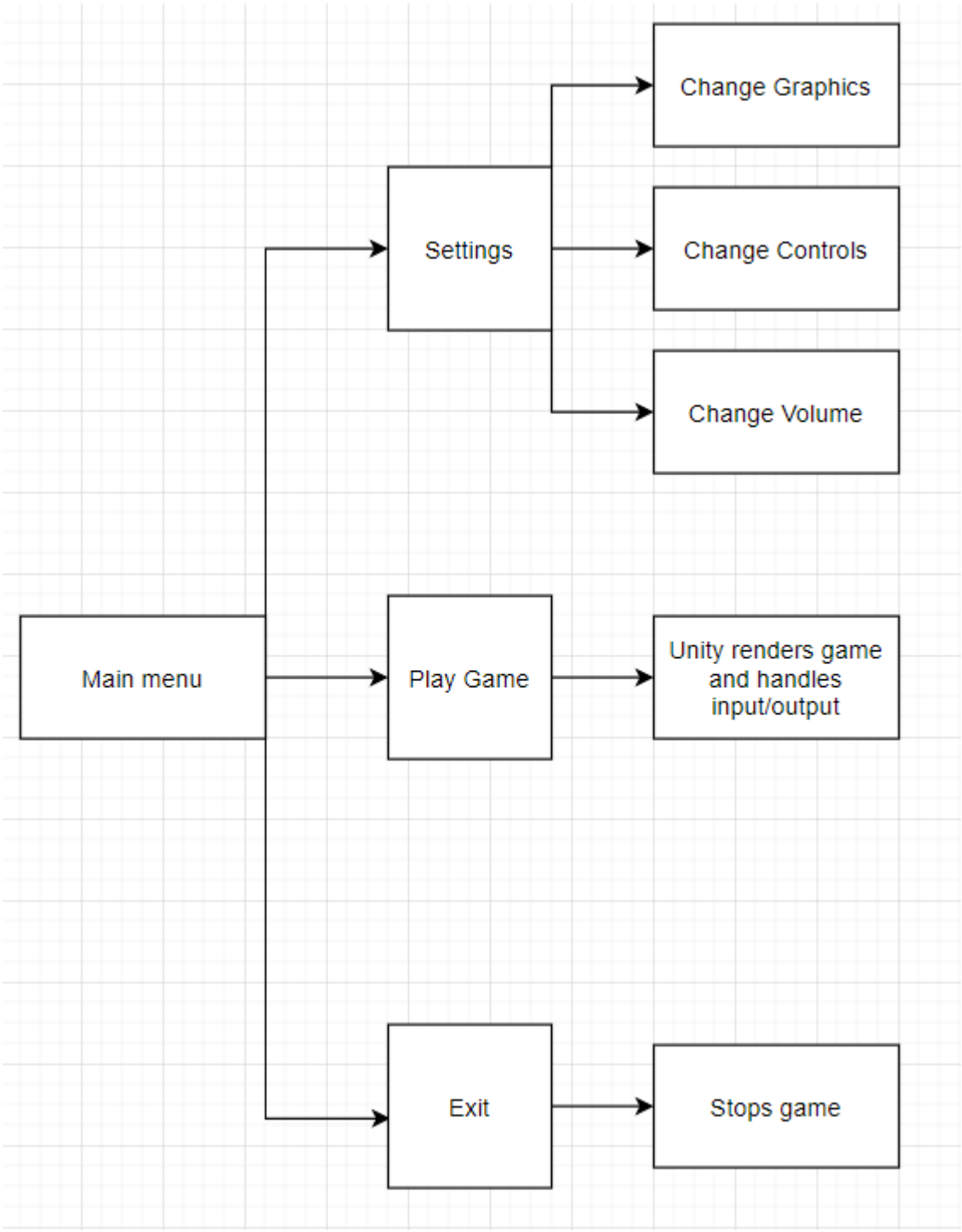


Figure 23 - Block diagram for Inertia

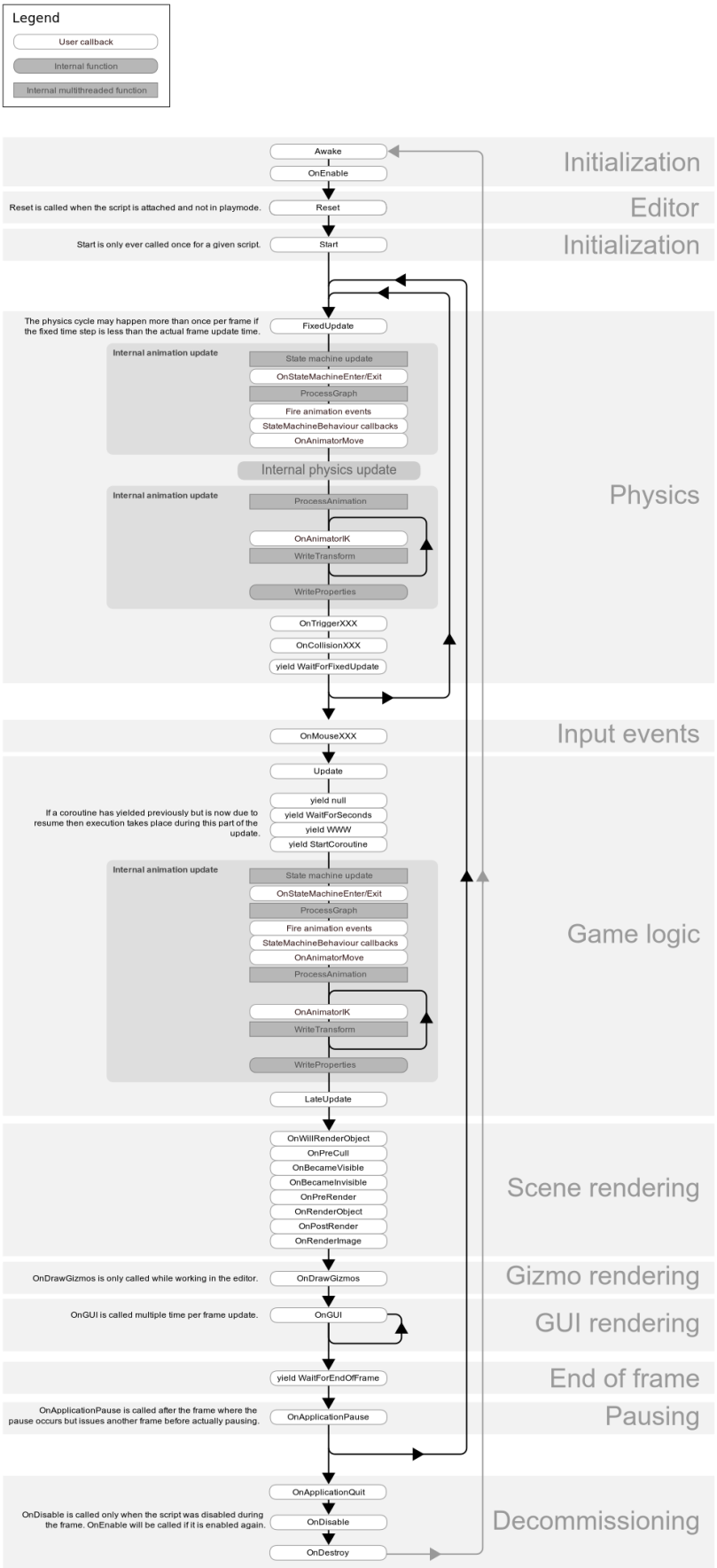


Figure 24 - Unity Mono application architecture, from https://docs.unity3d.com/uploads/Main/monobehaviour_flowchart.svg

Database design

Inertia is single-player and therefore does not have any database, so this section does not apply. All online interaction for the sale of the game would be handled by a third-party game launcher such as Steam.

User interface design

The UI for this game is designed to be straightforward and clear to the player. The menus obey typical conventions for video games as previously mentioned, so it should be easy to adopt for experienced gamers. The interface is designed to be intuitive for people new to gaming as well, this is the reason plain language and structure is used in favour of technical language or a complicated menu structure.

Wireframe

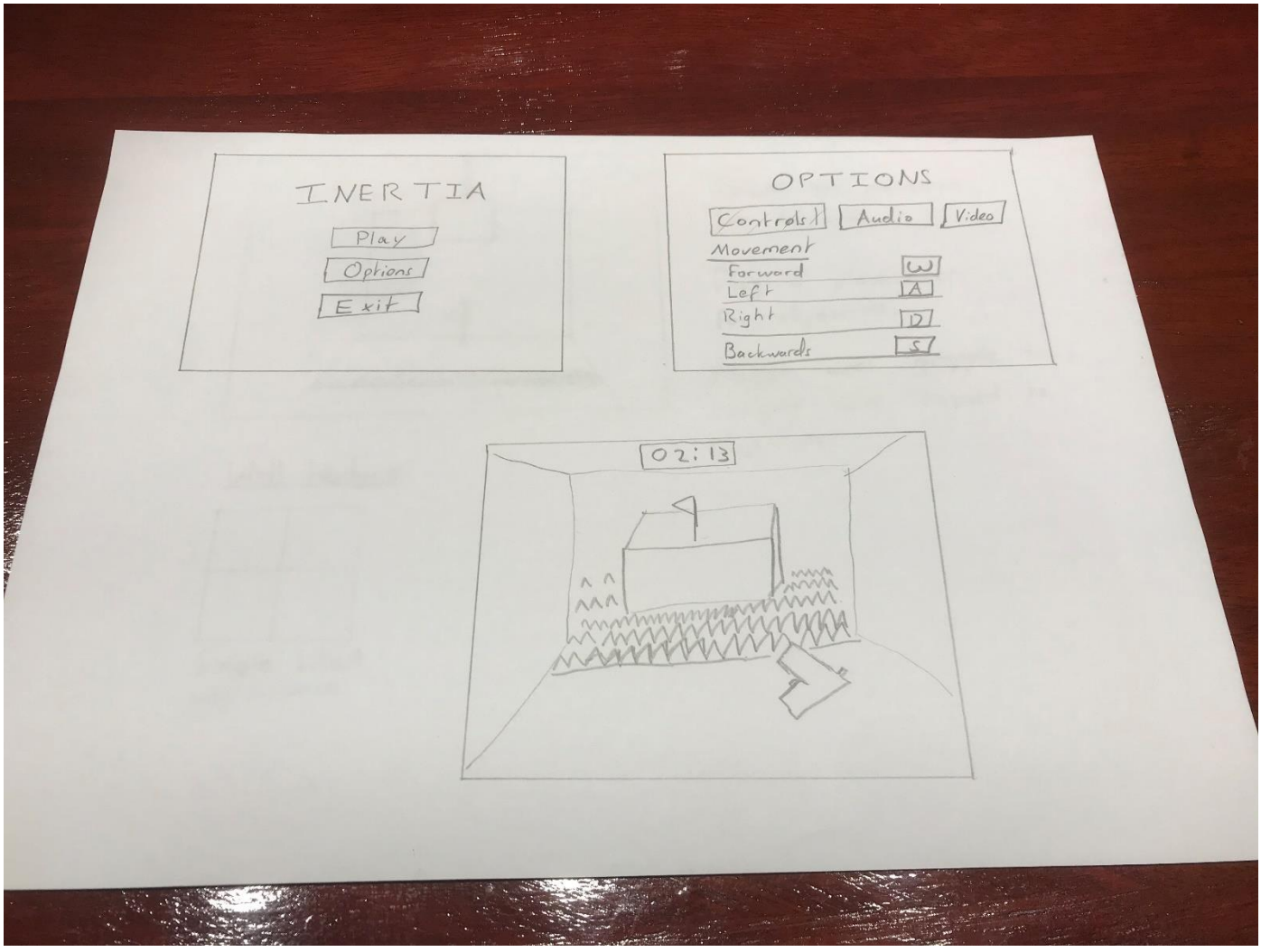


Figure 25 - Game sketch

A traditional wireframe is not applicable to this game. Instead, this is a representation of the main menu, the options menu, as well as the play state. The

flag in the final frame represents the objective, the spikes represent hazards. The player has a grapple gun in hand to navigate over the hazards to get to the objective platform.

User Flow Diagram

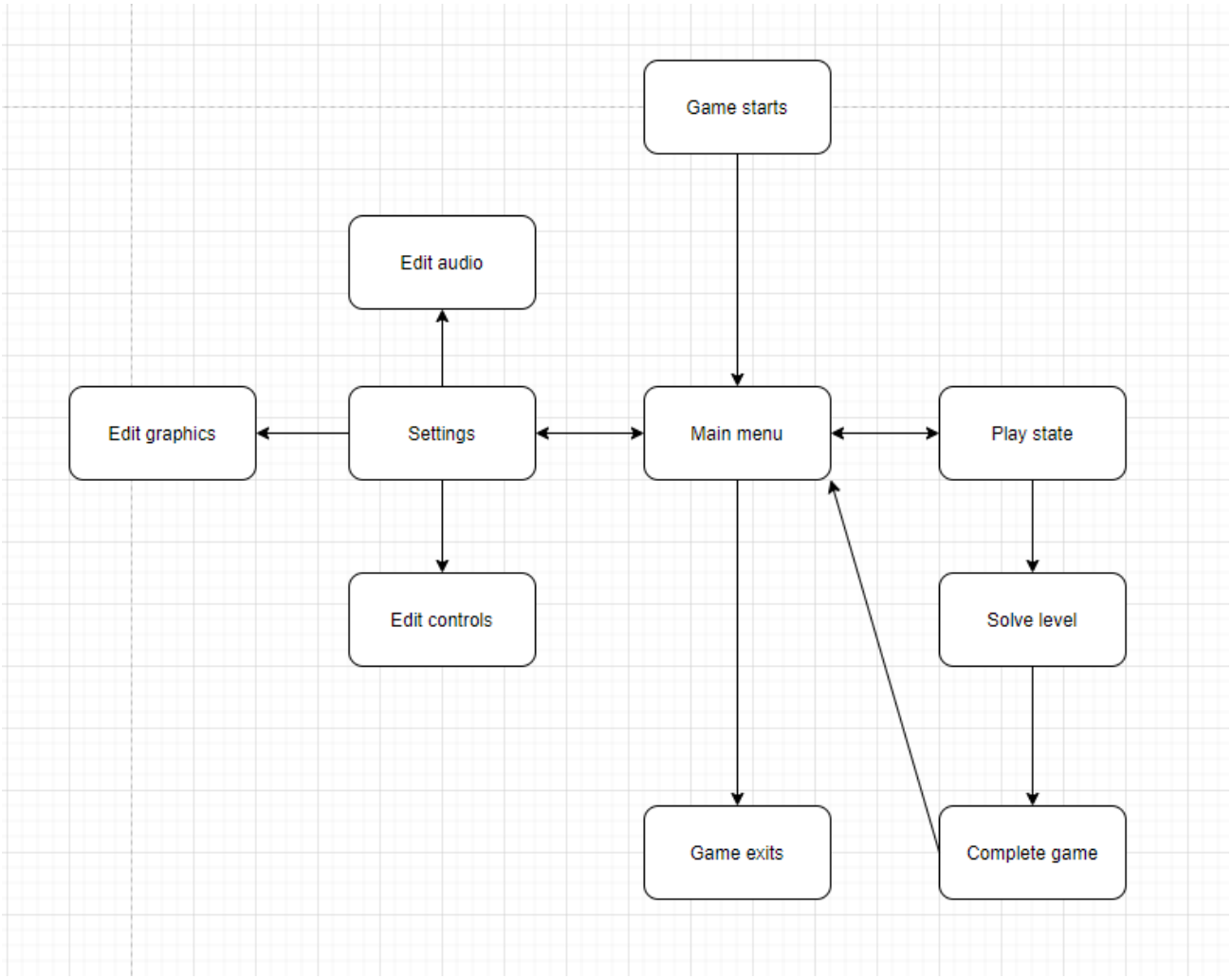


Figure 26 - User flow diagram for Inertia

This diagram represents the various actions a player can choose within the game, in the order that they can be chosen. Double-sided arrows represent direct navigation being possible between both states.

Colour Palette

Our game takes place in a lab so we will be using more muted and plain colours to match the aesthetics

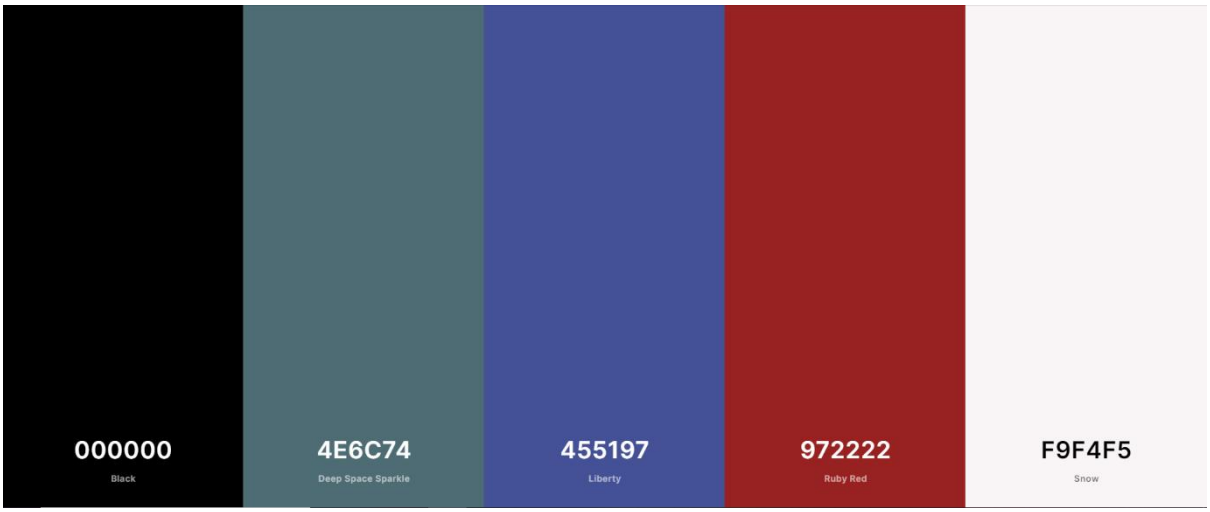


Figure 27 - Colour Palette

Typeface

The font we will be mainly using in this game is Liberation Sans it is very simple and straight forward and fits the theme of our game which is set in a lab

Figure 28 Liberation Sans

Menu’s

The menu aesthetic for the game looks like this.

Paper Prototype

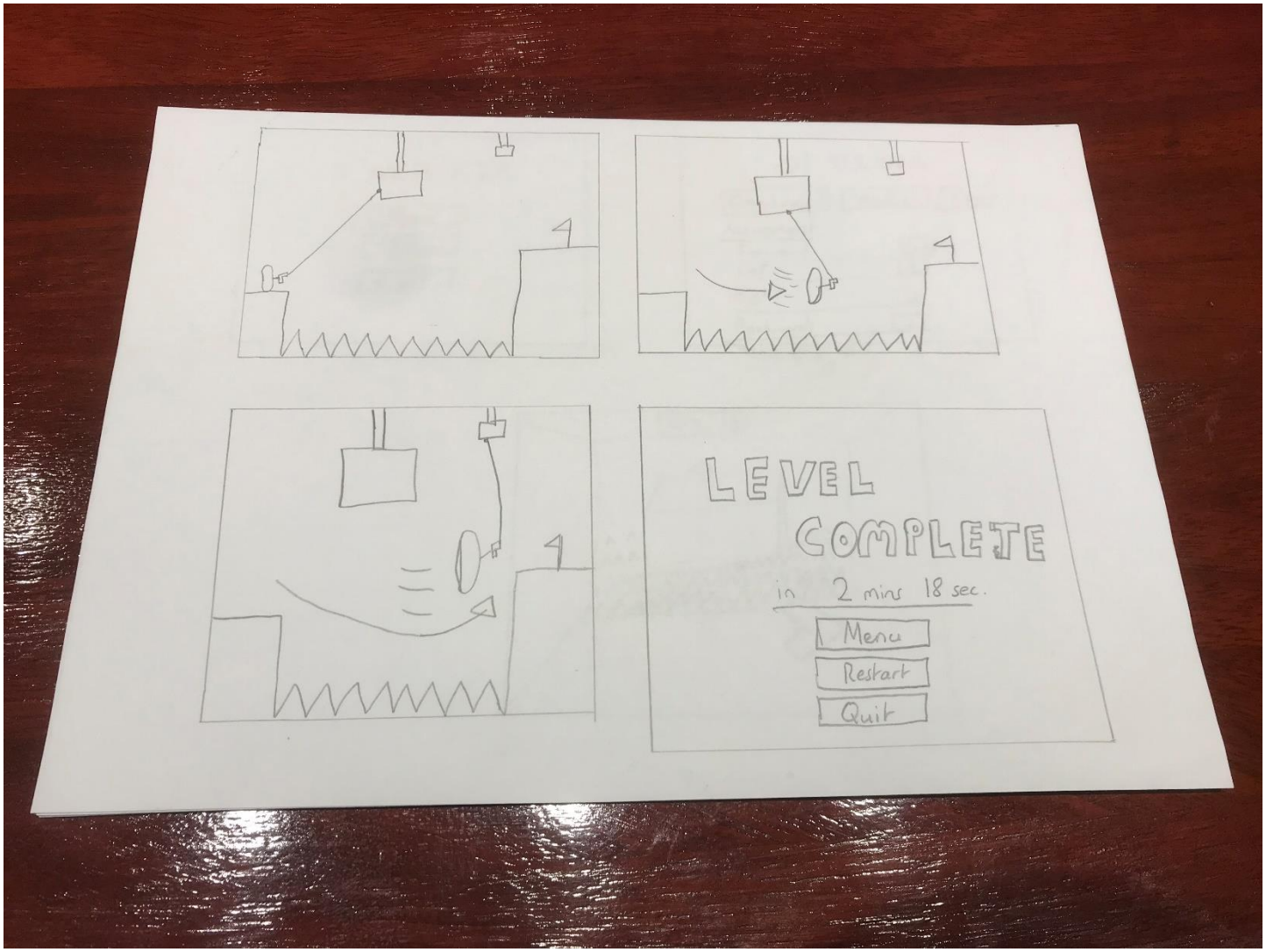


Figure 29 - A paper prototype for the game

This paper prototype shows a 2D representation of a 3D plane. The progression of this prototype is left to right. The player is represented by the capsule. The player uses a grapple gun to navigate from the starting platform to the goal platform (represented by the platform with a flag).

The grapple gun shoots a hook onto the cubes attached to the ceiling of the level, the player is pulled towards the cube, the player uses this momentum to swing over the hazard represented by the spikes that separate the starting platform from the

goal platform. In the third panel, the player gains additional momentum by shooting the grapple gun again at another smaller cube attached to the ceiling.

In the fourth panel, the player has reached the goal platform, which completes the level. The time it took to complete the level is then displayed to the player.

Storyboard

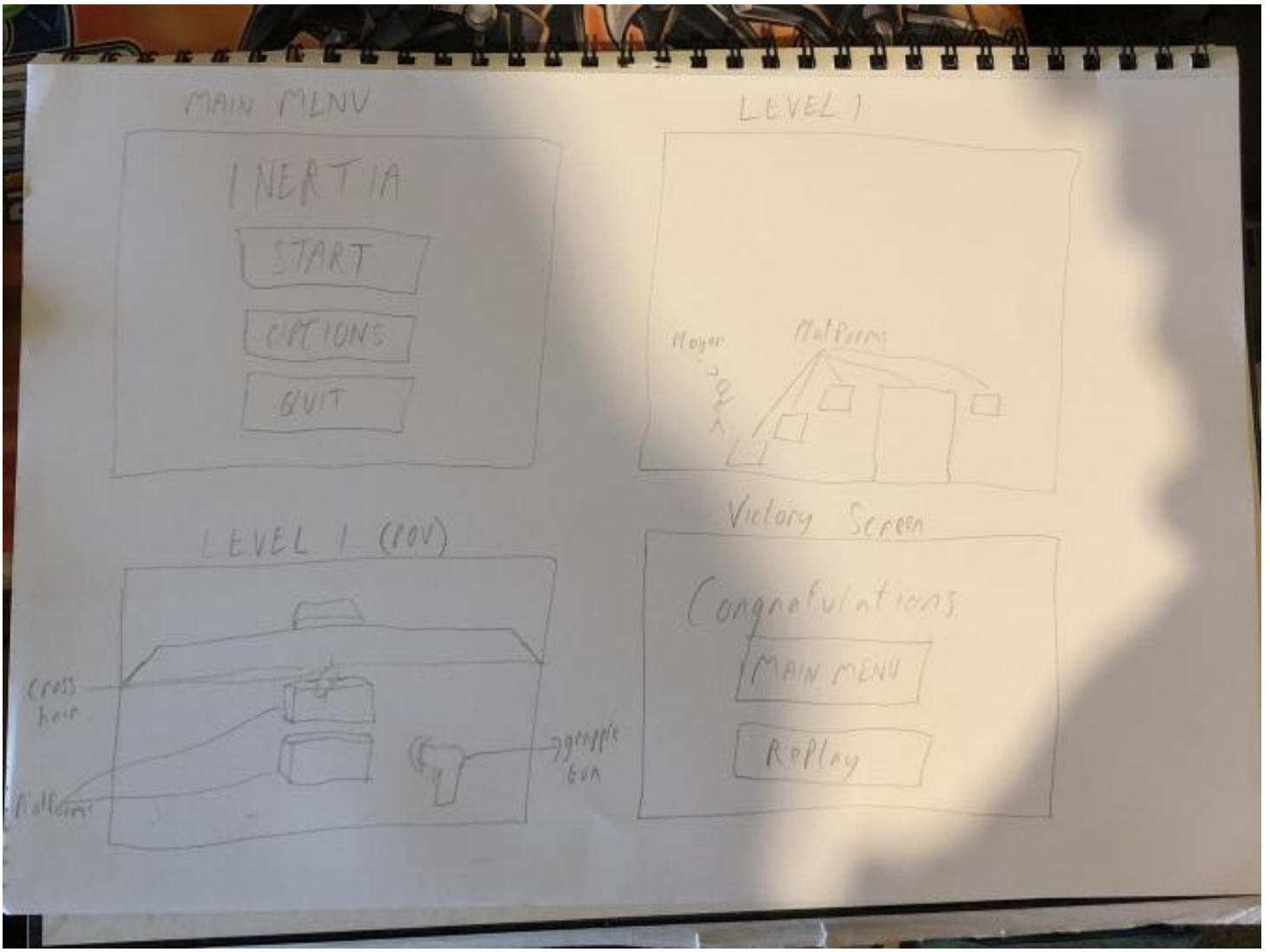


Figure 30 - Storyboard

Level Design

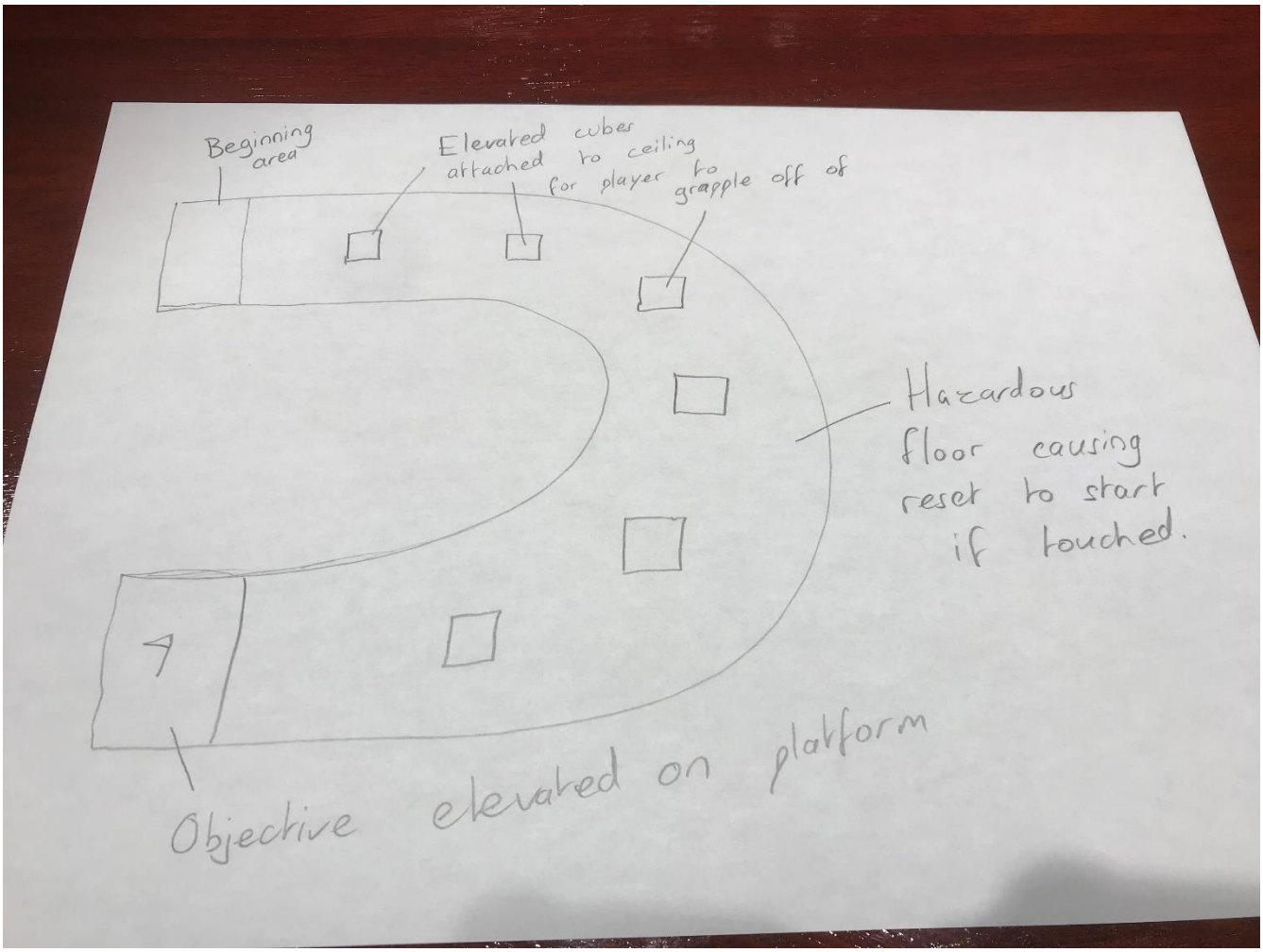


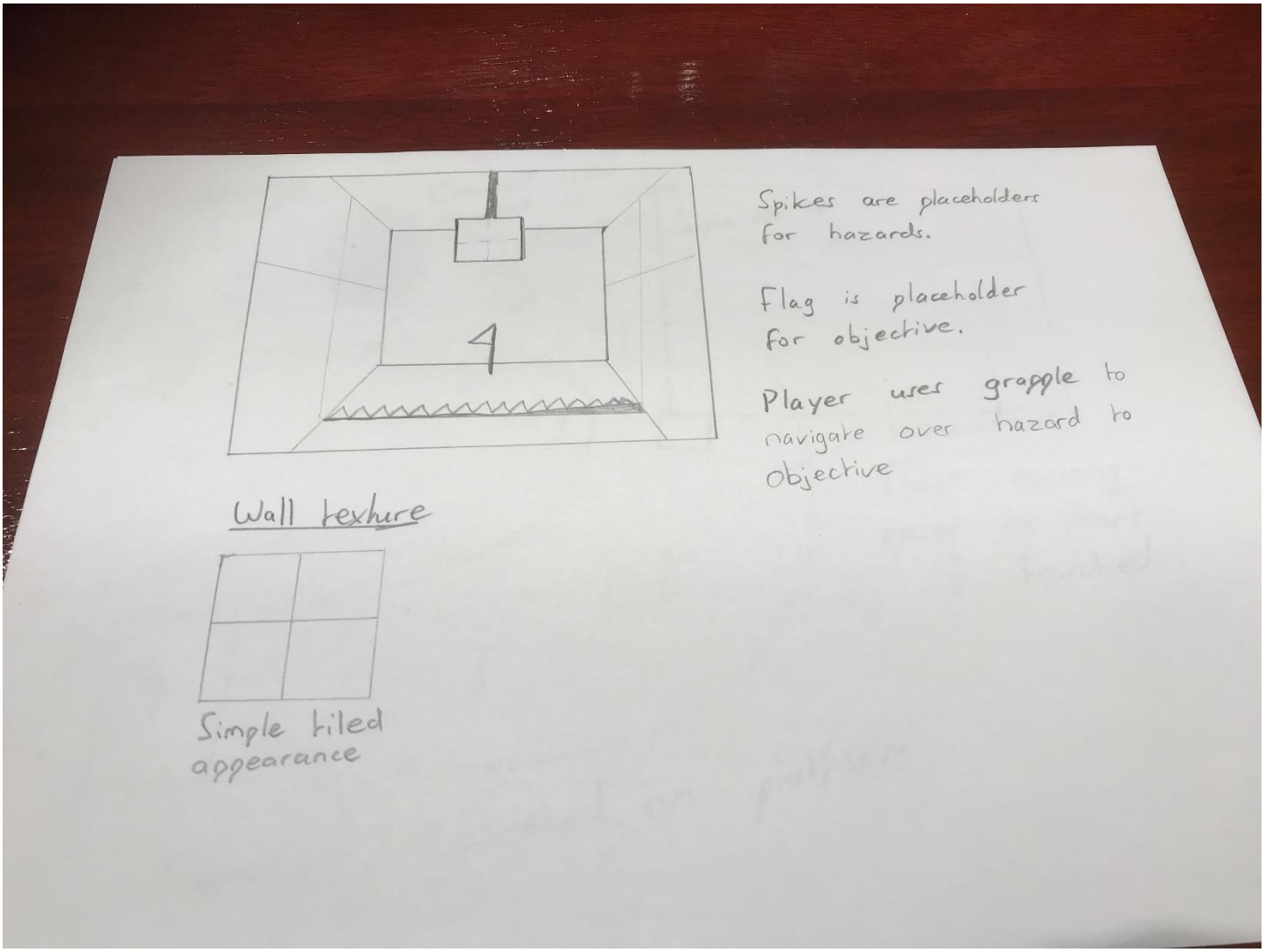
Figure 31 - An example of a level in Inertia

This is an example sketch in 2D drawn from a bird's eye view of a level. The lines on the side represent walls, the walls curve around from the beginning platform to the goal platform. In this particular level, all of the floor is hazardous, if the player touches it, they will be reset to the start.

Their objective is to grapple from cube to cube (which are attached to the ceiling of the level) and build enough momentum to reach the elevated platform at the end. If they lose momentum, they will not have enough speed to reach the final platform and they will instead fall to the floor and be reset.

Upon reaching the final platform at the end, the player can move on to the next level. A timer shows them how long it took for them to complete the level.

Environment



In the sketch at the top, there is a 3D representation of a level. The setting is indoors, with a focus on giving a lab feeling to the player, this is inspired by the highly successful Portal video game series games by VALVe (https://en.wikipedia.org/wiki/Portal_(series)).

In the environment of this game, the texture for the surfaces is a plain tiled colour. As previously mentioned, this game takes place indoors, this makes it more obvious where the player is supposed to go as the level is linear in direction. The levels will feel purposely claustrophobic and restrictive, this again is inspired by the Portal series, where the player character is a test subject to scientific testing.

Conclusion

This design chapter outlines the stylistic expectations for the video game. The User Interface and feel of the game are narrowed down. The style and aesthetics of the game are decided. The technologies involved in creating this game have been explored. The level design for the game has been developed upon, making it easier for the developers to create more levels with the same theme.

The expectations of the game and the functionality required have been made clearer for the developers. Unity's structure, as well as Unity's internal architecture, was investigated, this should give the developers a closer understanding of the technologies involved in creating the game.

User flow diagrams and sketches should also help with the development of the game as the desired user experience has been outlined.

4 Implementation

Introduction

The purpose of this chapter is to describe the implementation of assets and software solutions in the game.

The development of the game has depended primarily on the use of the following software:

- **Unity**:
  This is a game engine developed by Unity Technologies. It is free to use, and the Pro version Is available free to students. The Unity Collaborate feature makes developing with other people very easy. This project is being developed in Unity version 2020.1.6f1. The decision to stay on one version was taken to avoid the issues that can arise when upgrading a Unity project, as well as make it easier to develop without having to continually download new versions.
- **Blender:**
  Free open-source 3D model creation program. This software was used to make many models in the game – such as the rocket launcher model and rocket models.
- **Adobe Photoshop:**
  This raster graphics editor was used for editing various textures and improving 2D graphics.
- **Visual Studio 2019:**
  This is used for the development of the scripts for the game. This code editor has comprehensive support for a wide range of programming languages. It directly hooks up to Unity, so it can give the developer information about Unity classes and functions.
- **Google Cloud VM instances:**
  This is a service provided by Google. It allows you to pay for use of their servers. This is used for testing and hosting multiplayer over the internet.

If the game were to be published, it would use “Steam”, a game launcher that takes a fee in exchange for managing launching and installing games.
https://steamcommunity.com/

This application is a game based on applying different physics movement mechanics in a 3D world to achieve a goal, such as reaching a location or defeating enemies. One of the main focuses is having an easy to set up multiplayer system. It is titled “Inertia” as a reference to the movement mechanics within the game which tend to allow objects to retain the movement they have obtained unless slowed by something (once a player gains speed they mostly retain it unless something slows them).

Implementation roles

My roles in the completion of this game include:

- Majority of the C# scripts used for the game.
- Creation of independent redistributable server software written in C#.
- Creation of client scripts that handle and implement multiplayer features on the client.
- Extensive testing for game scripts and multiplayer functionality.
- Much of the User Interface and User Interface scripts.
- Designing and creating a portion of the level design.
- Asset creation such as 3D models in Blender.
- Graphics improvements in Photoshop.
- Implementation of 3D special effects such as explosions and explosion audio, impacts etc.
- Research into multiplayer servers, testing various solutions to find the best fit for our game.
- Signing up for and setting up a Google Cloud VM instance to be compatible with hosting the server.

My partner was responsible for:

- Creation of various levels and level design.
- Sourcing some of the assets from the Unity asset store.
- Various C# scripts.
- Asset creation.
- User Interface development.
- Testing, research & ideas.

Scrum methodology

Scrum methodology was used in the development of this project during the implementation phase. Scrum is an agile development methodology for the development of software. Scrum methodology comes from a paper written in 1986[1]. This methodology focuses on having a general idea at first, and implementing it in stages, through the implementation of various features. The success of the implementation of these features after a period is reported to the scrum master. For this project, the supervisor was Catherine Noonan. A Trello board was used as a tracking mechanism for the tasks being implemented.

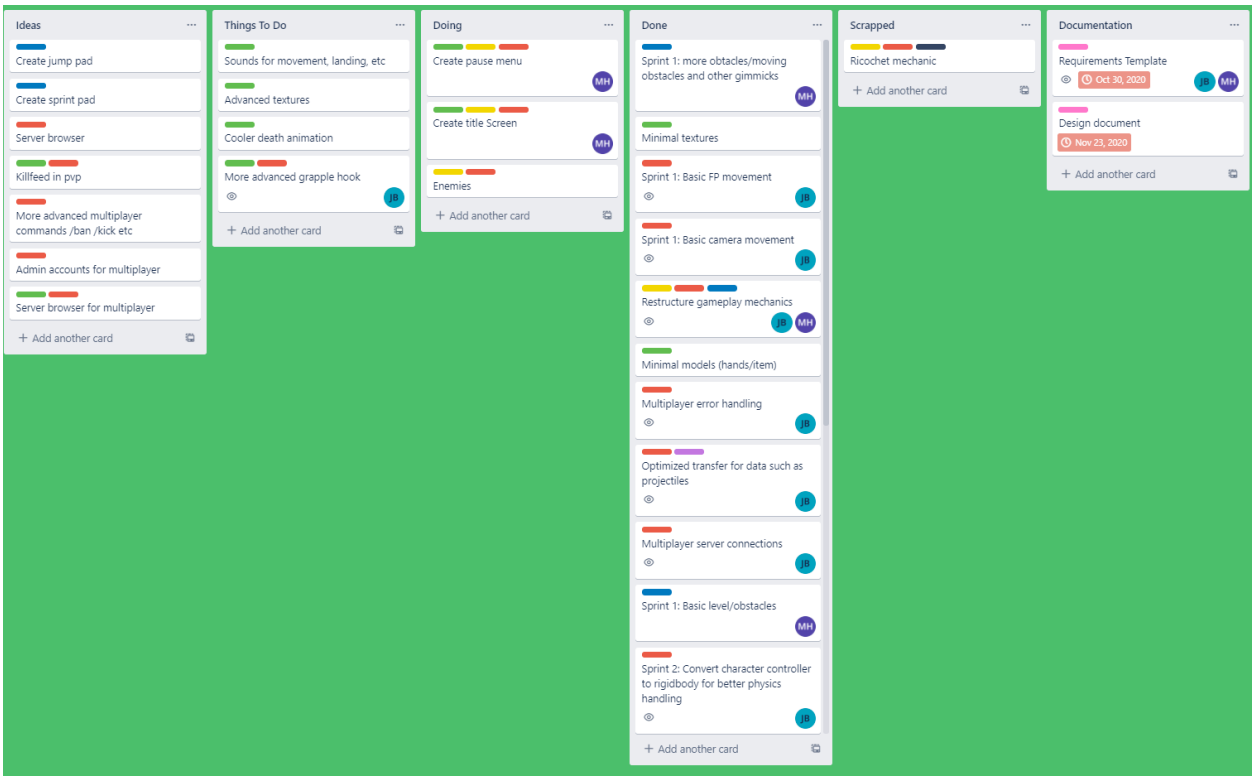


Figure 32 - Trello

The implementation phase for this project consisted of 7 sprints in total. There were 4 sprints before Christmas and 3 after. Each sprint lasted 2 weeks. During these sprints, specific features, elements and/or assets were developed for the game.

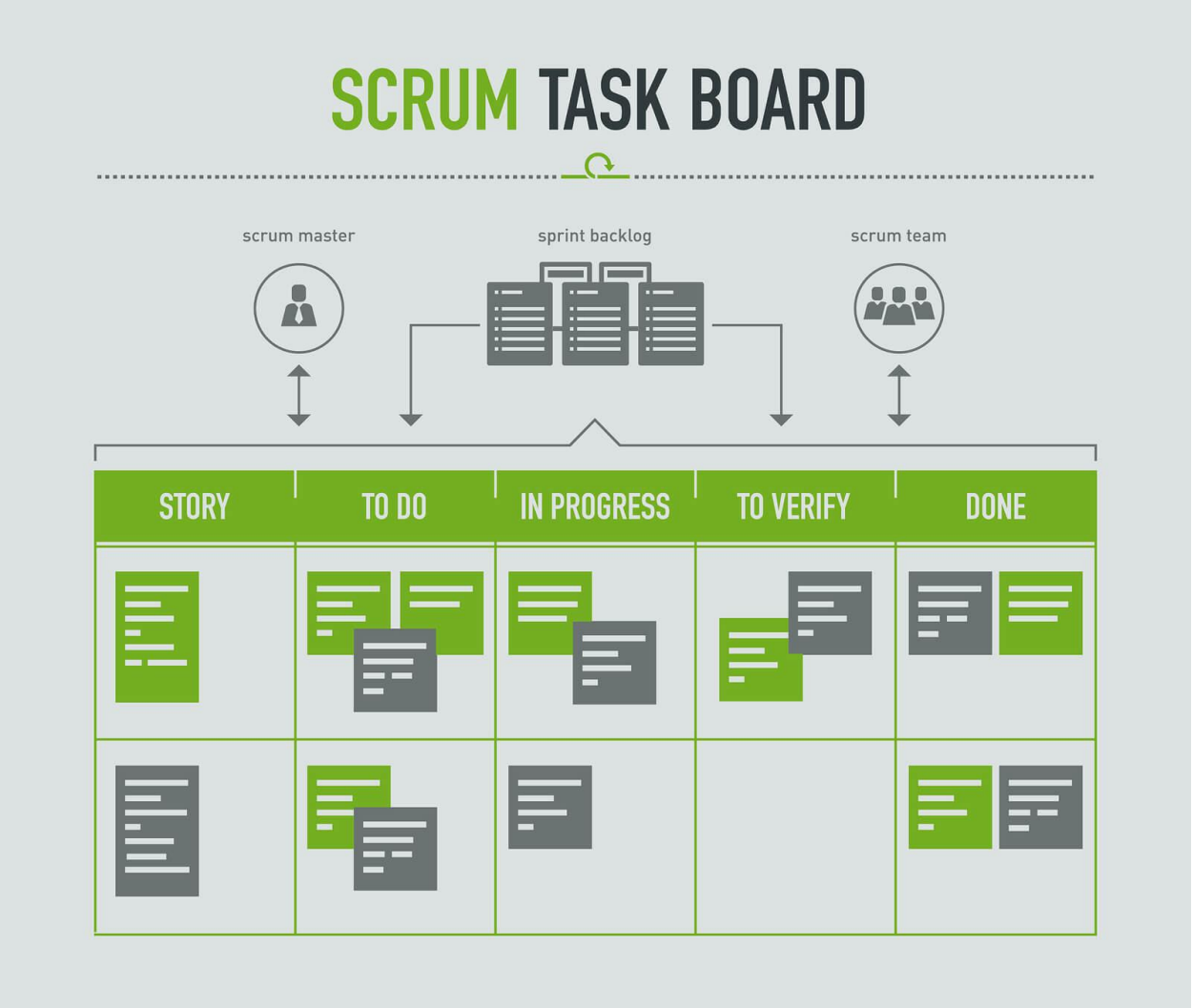


Figure 33 - Diagram (retrieved from listed source) [2]

The requirements for the application were listed in a product backlog. Each item on the product backlog was broken down into a series of tasks which formed a sprint.

Development environment

Microsoft Visual Studio 2019 was used primarily to develop the scripts for the game, and for the server. This IDE has many tools which aid development in C#, the intellisense hooks into Unity, it allows the developer to see information and signatures for Unity specific methods and classes, which is very useful and makes development faster.

Unity collaborate and Git bash were both used for version control for this game. Unity Collaborate empowered easy and rapid sharing of changes between the developers, this was developed by Unity technologies specifically for games that are being worked on in a team, it makes version control seamless and was very helpful for this project. This feature is free for students. Git bash was used to push the project to a remote repository on GitHub. There were 3 separate GitHub repositories for this project:

Client software: https://github.com/JonathanBerkeley/GameDevFPS/

Server software: https://github.com/JonathanBerkeley/Server

Main project: https://github.com/JonathanBerkeley/CC3_PROJECT

There are 3 different repositories because the server software is very involved, with many issues along the development, and deserved its own repository to track changes without affecting the rest of the project. There was also a repository for the client software which isolated client changes from the rest of the game to prevent issues that arose from regular development affecting the network development, also the issues that affect the client can be logged to the client repositories issues tab. The client software gets merged into the main project when it is stable enough.

Git was used very extensively with there being hundreds of commits, these commits all have messages describing the changes, and after some development also version numbers, so they can be navigated easily. This helps with bug diagnosis, as the version a bug was introduced can be investigated to find the cause more easily.

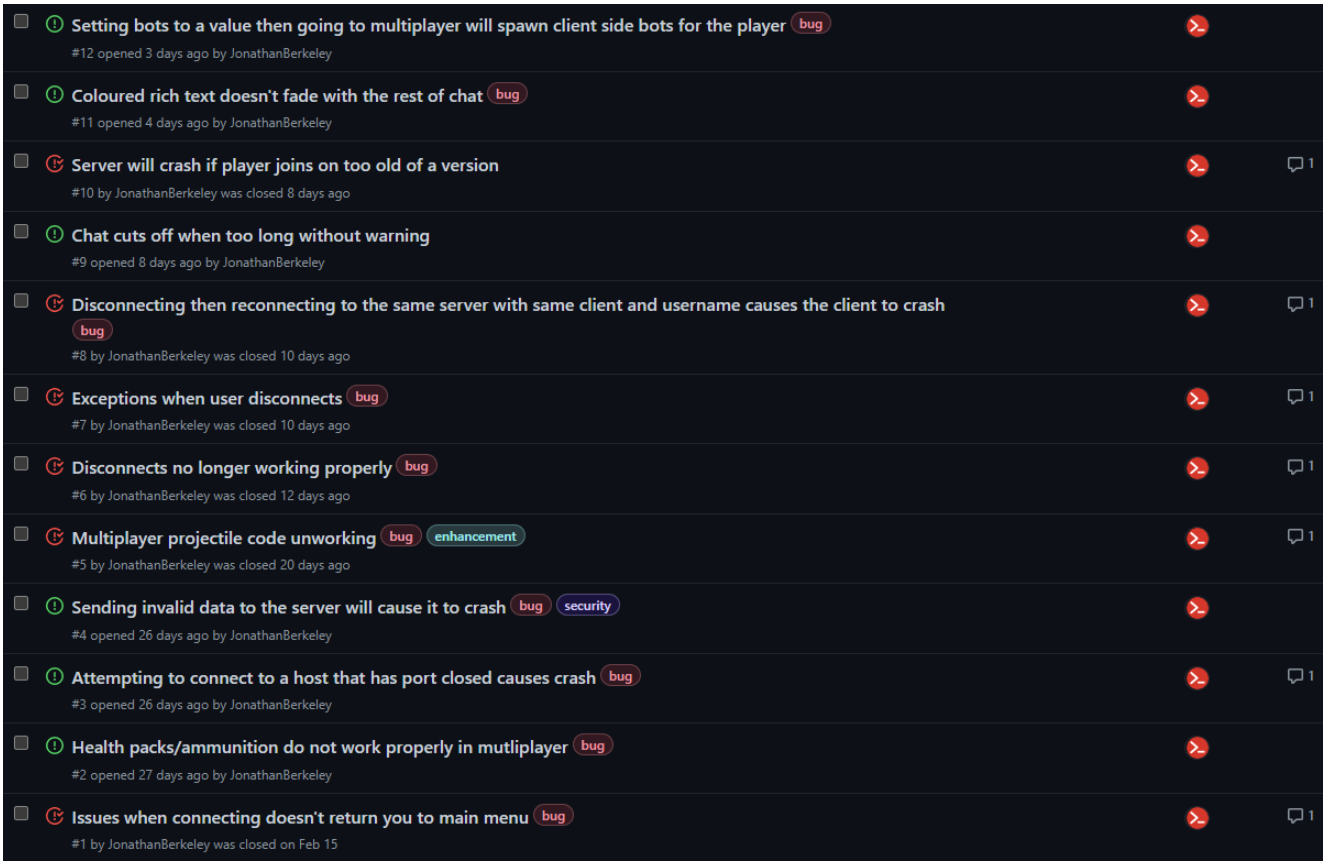


Figure 34 - Issue tab on client repository

The image above shows the issues page for the client repository as it appeared on 19/03/21. There is a separate issues page for the main project as well, located on the issues page of the main project repository.

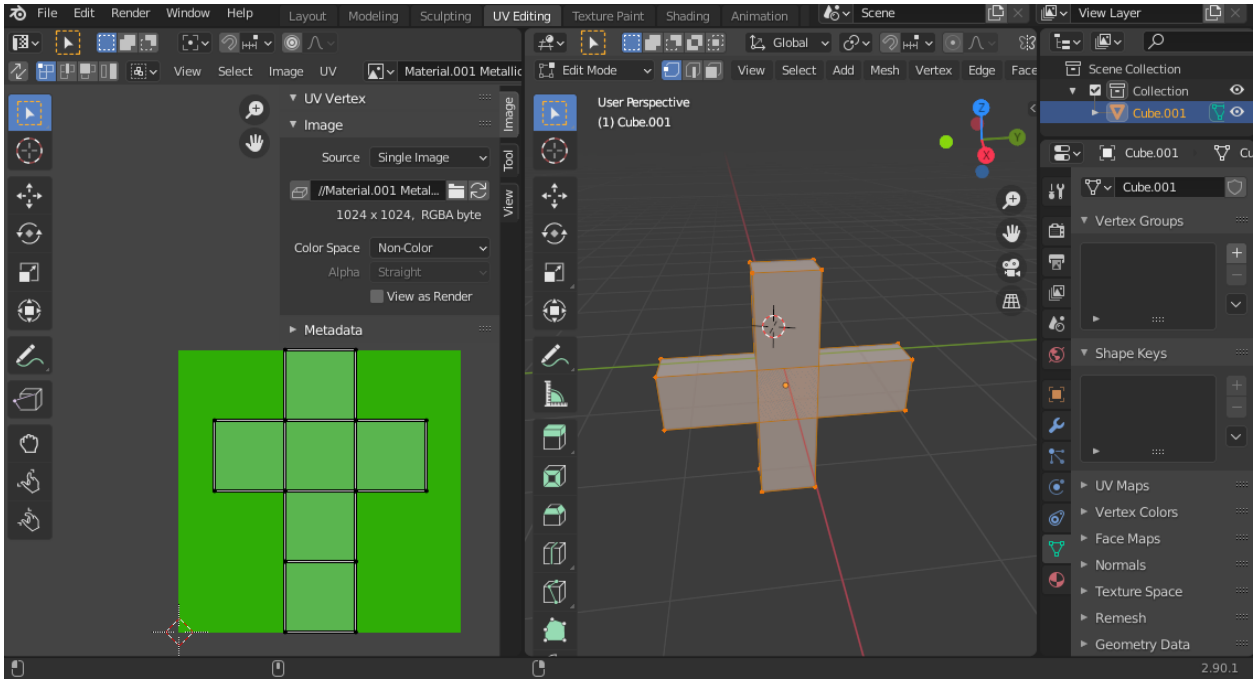


Figure 35 - Blender development environment with texture creation

Various 3D models for the game were developed in the open-source application called Blender.

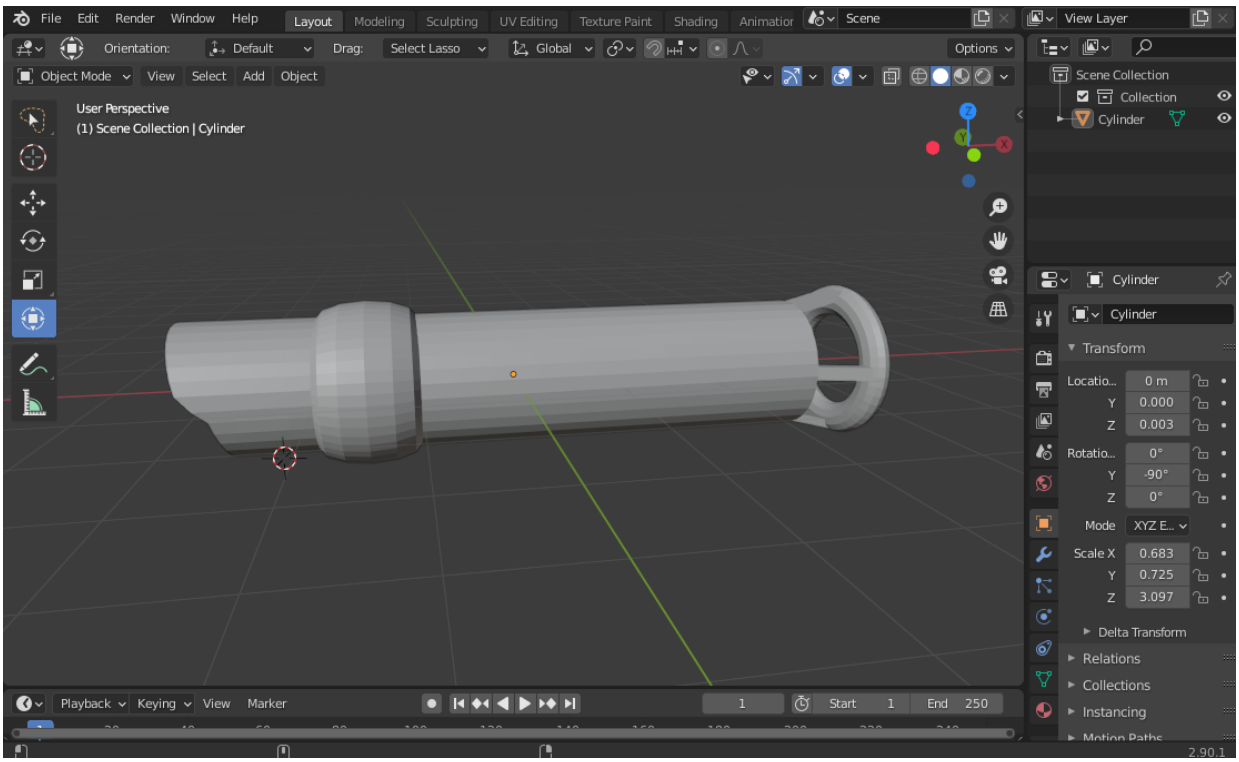


Figure 36 – More 3D model creation for the game in blender

Sprints

The initial files (such as the blank Unity project) and repository for the project was set up prior to the first sprint, ideas had already been discussed about the game and development had a solid direction to go in at this stage.

Sprint 1

Goal

Create the most basic scripts for the game that will control the player camera, and basic movement around the Unity scene.

Item 1

The first part of this sprint was to implement a way to move around the Unity scene. This was achieved with the FPMovement.cs script.

```
Unity Script | 0 references
public class FPMovement : MonoBehaviour
{
    Vector2 pub_rotation = Vector2.zero;
    [Range(0.5f, 10.0f)]
    public float cameraSpeed = 2f;
    [Range(0.5f, 100.0f)]
    public float moveSpeed = 2f;

    private bool noLook = false;
    Unity Message | 0 references
    void Start()
    {
        Cursor.visible = false;
    }
    Unity Message | 0 references
    void Update()
    {
        MouseLook();
        EventKeys();
    }

    1 reference
    void EventKeys()
    {
        if (Input.GetKeyDown(KeyCode.W))
            StartCoroutine(nameof(Forward));

        if (Input.GetKeyDown(KeyCode.S))
            StartCoroutine(nameof(Back));

        if (Input.GetKeyDown(KeyCode.A))
            StartCoroutine(nameof(Left));

        if (Input.GetKeyDown(KeyCode.D))
            StartCoroutine(nameof(Right));

        if (Input.GetKeyDown(KeyCode.Space))
            StartCoroutine(nameof(Space));

        if (Input.GetKeyDown(KeyCode.LeftControl))
            StartCoroutine(nameof(Ctrl));

        if (Input.GetKeyDown(KeyCode.LeftShift))
```

Figure 37 - Basic initial movement script

The initial movement class was very primitive, it used coroutines bound to hardcoded values.

```
    1 reference
IEnumerator Forward()
{
    for (;;)
    {
        if (Input.GetKeyUp(KeyCode.W))
            break;
        transform.position += transform.forward * moveSpeed * Time.deltaTime;
        yield return null;
    }
}
    1 reference
IEnumerator Back()
{
    for (;;)
    {
        if (Input.GetKeyUp(KeyCode.S))
            break;
        transform.position -= transform.forward * moveSpeed * Time.deltaTime;
        yield return null;
    }
}
    1 reference
IEnumerator Left()
{
    for (;;)
    {
        if (Input.GetKeyUp(KeyCode.A))
            break;
        Vector3 forward = new Vector3(transform.forward.x, transform.forward.y, transform.forward.z);
        Vector3 up = new Vector3(0.0f, 1.0f, 0.0f);
        Vector3 left = Vector3.Cross(forward.normalized, up.normalized);
        transform.position += left * moveSpeed * Time.deltaTime;
        yield return null;
    }
}
```

Figure 38 - Original input implementation

This script was attached to the camera, it moved the position around in the Unity scene, it was not affected by gravity. The implementation with coroutines is unusual for its purpose, as I improved my skill with Unity, I would figure out how to better work with Unity's physics classes and methods. However, this script worked well for testing the scene, it could be repurposed as a spectator camera in the future, as it ignores the Unity physics engine.

Item 2

The second part of this sprint was getting a working way to look around the world. Originally, this was placed in the same script as the movement.

```
    1 reference
118 ····void·LShift()
119 ····{
120 ····    moveSpeed·+=·moveSpeed;
121 ····}
    1 reference
122 ····void·MouseLook()
123 ····{
124 ····    //Pause·mouselook·key
125 ····    if·(Input.GetKeyDown(KeyCode.P))
126 ····        noLook·=·!noLook;
127 
128 ····    if·(!noLook)
129 ····    {
130 ····        Cursor.lockState·=·CursorLockMode.Locked;
131 ····        pub_rotation.y·+=·Input.GetAxis("Mouse·X");
132 ····        pub_rotation.x·+=·Input.GetAxis("Mouse·Y")·-·(Input.GetAxis("Mouse·Y")·*·2);
133 ····        transform.eulerAngles·=·pub_rotation·*·cameraSpeed;
134 ····    }
135 ····    else
136 ····        Cursor.lockState·=·CursorLockMode.None;
137 ····}
138 }
```

Figure 39 - Looking around with mouse.

This is using the Unity Input class to get data from the mouse attached to the computer, it orients the camera in the Unity scene that the player sees through to represent changes in mouse position. There was a key to pause mouse look input so that you could easy exit the scene in Unity.

Sprint 2

Goal

Improve the existing scripts to be much more developed and functional for a player in the world. Create a movement script that interacts with the Unity physics engine and allows the player to move and Jump around.

Item 1

Improvements to the script which allows the player to look around using the mouse.

```
7   public enum RotationAxis
8   {
9       MouseX = 1,
10      MouseY = 2
11  }
12
13  public RotationAxis Axis = RotationAxis.MouseX;
14
15  public float minimumVerticalLook = -90.0f;
16  public float maximumVerticalLook = 90.0f;
17
18  public float horizonitalSensitivity = 2.0f;
19  public float verticalSensitivity = 2.0f;
20
21  public bool lockCursor = true;
22  public bool forceHideCursor = true;
23
24  private float _rotationX = 0.0f;
25
    Unity Message | 0 references
26  void Update()
27  {
28      if (lockCursor)
29          Cursor.lockState = CursorLockMode.Locked;
30      else
31          Cursor.lockState = CursorLockMode.Confined;
32      if (forceHideCursor)
33          Cursor.visible = false;
34
35      //For X
36      if (Axis == RotationAxis.MouseX)
37          transform.Rotate(0, Input.GetAxisRaw("Mouse X") * horizonitalSensitivity, 0);
38      //For Y
39      else if (Axis == RotationAxis.MouseY)
40      {
41          _rotationX -= Input.GetAxisRaw("Mouse Y") * verticalSensitivity;
42          _rotationX = Mathf.Clamp(_rotationX, minimumVerticalLook, maximumVerticalLook);
43
44          float rotationY = transform.localEulerAngles.y;
45          transform.localEulerAngles = new Vector3(_rotationX, rotationY, 0);
46      }
47  }
48  }
```

Figure 40 - Improved independent mouse movement script

The script for controlling mouse movement was rewritten and placed in its own class later during this sprint, this version supported clamping angles so that the player could not look too far down or too far up. Sensitivity was also now editable easily in the Unity editor.

```
    [Range(0.0f, 100.0f)]
    public float jumpPower = 4.0f;
    public float downForce = -10.0f;
    public bool accelerateDownForce = false;

    private Vector3 playerVelocity;
    private bool playerGrounded;
    private CharacterController _playerController;

    Unity Message | 0 references
    void Start()
    {
        _playerController = GetComponent<CharacterController>();
    }

    Unity Message | 0 references
    void Update()
    {
        playerGrounded = _playerController.isGrounded;
        if (playerGrounded && playerVelocity.y < 0)
        {
            playerVelocity.y = 0.0f;
        }

        float deltaX = Input.GetAxisRaw("Horizontal") * speed;
        float deltaZ = Input.GetAxisRaw("Vertical") * speed;
        Vector3 pMove = new Vector3(deltaX, 0.0f, deltaZ);

        pMove = transform.TransformDirection(pMove);
        _playerController.Move(pMove * Time.deltaTime);

        if (accelerateDownForce && !_playerController.isGrounded)
            playerVelocity.y += downForce * Time.deltaTime;
        else if (accelerateDownForce && _playerController.isGrounded)
            playerVelocity.y = 0.0f;

        if (playerGrounded && Input.GetButton("Jump"))
        {
            playerVelocity.y += Mathf.Sqrt(jumpPower * -3.0f * downForce);
        }
        playerVelocity.y += downForce * Time.deltaTime;
        _playerController.Move(playerVelocity * Time.deltaTime);
    }
}
```

Figure 41 - Updated movement script

A lot of iterations went into improving the movement script, by the end of the first sprint this was the version that had been developed. There were still some issues present at this stage, but the player could now move around and interact with Unity's physics, including gravity, drag and acceleration.

The player now how the ability to jump around the scene, it is important to note that at this stage of the game's development, the movement script was using a CharacterController. A CharacterController is a class provided by Unity for moving a player around the scene. However, the player cannot be easily affected by the physics engine while using a CharacterController. This design proved to be a mistake in the future, as all this code would have to be rewritten to work better with physics since this game is so focused on physics and using a CharacterController was far too slow and tedious with physics.

Sprint 3

Goal

Begin coding the grapple gun script. Create fundamental assets and models for the game.

Item 1

First part of this sprint was spent creating the initial code for the grapple gun script.

```
{
    public float accelerationSpeed = 40.0f;
    public Transform grapplePointer;
    public Camera playerCam;
    public CharacterController player;


    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            if (Physics.Raycast(playerCam.transform.position, playerCam.transform.forward, out RaycastHit castHit))
            {
                //Grapple attempt was valid
                grapplePointer.position = castHit.point;
                Vector3 grappleDirection = (castHit.point - transform.position).normalized;
                player.Move(grappleDirection * accelerationSpeed * Time.deltaTime);
            }
        }
    }
}
```

Figure 42 - Initial grapple script

This script did not yet properly cause the player to accelerate towards the position they tried grapple to, but it laid the groundwork for it. I figured out a way to have the player press a button and accelerate towards the place in the world they were looking at (if it was a valid grapple target), though the model for the grapple gun and the line of the grapple hook had not been created. It was here that I realised there was an issue with using a CharacterController, as finding ways to make the grapple move the player proved difficult.

Item 2

Second part of this script was spent developing assets in blender to be used in the game.

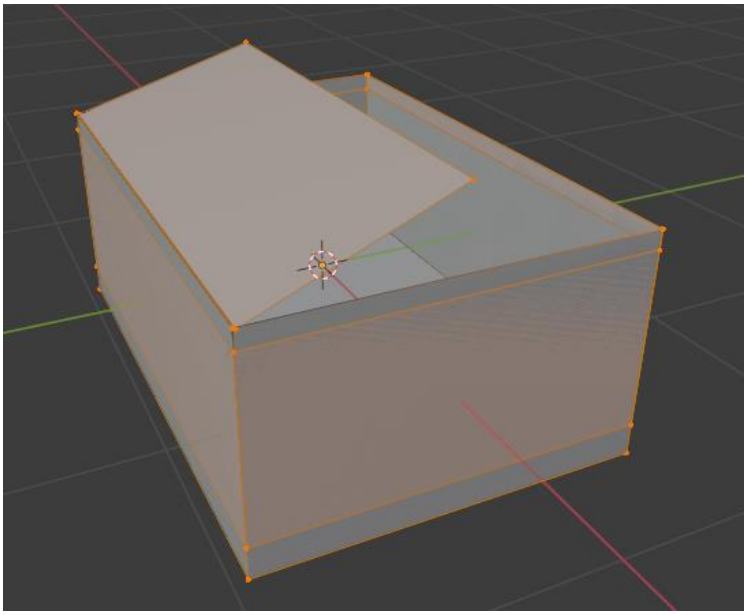


Figure 43 - Ammunition box used to increase players ammo.

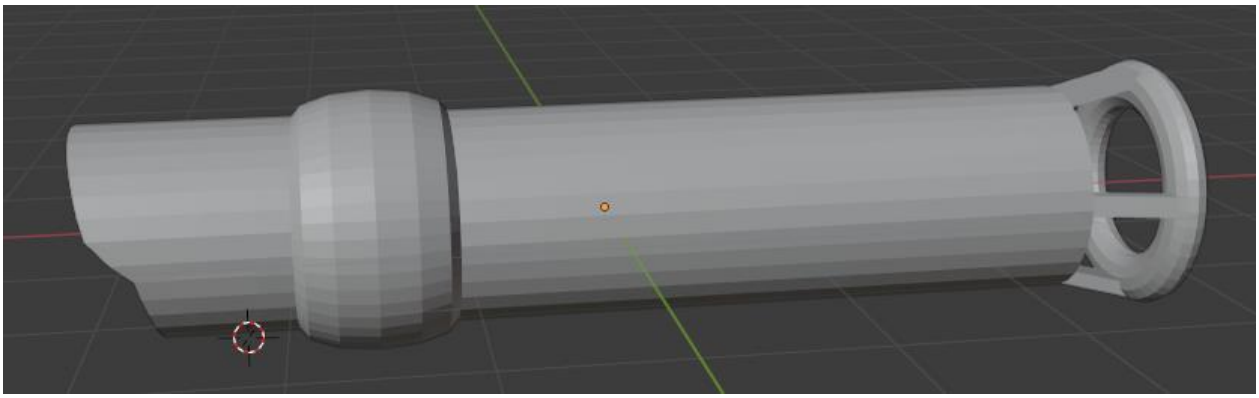


Figure 44 - A rocket launcher used to attack enemies and propel self.

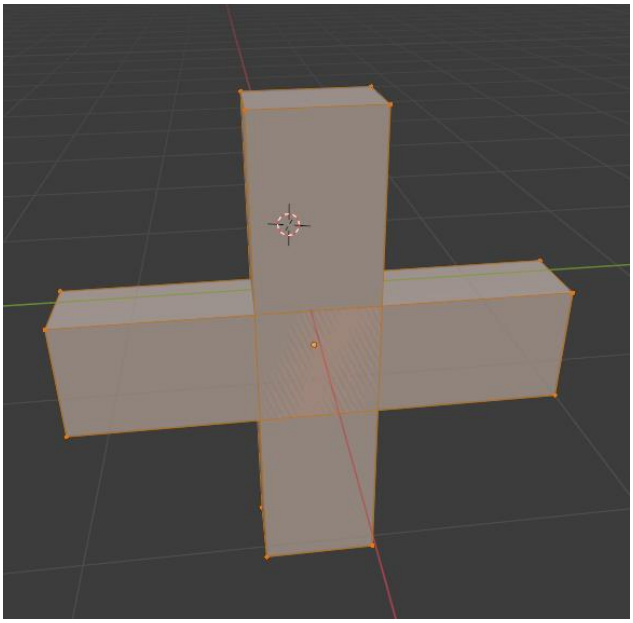


Figure 45 - A health kit used to increase players health.

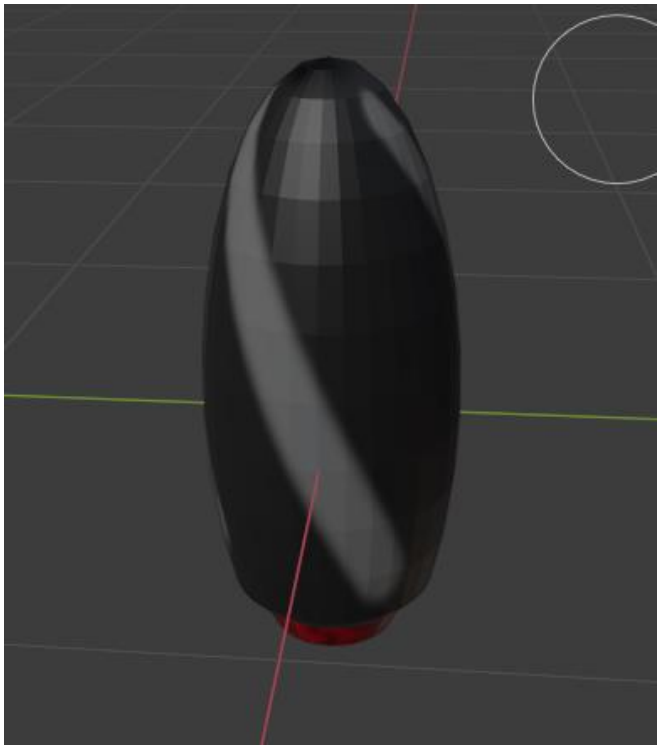


Figure 46 - A projectile that is sent out of the rocket launcher when fired.

These assets took some time to develop, as Blender has a steep learning curve. Unity allows easy importing of these models into the game. They were then slightly modified inside unity to better fit the world. Textures were then added to the models, which were also developed inside Blender.

Sprint 4

Goal

Develop a level for the game. Convert the CharacterController movement scripts to a solution that works better with physics. Improve lighting and setting. Finally, implement a health/ammunition system for the player. Create an AI for bots in the game.

Item 1

```
    Unity Message | 0 references
25  void Start()
26  {
27      originalMoveSpeed = moveSpeed;
28      _playerBody = GetComponent<Rigidbody>();
29      _playerCollider = GetComponent<CapsuleCollider>();
30  }
31
    Unity Message | 0 references
32  void FixedUpdate()
33  {
34      Movement();
35  }
36
    Unity Message | 0 references
37  void Update()
38  {
39      inputManager.UpdatePlayerValues();
40  }
41
    1 reference
42  private bool IsGrounded()
43  {
44      //This will draw an invisible ray downwards, if it hits an object, the player is grounded.
45      return Physics.Raycast(transform.position, Vector3.down,
46          _playerCollider.bounds.extents.y + groundedLeniancy);
47  }
48
    1 reference
49  private void Movement()
50  {
51      if (IsGrounded())
52      {
53          if (inputManager.GetJumping())
54          {
55              //Player jump by adding vertical force, accounting for player mass.
56              _playerBody.AddForce(Vector3.up * jumpHeight, ForceMode.Impulse);
57          }
58      }
59
60      _playerBody.AddForce(orientation.transform.forward
61          * inputManager.GetVertical() * moveSpeed * Time.deltaTime);
62
63      _playerBody.AddForce(orientation.transform.right
64          * inputManager.GetHorizontal() * moveSpeed * Time.deltaTime);
65
66  }
67
```

Figure 47 - Rigid movement script for player movement.

This is how the script appeared at the end of the sprint. During the sprint, many bugs and issues were encountered with the movement script. In this final version, movement is caused by the Unity engine adding physical force to the player's object. In a sense, the player is pressing a key to cause the Unity physics engine to push them, rather than themselves moving through the world. Creating this solution took

some rethinking of how best to implement a character that works seamlessly with physics.

```
        float hMove = Input.GetAxis("Horizontal") * moveSpeed;
        float vMove = Input.GetAxis("Vertical") * moveSpeed;
        hMove *= Time.deltaTime;
        vMove *= Time.deltaTime;

        transform.Translate(hMove, 0, vMove);
```

Figure 48 - Old implementation of rigid movement, with bug.

This is a snippet from an older implementation of the code, this used the method:

```
transform.Translate(hMove, 0, vMove);
```

This crucial line generated a bug that took some debugging to figure out, the Translate method just changes the location of the player literally, without taking the world into account. If the player was going slow, this did not cause any issues. However, if the player were moving fast, they could pass through walls and colliders and fall out of the world.

The cause of this bug was that the value supplied to the Translate method was big enough when going at a high speed that it could essentially teleport the player beyond the colliders in the world, bypassing the colliders collision checks. Originally, I had thought the issue was to do with the colliders being badly implemented, it took some research of the Unity documentation to understand the issue.

Item 2

A level for the game was developed. Researched, found, and imported free Unity store assets for the game:

https://assetstore.unity.com/packages/audio/sound-fx/free-sound-effects-pack-155776

https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-particle-pack-127325

https://assetstore.unity.com/packages/2d/textures-materials/gridbox-prototype-materials-129127

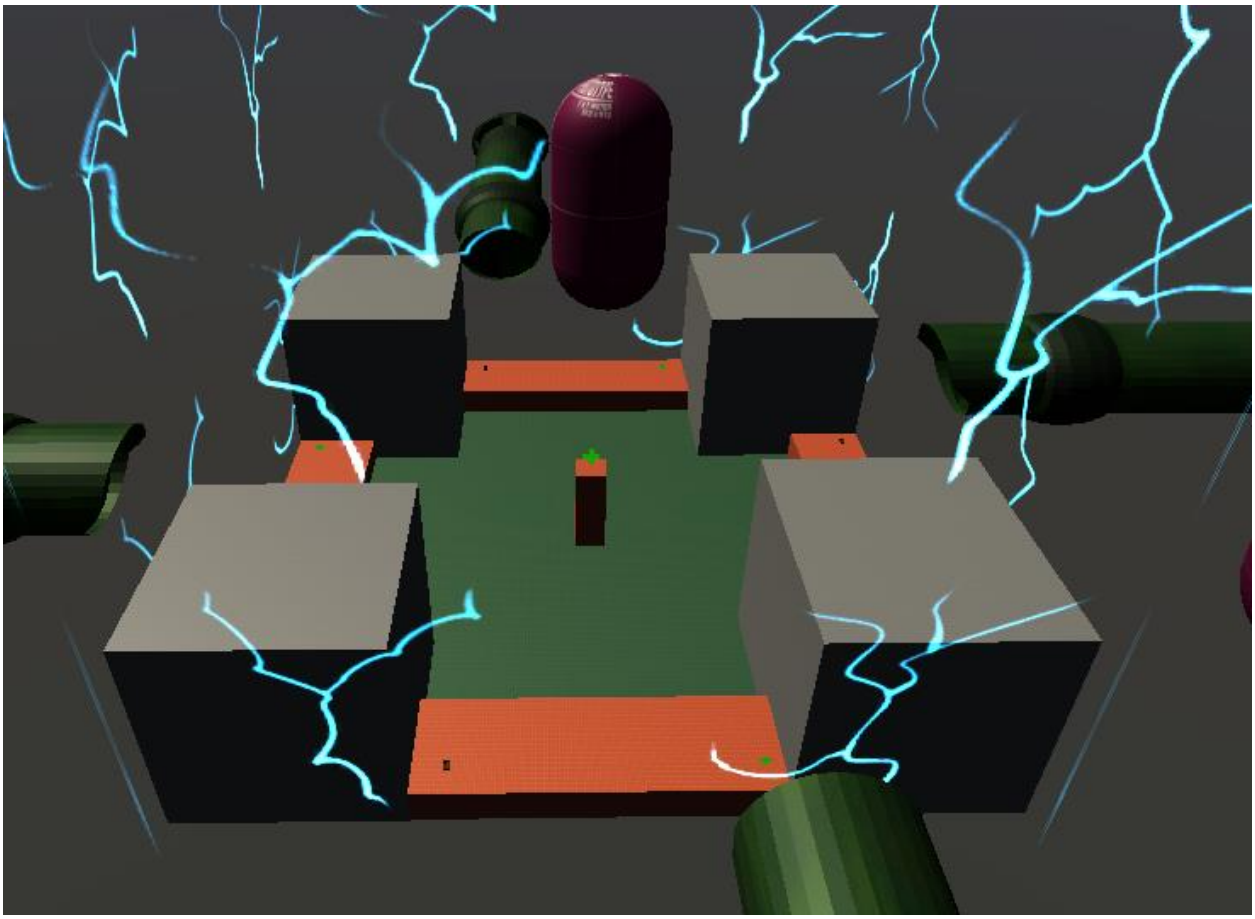
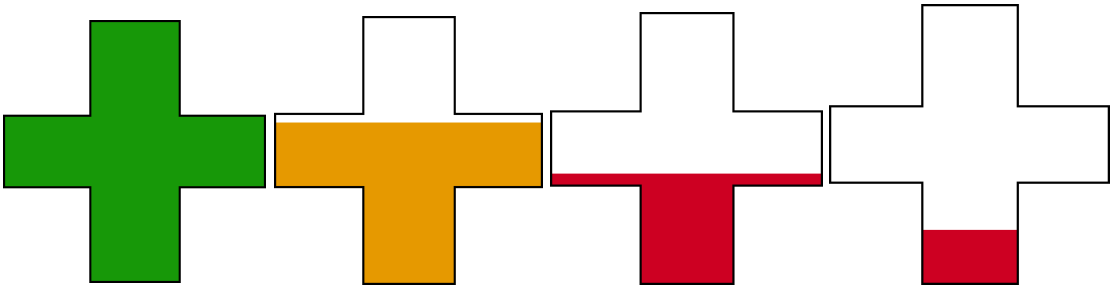


Figure 49 - Level made for the game.

This level was added to the game to specifically focus on player vs player combat with rocket launchers. There are health kits and ammunition kits that the player can pick up to gain health placed around the map. The large cubes were created to give players skilled in rocket-jumping an edge over other players. Scripts that made the ammunition boxes and health kits rotate were created, audio was sourced for them to be played when picked up.

Item 3

Prefabs were made for the player, as well as UI elements such as a crosshair and health information.



Various graphics representing the players health, made in photoshop.

```
Unity Message | 0 references
50      private void Update()
51      {
52          if (health > 80)
53          {
54              healthImage.texture = healthTextures[0];
55          }else if (health > 60)
56          {
57              healthImage.texture = healthTextures[1];
58          }else if (health > 40)
59          {
60              healthImage.texture = healthTextures[2];
61          }else
62          {
63              healthImage.texture = healthTextures[3];
64          }
65
66          ammoText.text = ammo.ToString();
67
68
69          //Triggered on death
70          if (awaitingRespawn)
71          {
72              respawnTimer -= Time.deltaTime;
73              if (respawnTimer > 0.2f)
74              {
75                  respawnText.text = "Respawn in (" + (int)(respawnTimer) + ")";
76              }else
77              {
78                  Respawn();
79              }
80          }
81      }
82
```

Figure 50 - Snippet of code that controls health graphics.

This code controls which graphic is displayed to the player on the UI. It is a snippet from a much larger script “PlayerStats.cs”, which is too large to screenshot. This class was created to contain a lot of information about the player and ways for other scripts to change player values such as ammunition and health.

```
Unity Script | 0 references
public class HealthPack : MonoBehaviour
{
    public int respawnDelay = 10;
    public float spinSpeed = 200.0f;
    public int regenAmount = 100;
    public AudioClip[] packAudio;

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        //Spinning animation
        transform.Rotate(0, 0, spinSpeed * Time.deltaTime, Space.Self);
    }

    //Detecting if a player touches healthpack
    Unity Message | 0 references
    private void OnTriggerEnter(Collider collision)
    {
        if (collision.gameObject.layer == 9)
        {
            //Increases the player's health that touches the health pack
            collision.gameObject.GetComponent<PlayerStats>().IncreaseHealth(regenAmount);

            StartCoroutine(RespawnHealthPack());

        }
    }

    1 reference
    IEnumerator RespawnHealthPack()
    {
        //Play regenerate sound
        AudioSource.PlayClipAtPoint(packAudio[1], transform.position);

        //Hide under world
        transform.position = new Vector3(
            transform.position.x
            , transform.position.y - 1000
            , transform.position.z);

        //Wait for two seconds efficiently
        yield return new WaitForSeconds(respawnDelay);

        //Reset to position
        transform.position = new Vector3(
            transform.position.x
            , transform.position.y + 1000
            , transform.position.z);

        //Play respawn sound
        AudioSource.PlayClipAtPoint(packAudio[0], transform.position);
    }
}
```

No issues found

Figure 51 - Health pack code

The code for the health packs in the level is shown above. The player object is given its own layer in Unity, which made it easy to reference from other scripts. A coroutine is started which respawns the health pack after a given amount of time. These scripts are designed to be modular and reusable, avoiding hardcoded values, so that they could be potentially reused or easily modified in the future. The ammunition script was also created and is very similar in code.

Item 4

Additional logic inside of the PlayerStats.cs script for handling the player lifecycle was developed.

```
128     //Code triggered on death
        1 reference
129     private void Die()
130     {
131         gameObject.transform.position = new Vector3(0, -3000, 0);
132         deathPanel.SetActive(true);
133         uiCanvas.SetActive(false);
134
135         //Stops select scripts on player
136         foreach (MonoBehaviour mb in disableOnRespawnAwait)
137         {
138             mb.enabled = false;
139         }
140         Cursor.visible = true;
141         Cursor.lockState = CursorLockMode.None;
142
143         awaitingRespawn = true;
144     }
145
        1 reference
146     private void Respawn()
147     {
148         //Resets values
149         this.health = cachedHealth;
150         this.ammo = cachedAmmo;
151         awaitingRespawn = false;
152         uiCanvas.SetActive(true);
153         deathPanel.SetActive(false);
154
155         foreach (MonoBehaviour mb in disableOnRespawnAwait)
156         {
157             mb.enabled = true;
158         }
159         Cursor.visible = false;
160         respawnTimer = respawnCachedTimer;
161
162         gameObject.transform.position = spawnHandler.GetRandomGenericSpawn().transform.position;
163     }
164 }
```

Figure 52 – Snippet of respawn / death code logic.

There is additional code inside this class which was developed alongside the health/ammo code to support player death and respawn mechanics within the game. This is targeting the combat game mode in the game, where the player can die and respawn.

```
        Unity Message | 0 references
60      void Start()
61      {
62          spawnLocations = GameObject.FindGameObjectsWithTag("SpawnPoint");
63
64          if (DEBUG_PLAYERS)...
74
75          //Shuffles array of spawn locations
76          RandomShuffle(spawnLocations);
77
78          //Place all players/bots at spawn points
79          foreach (GameObject player in playersAsList)
80          {
81              GameObject initialSpawnPoint = RandomVacantSpawn(spawnLocations);
82              player.transform.position = initialSpawnPoint.transform.position;
83              player.transform.rotation = initialSpawnPoint.transform.rotation;
84
85          }
86      }
87
88
89      //Returns gameobject to location of vacant spawn
            2 references
90      private GameObject RandomVacantSpawn(GameObject[] rl)
91      {
92          //Keeps track of how many spawns are left available
93          --freeSpawns;
94          return rl[freeSpawns];
95      }
96
```

Figure 52 - Snippet of spawn handler script

This code shown above was developed alongside the death/respawn logic, which chooses a location in an array of locations to pick at random. This is used in the multiplayer and singleplayer to pick a random location on the map to spawn the player. It contains logic to avoid spawning two players at the same location.

Item 5

```
0 references
41      private Vector3 DestinationOffset(Vector3 sd)...
50
51      //Random actions
        1 reference
52      IEnumerator BotAI()
53      {
54          rb.AddForce(Random.Range((0 - offsetAmount), (0 + offsetAmount))
55                      ,0
56                      , Random.Range((0 - offsetAmount), (0 + offsetAmount)));
57
58          //Random jump
59          if ((int)Random.Range(1, 4) == 3)
60          {
61              rb.AddForce(0, jumpForce, 0);
62          }
63
64          //Randomally decide to shoot
65          if ((int)Random.Range(1, 4) == 3)
66          {
67              botLauncher.RequestShoot();
68          }
69
70          yield return new WaitForSeconds(waitForNewDestination);
71          moveAgain = true;
72      }
73  }
74
```

Figure 53 - Bot AI for the game.

I did not want to use a premade AI for this game, instead I decided to try to make my own in C#. The end result is quite good, it is simple, its controlled by random movements, which decides to shoot at random intervals at the player. The bots are certainly not clever, but they can be entertaining to fight.

Sprint 5

Goal

The goal for this sprint was to fix issues such as the messy input system, rocket launcher logic scripts, develop the initial grapple hook scripts, and begin work on a multiplayer mode.

Item 1

```
12 references
public static class StaticInput
{
    private static float horizontal = 0.0f;
    private static float vertical = 0.0f;
    private static bool jumping = false;
    private static bool shooting = false;
    private static bool pause = false;
    private static bool chatDown = false;

    private static bool inputSuspended = false;

    /// <summary>
    /// Updates all the player input values
    /// </summary>
    1 reference
    public static void UpdatePlayerValues()
    {
        if (!inputSuspended)
        {
            horizontal = Input.GetAxisRaw("Horizontal");
            vertical = Input.GetAxisRaw("Vertical");
            jumping = Input.GetButton("Jump");
            shooting = Input.GetButtonDown("Fire1");
        }
    }

    /// <summary>
    /// Updates the horizontal value only
    /// </summary>
    0 references
    public static void UpdateHorizontal()
    {
        if (!inputSuspended)
        {
            horizontal = Input.GetAxisRaw("Horizontal");
        }
    }

    /// <summary>
    /// Updates the vertical value only
    /// </summary>
    0 references
    public static void UpdateVertical()...
```

Figure 54 - Input put into a static class.

A script called StaticInput was developed, as a problem with scripts becoming messy had become a problem. Before this, scripts checked inputs on their own for their own purpose. If you needed to change an input key, you would have to find the exact script that checks for it. To remedy this, all the input checks were moved to one standardised class, that other classes can call or reference.

Item 2

Rocket launcher script development. The rocket launcher in this game is inspired by popular games such as Team Fortress and Quake. It is essentially a bazooka, but with unrealistic cartoonish properties such as being able to propel your character with it, and not needing to reload.

```
    Unity Message | 0 references
23  void Start()
24  {
25      //Gets a reference to the block inside the launcher
26      launchBlock = gameObject.GetComponent<Rigidbody>();
27      launchBlock.transform.parent = transform.parent.transform;
28
29      //Gets a reference to the player
30      _player = gameObject.transform.root.gameObject;
31      _playerStats = _player.GetComponent<PlayerStats>();
32
33      canLaunch = true;
34  }
35
    Unity Message | 0 references
36  void Update()
37  {
38      if (StaticInput.GetShooting() && canLaunch)
39      {
40          //Embedded for efficiency
41          int _pAmmo = _playerStats.GetAmmo();
42          if (_pAmmo > 0)
43          {
44              FireProjectile();
45
46              //Sets the ammo to one less
47              _playerStats.SetAmmo(--_pAmmo);
48              StartCoroutine(PauseFiring());
49          }
50
51      }
52
53      //Keeps the firing particle effects on the launcher regardless of speed
54      if (launchObject != null)
55      {
56          launchObject.transform.position = launchBlock.transform.position;
57      }
58  }
59
    1 reference
60  void FireProjectile()
61  {
62      //Adjust Y position of projectile launch if necessary
63      Vector3 adjustedPosition = transform.position;
64      adjustedPosition.y = adjustSpawnPositionY;
```

Figure 55 - Snippet from projectile launching code.

This code is part of the rocket launcher prefab scripts, from the script LaunchProjectile.cs. The ammo is decreased here, as well as the logic to prevent the player firing the rocket launcher too quickly, and plays sounds on firing, as well as particle effects. The script is using a tiny invisible cube at the tip of the rocket launcher to shoot from called launchObject, this is the point where the sound effect and particle effects are played from. On line 38 in the screenshot, the new StaticInput script can be seen being used.

```
1 reference
void FireProjectile()
{
    //Adjust Y position of projectile launch if necessary
    Vector3 adjustedPosition = transform.position;
    adjustedPosition.y -= adjustSpawnPositionY;

    //Launch particle effect
    launchObject = (GameObject)Instantiate(launchEffect, transform.position, transform.rotation);

    //Plays audio for launch
    AudioSource.PlayClipAtPoint(soundEffects[0], transform.position, GlobalAudioReference.instance.GetEffectsVolume());

    //Creates rocket at top of launcher
    GameObject projectile = Instantiate(projectilePrefab, adjustedPosition, transform.rotation);



    //Gives the rocket an ID to parent so that it doesn't collide with owner of rocket
    projectile.GetComponent<Projectile>()
        .SetParentByID(
            PlayerID.GetIDByGameObject(transform.parent.root.gameObject)
        );

    //Gives rocket momentum
    Rigidbody rb = projectile.GetComponent<Rigidbody>();
    rb.AddForce(transform.forward * projectileSpeed, ForceMode.VelocityChange);

    //Cleanup for efficiency
    Destroy(launchObject, 2);
}
```

Figure 57 - Fire projectile logic

This code is what gives the projectile momentum and launches it away from the rocket launcher. On line 67 in the screenshot, the code creates particle effects at the top of the rocket launcher. On line 70, the code plays audio from the same position.

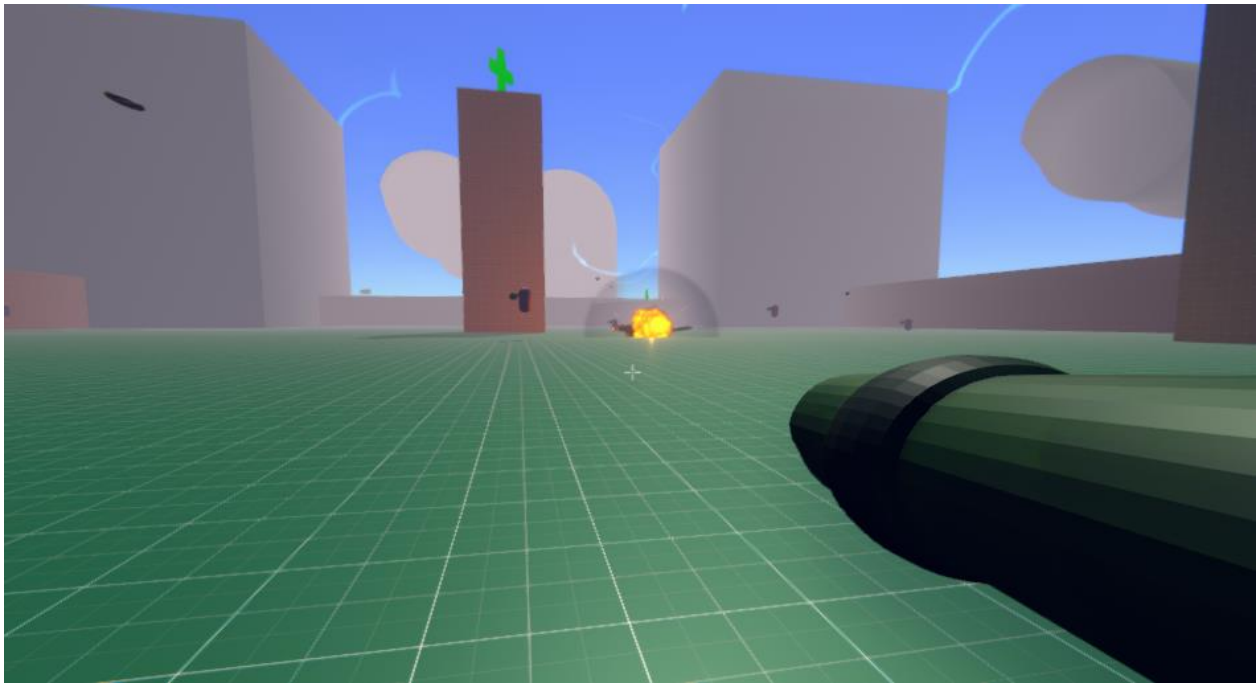


Figure 56 - The explosion effects after a projectile hit the ground.

Item 3

Grapple hook development. After having a failed attempt at making a grapple hook script from scratch, I investigated other solutions. I found a YouTube channel with a fantastic tutorial on the subject. The logic for the grapple hook is derived from that tutorial.

```
2 references
private void Grapple(bool enabled)
{
    if (enabled)
    {
        RaycastHit hitLocation;
        if (Physics.Raycast(gunCamera.position, gunCamera.forward, out hitLocation, maxGrappleDistance, grapplableSurface))
        {
            grapplePosition = hitLocation.point;

            springJoint = player.gameObject.AddComponent<SpringJoint>();
            springJoint.autoConfigureConnectedAnchor = false;
            springJoint.connectedAnchor = grapplePosition;

            float distanceFromPoint = Vector3.Distance(player.position, grapplePosition);
            springJoint.maxDistance = distanceFromPoint * 0.8f;
            springJoint.minDistance = distanceFromPoint * 0.25f;

            springJoint.spring = 4.5f;
            springJoint.damper = 2.0f;
            springJoint.massScale = 400.5f;

            lineRenderer.positionCount = 2;
        }
    }
    else
    {
        lineRenderer.positionCount = 0;
        Destroy(springJoint);
    }
}

1 reference
private void DrawGrappleRope()
{
    //Prevent drawing when not grappling
    if (!springJoint)
        return;

    lineRenderer.SetPosition(0, gunBarrel.position);
    lineRenderer.SetPosition(1, grapplePosition);
}
}
```

Figure 58 - Snippet of grapple hook code

This code was written from following an online tutorial - https://www.youtube.com/watch?v=Xgh4v1w5DxU. Following this helpful tutorial was key to getting the grapple hook working. There are various changes to the script from the tutorial version to make it fit the game.

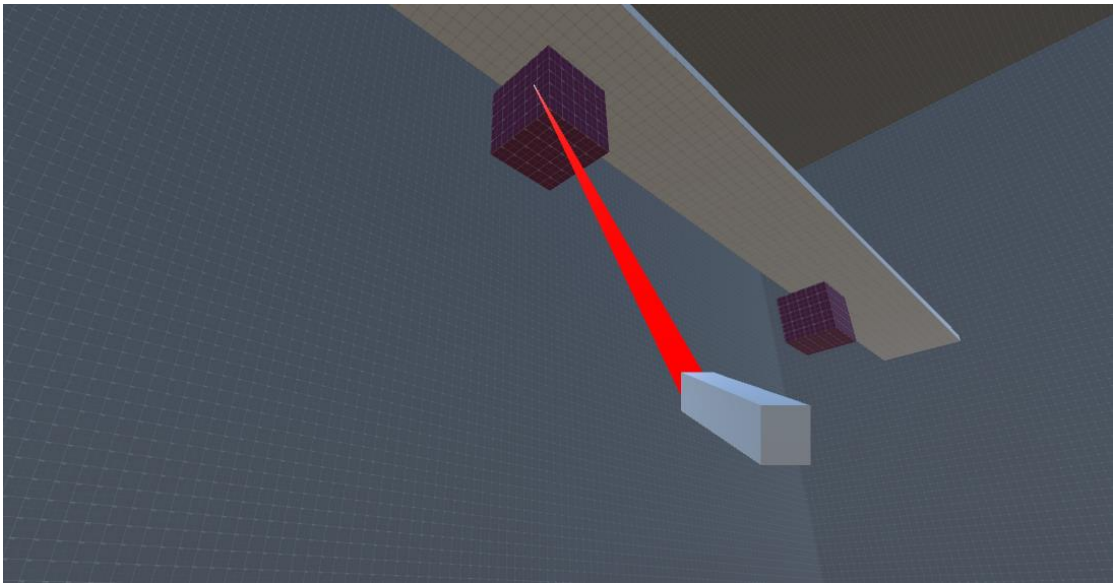


Figure 59 – Initial prototype grapple without model.

Shown above is an example of the player in the game swinging off an elevated cube to gain speed and momentum. This screenshot is from early development, before the game's appearances were improved.

Item 4

The beginning of the multiplayer code was the last part of this sprint. This is by far the most ambitious undertaking of the entire project. The software for multiplayer in this game is written from scratch in C#, from following an online tutorial. The code is an entire project on its own and was more technically challenging to create than the entire rest of my portion of the game. The tutorial for the server code is a set of videos which starts at https://www.youtube.com/watch?v=uh8XaC0Y5MA&list=PLXkn83W0QkfnqsK8I0RAz5AbUxfg3bOQ5 . The first 5 video tutorials were followed in this series, after which the tutorial differed from the demands of our game, and was developed independently thereon.

```
29 references
class Server
{
    7 references
    public static int MaxPlayers { get; private set; }
    4 references
    public static int Port { get; private set; }
    public static Dictionary<int, Client> clients = new Dictionary<int, Client>();
    public delegate void PacketHandler(int _fromClient, Packet _packet);
    public static Dictionary<int, PacketHandler> packetHandlers;

    private static TcpListener tcpListener;
    private static UdpClient udpListener;
    1 reference
    public static void Start(int _maxPlayers, int _port)
    {
        MaxPlayers = _maxPlayers;
        Port = _port;

        //Server information formatted for the console
        Console.WriteLine(
            $"SERVER INFORMATION:\n" +
            $"_____________________________\n\n" +
            $"Version: \t{Constants.SERVER_VERSION}\n" +
            $"Max players: \t{Constants.MAX_PLAYERS}\n" +
            $"TPS: \t\t{Constants.TICKS_PER_SEC}\n" +
            $"_____________________________\n"
        );

        InitializeServerData();

        tcpListener = new TcpListener(IPAddress.Any, Port);
        tcpListener.Start();
        tcpListener.BeginAcceptTcpClient(TCPConnectCallback, null);

        udpListener = new UdpClient(Port);
        udpListener.BeginReceive(UDPReceiveCallback, null);

        Console.WriteLine($"Server started on {Port}.");
    }

    2 references
    private static void TCPConnectCallback(IAsyncResult _result)
    {
```

Figure 60 - Small snippet of server start-up code

The server was designed to be similarly setup to that of the very popular game “Minecraft”. The idea is that a user with some experience with computers can download an executable server file and host a server on their own computer. The servers executable file will host the games logic on the computer it is run on, provided there is admin rights, and the IP of the network is port forwarded (or the joining clients are on the same network).

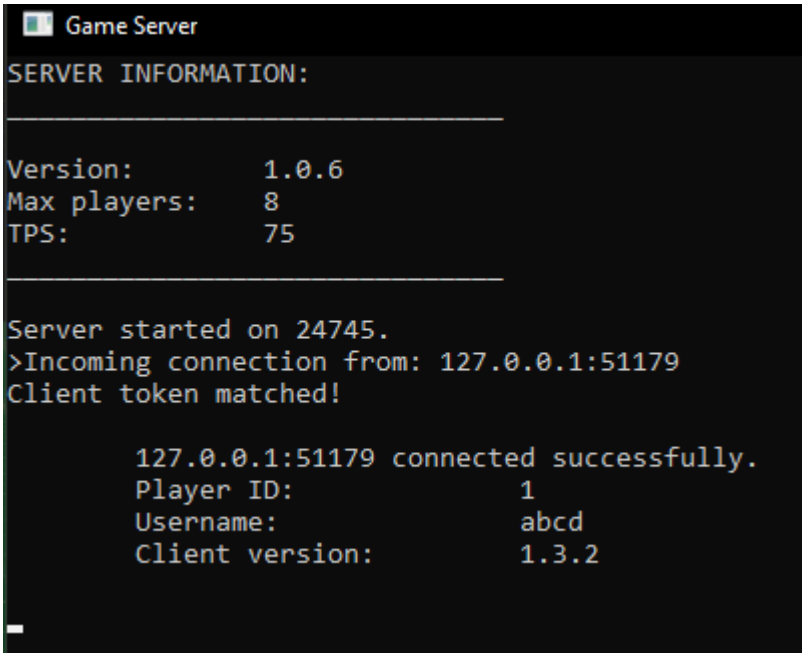


Figure 61 - Server admin console showing a client connected.

Screenshot above shows a later version of the server with a client connected. This console gives information about the game state and of the players that are connected. The screenshot was taken of a locally hosted server.

Early versions of the server, such as the one created in Sprint 5, were unstable. It was not until a later sprint that the server became functional and stable.

Sprint 6

Goal

Develop multiplayer to the point where two players can see each other in-game. Transfer information about player positions and projectile firing across the network. Create a server diagnostics branch to help with debugging server code. Create a multiplayer scene and assets.

Item 1

```
107         //Tells other clients that a player has disconnected so they can cleanup the prefab and dictionary
            1 reference
108         public static void PlayerDisconnected(int _playerID)
109         {
110             using (Packet _packet = new Packet((int)ServerPackets.playerDisconnected))
111             {
112                 _packet.Write(_playerID);
113
114                 SendTCPDataToAll(_packet);
115             }
116         }
117
118         //My methods for sending client computed data to other clients below
119         //Sends the players location to clients
            1 reference
120         public static void PlayerLocation(int _playerID, Vector3 _location)
121         {
122             using (Packet _packet = new Packet((int)ServerPackets.playerPosition))
123             {
124                 _packet.Write(_playerID);
125                 _packet.Write(_location);
126
127                 SendUDPDataToAll(_playerID, _packet);
128             }
129         }
130
131         //Sends the players rotation to clients
            1 reference
132         public static void PlayerRotation(int _playerID, Quaternion _rotation)
133         {
134             using (Packet _packet = new Packet((int)ServerPackets.playerRotation))
135             {
136                 _packet.Write(_playerID);
137                 _packet.Write(_rotation);
138
139                 SendUDPDataToAll(_playerID, _packet);
140             }
141         }
```

Figure 62 - Snippet of server code which sends player data to clients.

The snippet above shows the code that sends position data on to other clients. The SeverPackets Enum class is identical on the client and on the server, they are used to tell what type of packet it is. The packets are then decoded accordingly and sent on to the other clients.

This type of server requires minimal server computation, essentially it is mailing the result of computations of clients to other clients and leaving it up to the client to interpret the data. This makes it inexpensive to host, however if it were a game that was realised into the public, it would have disadvantages such as making hacking much easier.

```
102
103         //This will handle the trusted client data and skip computation of player input on the server
            1 reference
104         public static void ReceivePlayerData(int _fromClient, Packet _packet)
105         {
106             Vector3 _playerLocation = _packet.ReadVector3();
107             Quaternion _playerRotation = _packet.ReadQuaternion();
108
109             //Trusting the data and sending it on to other clients
110             ServerSend.PlayerLocation(_fromClient, _playerLocation);
111             ServerSend.PlayerRotation(_fromClient, _playerRotation);
112         }
113
            1 reference
114         public static void ReceiveProjectileData(int _fromClient, Packet _packet)
115         {
116             Vector3 _projectileLocation = _packet.ReadVector3();
117             Quaternion _projectileRotation = _packet.ReadQuaternion();
118
119             ServerSend.ProjectileData(_fromClient, _projectileLocation, _projectileRotation);
120         }
121
```

Figure 63 - Snippet from SeverHandle.cs - handling Realtime client information.

Above is a snippet of the receiving code for the server. The idea of the server is to take information from the clients, then send the data on to the other connected clients, and have them reconstruct the information in their game. On line 114 of the image, the ReceiveProjectileData function for example, reads a Vector3 and a Quaternion. This data represents the position and rotation in 3D space of the projectile fired by a client. The server then sends this data on to all other connected clients, the rocket is then recreated on the other client's computer with that information.

```
            1 reference
81      public void CreateProjectile(int _id, Vector3 _location, Quaternion _rotation)
82      {
83          //Debug.Log($"CreateProjectile was called! With data {_id} {_location} {_rotation}");
84          GameObject projectile = Instantiate(projectilePrefab, _location, _rotation);
85
86          //Gives rocket momentum
87          Rigidbody rb = projectile.GetComponent<Rigidbody>();
88          rb.AddForce(projectile.transform.forward * MultiplayerLaunchProjectile.projectileSpeed, ForceMode.VelocityChange);
89      }
90
```

Figure 64 - Create projectile code snippet from client.

Above is code from the client, which creates a projectile in the game and gives it momentum. Also seen in this code snippet on line 88 is the class MultiplayerLaunchProjectile being referenced. Most of the singleplayer classes had to be recreated to support multiplayer, so the "Multiplayer" prefix is used for the multiplayer version of a script that exists for singleplayer. These classes are modified to support networked values and conditions.

Item 2

The GitHub diagnostic branch was made, a separate branch of the server. This diagnostic branch was crucial during the early development of the server. There was a lot of bugs and issues with the server at the beginning, the diagnostics branch runs at a far slower rate, and logs information as it goes, making it much easier to determine the source of bugs.

Debugging networked software proved to be especially difficult since there are so many moving parts, and the cause of the issue could be on either the client or the server. The server typically runs at 75 TPS (Ticks Per Second) which means it can send and receive 75 messages per second, however the diagnostics branch runs at just 1 TPS, so that data transferring to and from the server can be monitored in real time.

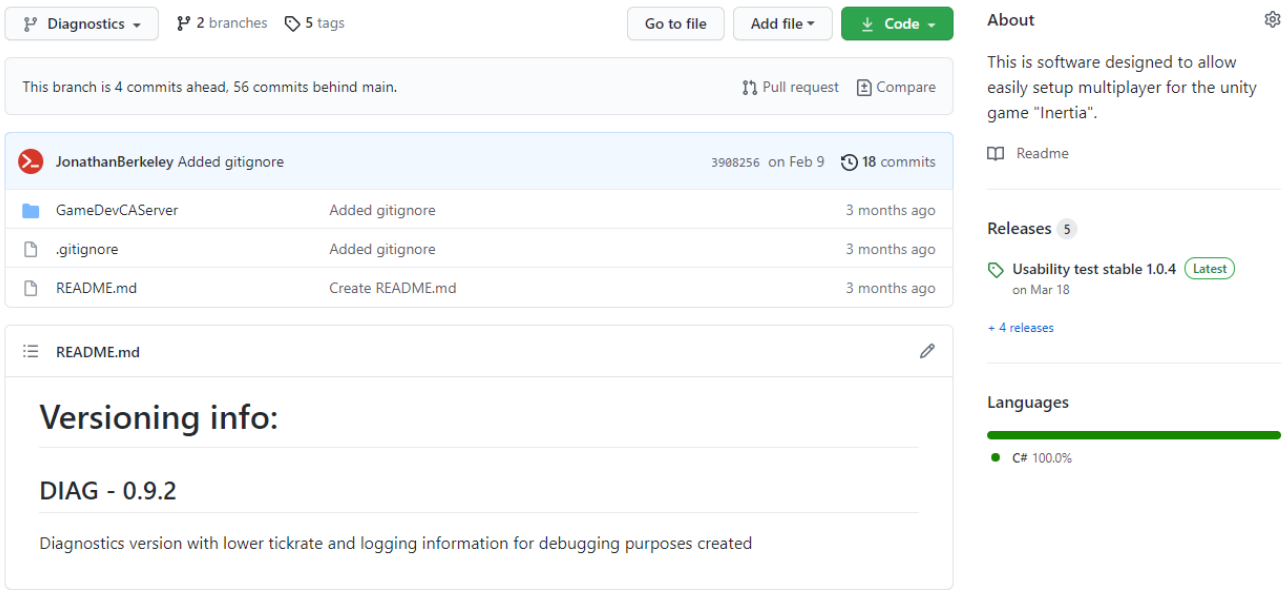


Figure 65 - GitHub diagnostics branch of the server

Item 3

Additional visual assets were developed for the multiplayer mode. As previously mentioned, most of the scripts for the singleplayer had to be modified to support multiplayer.

Figure 66 - Connection menu for the multiplayer

The error prevention has been highly invested in for this part of the game. In early versions of this menu, the client would crash when trying to join an invalid IP, or if the server were on a different port, etc. But with the newest versions there is error handling with various feedback to the client.

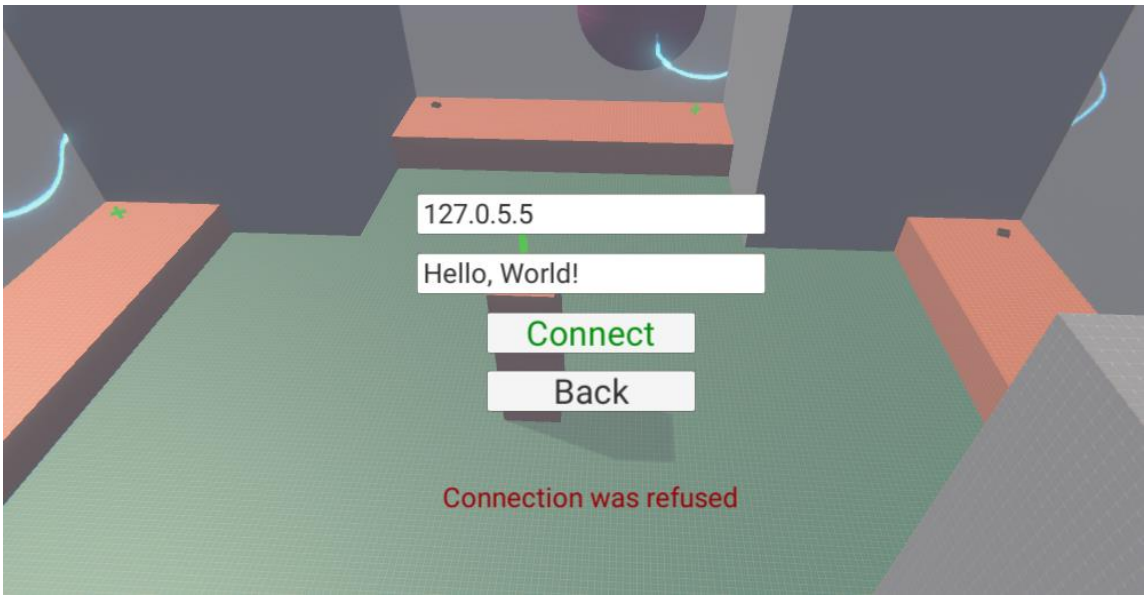


Figure 67 - Error handling example

```
Unity Message | 0 references
private void Awake()
{
    //Runtime retrieval of components for multiplayer prefabs

    //Image
    healthImage = GameObject.FindGameObjectWithTag("MP_HealthImage").GetComponent<RawImage>();

    //Text
    ammoText = GameObject.FindGameObjectWithTag("MP_AmmoText").GetComponent<Text>();

    //Gameobjects
    gameLogic = GameObject.FindGameObjectWithTag("MP_GameLogic");
    deathPanel = GameObject.FindGameObjectWithTag("MP_DeathPanel");
    uiCanvas = GameObject.FindGameObjectWithTag("MP_UiCanvas");

}

Unity Message | 0 references
private void Start()
{
    //Sets the text to initial values
    ammoText.text = ammo.ToString();

    healthImage.texture = healthTextures[0];
    spawnHandler = gameLogic.GetComponent<SpawnHandler>();

    //Used for respawn timer
    awaitingRespawn = false;

    //Caching values so they can be reset later
    respawnCachedTimer = respawnTimer;
    cachedAmmo = ammo;
    cachedHealth = health;

    _disabledComponents = GameObject.FindGameObjectWithTag("MP_DisableComponents").GetComponent<HoldReferences>();

}
```

Figure 68 - Multiplayer version of the PlayerStats class

Classes such as this were rewritten to avoid needing absolute references, and instead could obtain references on runtime. As well as other differences such as networking projectile firing.

```
1 reference
private void SendProjectileDataToServer(GameObject projectile)
{
    var _location = projectile.transform.position;
    var _rotation = projectile.transform.rotation;
    ClientSend.ProjectileLaunchData(_location, _rotation);
}
}
```

Figure 69 - Networked projectile logic

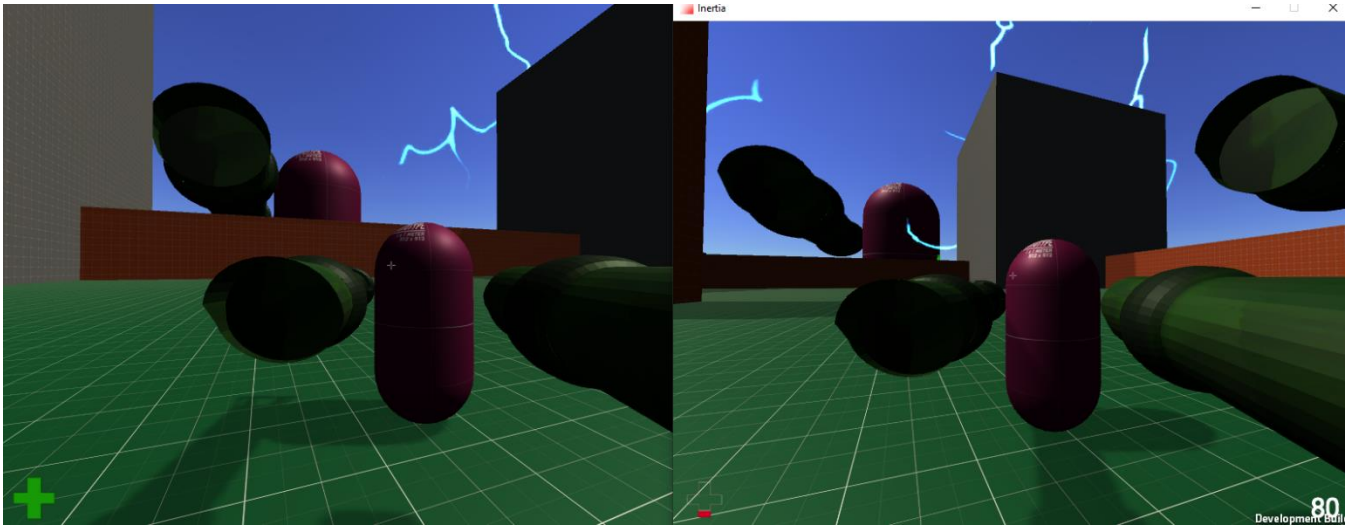


Figure 70 - Two different clients looking at each other in-game.

Above screenshot shows two different clients connected and able to play on the same multiplayer server.

Sprint 7

Goal

Improve the multiplayer, introduce a multiplayer chat function, create a multiplayer command system. Make multiplayer gameplay smoother.

Item 1

Issues surrounding the multiplayer projectiles were fully fixed during this sprint, previously there was bugs with them exploding prematurely, and recursively spawning projectiles.

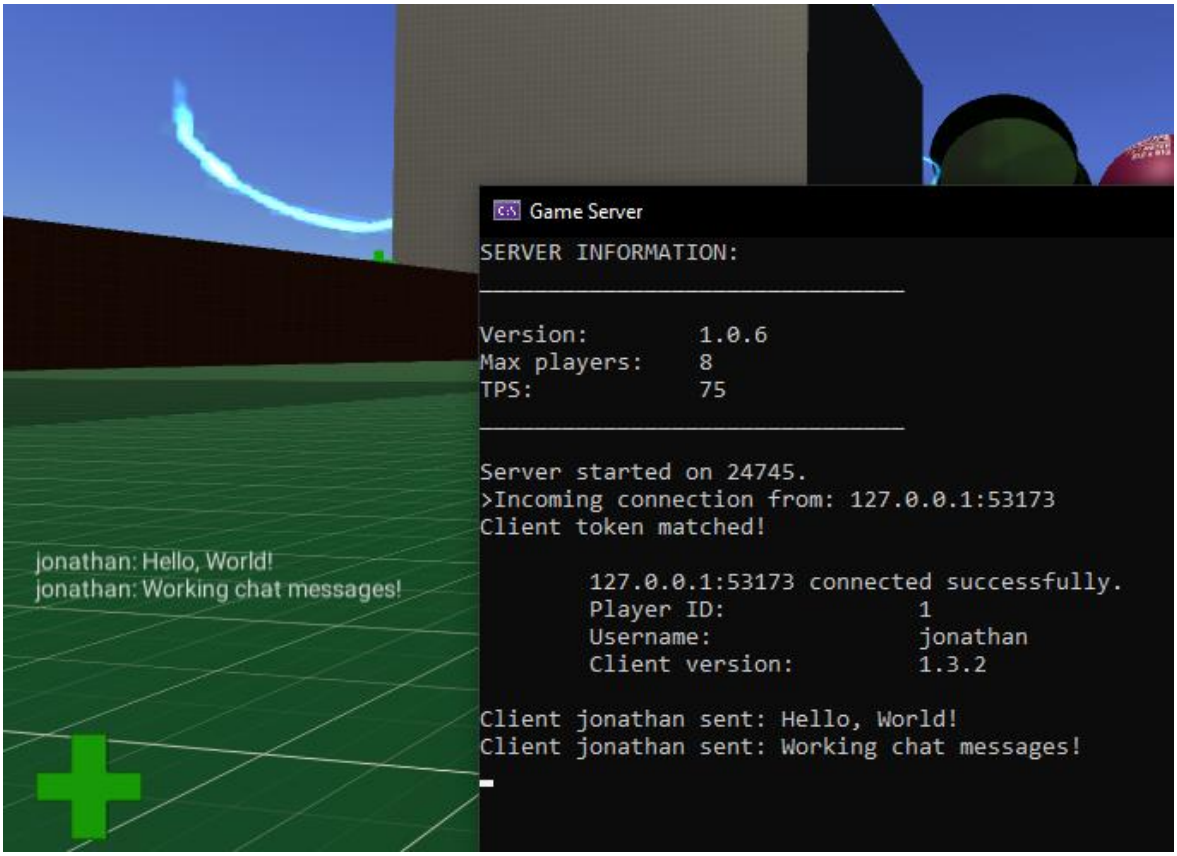


Figure 71 - Game/Server view of chat messages.

The client can send messages to all other clients through chat messages. The messages are put through some sanitization on the server to ensure they do not contain code or rich text or anything else malicious, the chat message packets are checked for size as well and rejected if they are too large.

This functionality was coded from scratch and took some time to get fully working. The messages are sent over the network protocol TCP, whilst the positional/rotational information is sent over UDP. Using two protocols in tandem required a lot of code, as their implementations differ.

```
1 reference
public static void ReceiveClientChat(int _fromClient, Packet _packet)
{
    String _message = _packet.ReadString();

    if (_message.Length < 100)
    {
        #region Sanitization
        //Discard empty/newline/carriage return/whitespace
        string _emptyCheck = Regex.Replace(_message, @"[\n\r\s]+", "");
        if (_emptyCheck.Length < 1)
            return;
        //Sanitization of characters
        _message = Regex.Replace(_message, @"[><\\]", "");
        #endregion

        Console.WriteLine($"Client {Server.clients[_fromClient].player.username} sent: {_message}");
```

Figure 72 - Snippet of client chat code

Above snippet shows the servers sanitization of the messages that come in from the client.

Item 2

Server command system. This system allows clients to type commands that perform actions on the server. For example, the “/msg” command allows private messaging between clients. The client needs to write into the chat “/msg [user] [message]” and if the user exists, they will receive the message.

```
//For server commands
if (_message.Length > 0 && _message[0] == '/')
{
    string _command = Regex.Match(_message, @"^(\/[a-z]+)").ToString();

    if (_command.Length > 0)
    {
        string[] serverCommands = { "/msg", "/colour" };

        bool _validCommand = false;
        //Check if request was a valid command
        for (int i = 0; i < serverCommands.Length; ++i)
        {
            if (serverCommands[i] == _command)
            {
                _validCommand = true;
                break;
            }
        }

        if (!_validCommand)
        {
            ServerSend.ServerMessage(_fromClient, ServerCodeTranslations.invalidCommand);
            return;
        }
```

Figure 73 - Snippet of command checking code.

Many other improvements were made to the networking code during this sprint as well, mostly aimed at preventing crashes and improving server stability. Some errors can cause the server to crash, effort was put in during this sprint to reduce the prevalence of this massively. The server can comfortably handle 8 players on a cheap Google cloud server at this stage.

```
//Server command '/msg'
if (_command == serverCommands[0])
{
    string[] _args = _message.Split(' ');

    //Checks if too few arguments
    if (_args.Length < 2)
    {
        ServerSend.ServerMessage(_fromClient, ServerCodeTranslations.badArguments);
        return;
    }

    string _target = _args[1];

    //Removes the command from the message to be sent
    _message = _message.Remove(0, serverCommands[0].Length + _target.Length + 2);
    //_message = $"<color=\"#A0A0A0\">whispered: " + _message + "</color>";

    //To be sent to self
    string _selfMessage = "You sent: " + _message;

    foreach (var p in Server.clients)
    {
        if (p.Value.player.username == _target)
        {
            ServerSend.ClientChat(_fromClient, _message, p.Key);
            //Prevent doublesending if /msg'd self
            if (_fromClient != p.Key)
            {
                ServerSend.ClientChat(_fromClient, _selfMessage, _fromClient);
            }

            break;
        }
    }
```

Figure 74 - /msg command logic

This functionality was also inspired by that of the popular game “Minecraft”. That game has commands that can be typed into the chat with the same syntax.

```
156         //For message receiving and sending between clients
            1 reference
157         public static void ClientChat(int _playerID, String _message)
158         {
159             using (Packet _packet = new Packet((int)ServerPackets.userMessage))
160             {
161                 _packet.Write(_playerID);
162                 _packet.Write(_message);
163
164                 //Message wont render for the player unless it successfully gets through the server
165                 //Could give latency between sending a message and seeing it on their own screen
166                 SendTCPDataToAll(_packet);
167             }
168         }
169
170         //For messaging specific client
            2 references
171         public static void ClientChat(int _playerID, String _message, int _toClient)
172         {
173             using (Packet _packet = new Packet((int)ServerPackets.userMessage))
174             {
175                 _packet.Write(_playerID);
176                 _packet.Write(_message);
177
178                 SendTCPData(_toClient, _packet);
179             }
180         }
181
```

Figure 75 - Snippet of server's client messaging handling logic.

Item 3

Improving the networking code to make movement smoother was the last part of this sprint. The movement in multiplayer before this was choppy and jittery. The cause of this issue took some investigation using the diagnostic server.

The issue was that the clients were calculating the physics of other clients whilst also updating the clients to the positions sent by the server. This was causing a minor disagreement between the Unity physics engine and the server on where the player should be positioned, causing a jitter between the two close positions.

This was remedied by removing all physics components from the other client multiplayer prefabs. They now obey the position that the server says they are at. This makes the movement far smoother, but again opens the window to video game hacking, where potentially a malicious client could spoof their position to gain an advantage.

```
54      //This is used to handle all code related to spawning players
        1 reference
55      public void SpawnPlayer(int _id, string _username, Vector3 _position, Quaternion _rotation)
56      {
57          GameObject _player;
58          GameObject[] rl = spawnHandler.GetRandomisedSpawns();
59          //If this is data for the current client
60          if (_id == Client.instance.myId)
61          {
62              _player = Instantiate(localPlayerPrefab
63                  , spawnHandler.GetRandomSpawn(rl).transform.position
64                  , _rotation);
65              PlayerID.AssignNewID(_player);
66          }
67          // If it's data for other clients
68          else
69          {
70              _player = Instantiate(playerPrefab
71                  , spawnHandler.GetRandomSpawn(rl).transform.position
72                  , _rotation);
73              PlayerID.AssignNewID(_player);
74          }
75
76          _player.GetComponent<PlayerManager>().id = _id;
77          _player.GetComponent<PlayerManager>().username = _username;
78          players.Add(_id, _player.GetComponent<PlayerManager>());
79      }
```

Figure 76 - Snippet of code for spawning a networked player into the game.

Sprint 8

Goal

The goal for sprint 8 was to fix multiplayer issues relating to disconnections, and to develop an error reporting system. Fix bugs and problems, polish and optimize code.

Item 1

The tutorial I had followed for the server software in the beginning had a severe bug relating to the handling of disconnections which required a recode of sections of the software. There was an issue with disconnected players leaving behind “ghost” prefabs, where other clients would still see the disconnected client after they had disconnected.

As well, if the disconnected client reconnected, it would cause exceptions and sometimes crash the server. The underlying player ID system was reworked, and new code to support the server forcing clients to disconnect was introduced to fix these problems.

```
                4 references
232             public void Disconnect()
233             {
234                 Console.WriteLine($"{tcp.socket.Client.RemoteEndPoint} has disconnected.");
235                 player = null;
236                 tcp.Disconnect();
237                 udp.Disconnect();
238                 //Tells the clients to update their dictionaries
239                 ServerSend.PlayerDisconnected(id);
240             }
241
242             //For disconnects which the other clients have not yet updated their dictionaries for
                3 references
243             public void DisconnectUnregistered()
244             {
245                 Console.WriteLine($"{tcp.socket.Client.RemoteEndPoint} has disconnected.");
246                 player = null;
247                 tcp.Disconnect();
248                 udp.Disconnect();
249             }
250
```

Figure 77 - Snippet of new disconnect logic.

The DisconnectUnregistered function shown above is used for disconnection clients which have connected to the server but for which the other clients on the server have not yet been told exists.

Item 2

The server makes use of Enums to report errors between the client and the server. This is a more human readable way to transfer information about error states.

```
namespace FlagTranslations
{
    12 references
    internal enum ServerCodeTranslations
    {
        serverFull = 1,
        invalidUsername,
        badVersion,
        usernameTaken,
        userNotFound,
        badArguments,
        invalidCommand,
        badToken
    }
}
```

Figure 78 - Enum class on server software

```
//Used to make code neater and more readable
namespace FlagTranslations
{
    15 references
    internal enum ServerCodeTranslations
    {
        serverFull = 1,
        invalidUsername,
        badVersion,
        usernameTaken,
        userNotFound,
        badArguments,
        invalidCommand,
        badToken
    }

    13 references
    internal enum ClientCodeTranslations
    {
        noError,
        connectionRefused,
        badForms,
        lostConnection
    }
}
```

Figure 79 - Enum class on client software

```
2 references
internal void ProcessServerMessage(ServerCodeTranslations _message)
{
    //To make future improvements easier
    GameObject[] toDeactivate = { errorIngameMessage, errorBackPanel };

    switch (_message)
    {
        case ServerCodeTranslations.serverFull:
            errorConnectMessage.text = "Server full";
            break;
        case ServerCodeTranslations.invalidUsername:
            errorConnectMessage.text = "Username invalid";
            break;
        case ServerCodeTranslations.badVersion:
            errorConnectMessage.text = $"Server not accepting client version {Constants.CLIENT_VERSION}";
            break;
        case ServerCodeTranslations.usernameTaken:
            errorConnectMessage.text = "Username taken";
            break;
        case ServerCodeTranslations.userNotFound:
            IngameError("User not found");
            break;
        case ServerCodeTranslations.badArguments:
            IngameError("Invalid arguments");
            break;
        case ServerCodeTranslations.invalidCommand:
            IngameError("Invalid command");
            break;
```

Figure 80 - Translation of Enums into end user UI alerts

As shown above, the errors from the server are converted into human readable text and displayed to the end user to give them information about what went wrong. Before this, an error would cause an abrupt exit of gameplay without any information to a user with no access to the debugging tools.

```
4 references
internal void ProcessClientMessage(ClientCodeTranslations _message)
{
    switch (_message)
    {
        case ClientCodeTranslations.noError:
            errorConnectMessage.text = "";
            break;
        case ClientCodeTranslations.connectionRefused:
            errorConnectMessage.text = "Connection was refused";
            break;
        case ClientCodeTranslations.badForms:
            errorConnectMessage.text = "Forms unfilled";
            break;
        case ClientCodeTranslations.lostConnection:
            errorConnectMessage.text = "Lost connection";
            break;
        default:
            break;
    }
}
```

Figure 81 - Translation of client errors

Client errors are not the fault of the server, but instead of the client. They are processed in a similar way using their own Enum flags.

Item 3

The rest of this sprint was dedicated to improving code. Some code from the game had crept into the networking code. This was making the code unnecessarily confusing, as there was game logic mixed with networking logic. The code that interfaces the network to the game was all moved into the GameManager.cs class.

```
211     //These are placed in here to remove code that interacts with the game scene from networking code
            1 reference
212     internal void LoadScene(string scene, ClientCodeTranslations clientCode)
213     {
214         StaticError.CCT = clientCode;
215         SceneManager.LoadScene(scene);
216     }
217
            0 references
218     internal void LoadScene(string scene, ServerCodeTranslations serverCode)
219     {
220         StaticError.SCT = serverCode;
221         SceneManager.LoadScene(scene);
222     }
223
224
225     //Resets player list on disconnect
            2 references
226     public void ResetDictionary()
227     {
228         players = new Dictionary<int, PlayerManager>();
229     }
230
```

Figure 82 - Networked loading scene code

Above shows a portion of the code that was moved into the GameManager class. This code loads a scene with an error message to be displayed from the server when loading is finished.

Sprint 9

Goal

This is the last and final sprint, for this sprint the goal was to improve the appearances of the game, from the user interface to the general graphics. Alongside this, the development of a settings menu with more advanced options for users.

Item 1

Graphical enhancements such as post-processing was developed for the game.

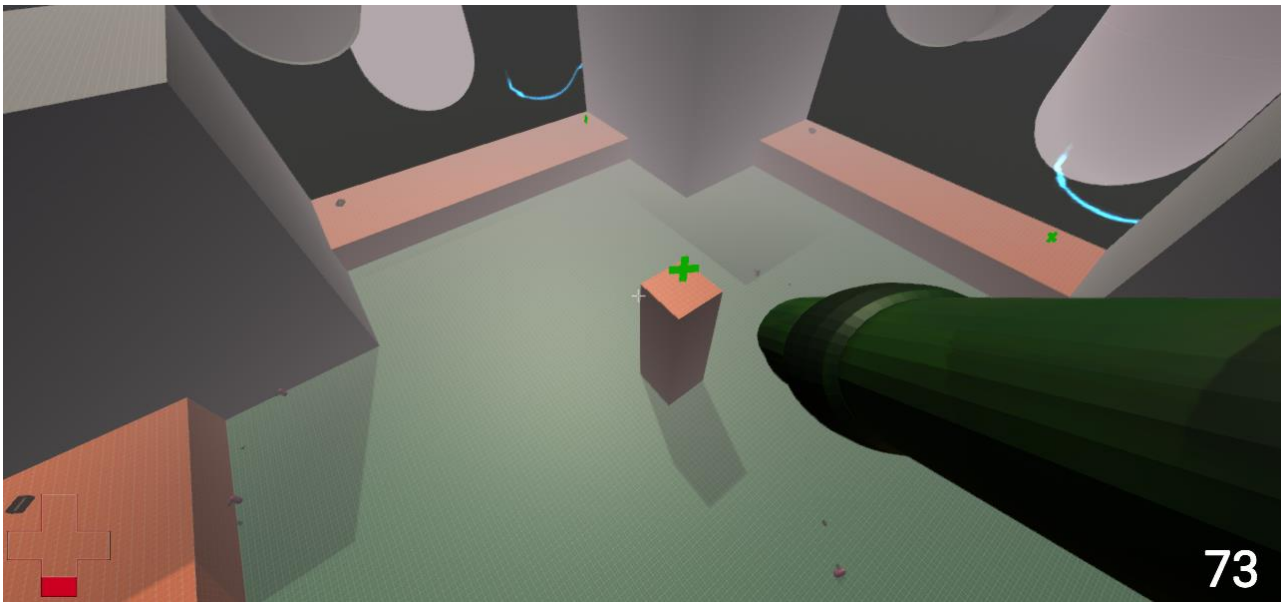


Figure 83 - Post processing off

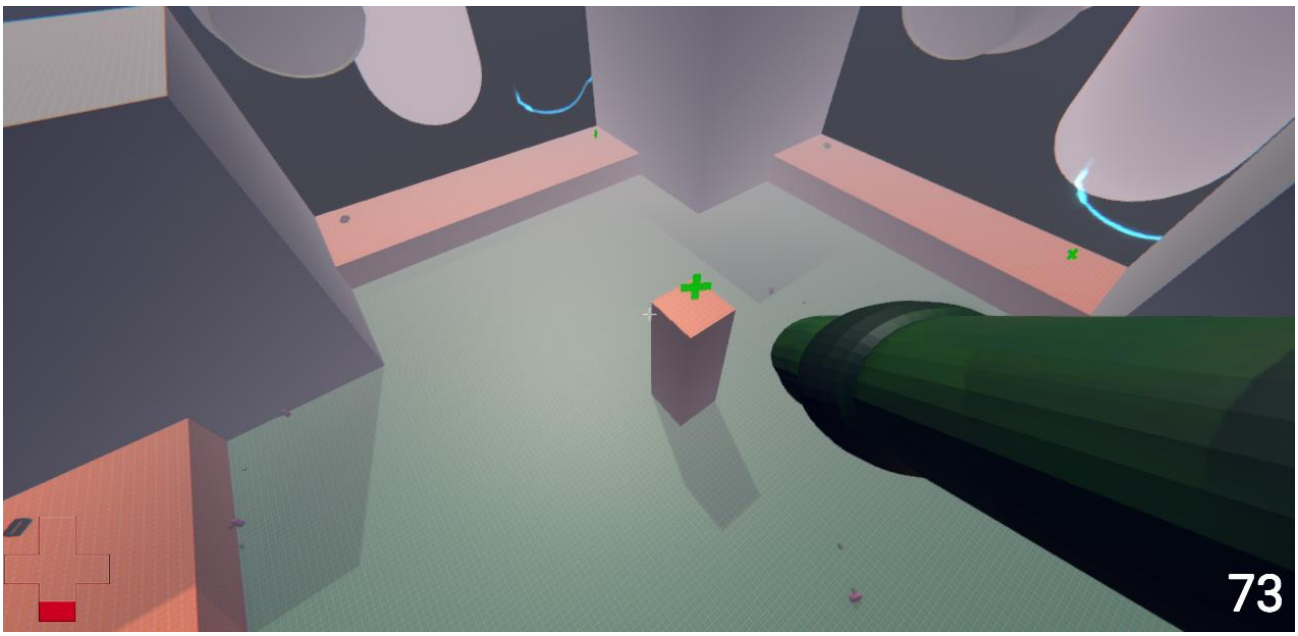


Figure 84 - Post processing on

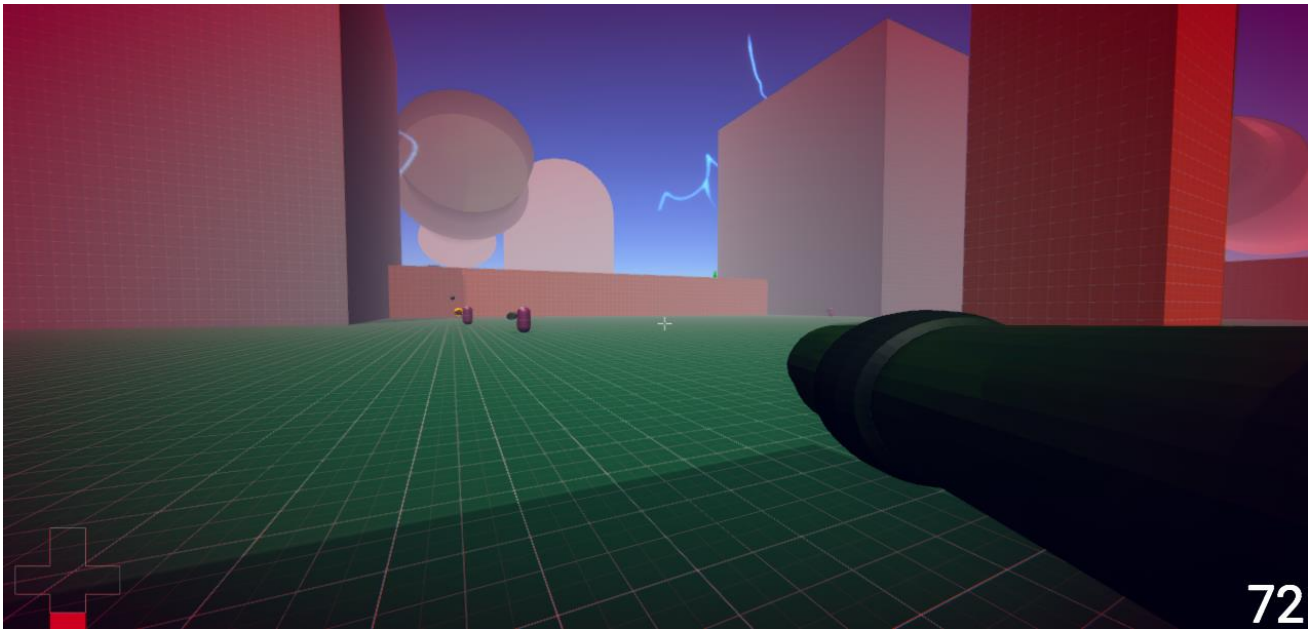


Figure 85 - Red vignette effect to emphasis the players low health

These effects are added to make the game feel more polished, without straying too far from the cartoonish theme.

The textures of the props in this combat mode are inspired by popular maps in similar themed games that its players know and love.

Figure 86 - Picture from CP_Orange, a custom map used in source games.

The above map is a screenshot from the game Team Fortress 2, it is a map which inspired the model choices of the games combat mode. Although the bright orange can be jarring at first, it helps with spotting enemies, and is very well known and liked by players within this genre of video games.[3]

Item 2

Settings menu created with settings saving to registry. This menu was made to be easy to use with little clutter. It is separated into three tabs. The settings save to the registry and persist through restarts of the game.

Figure 87 – Controls information page

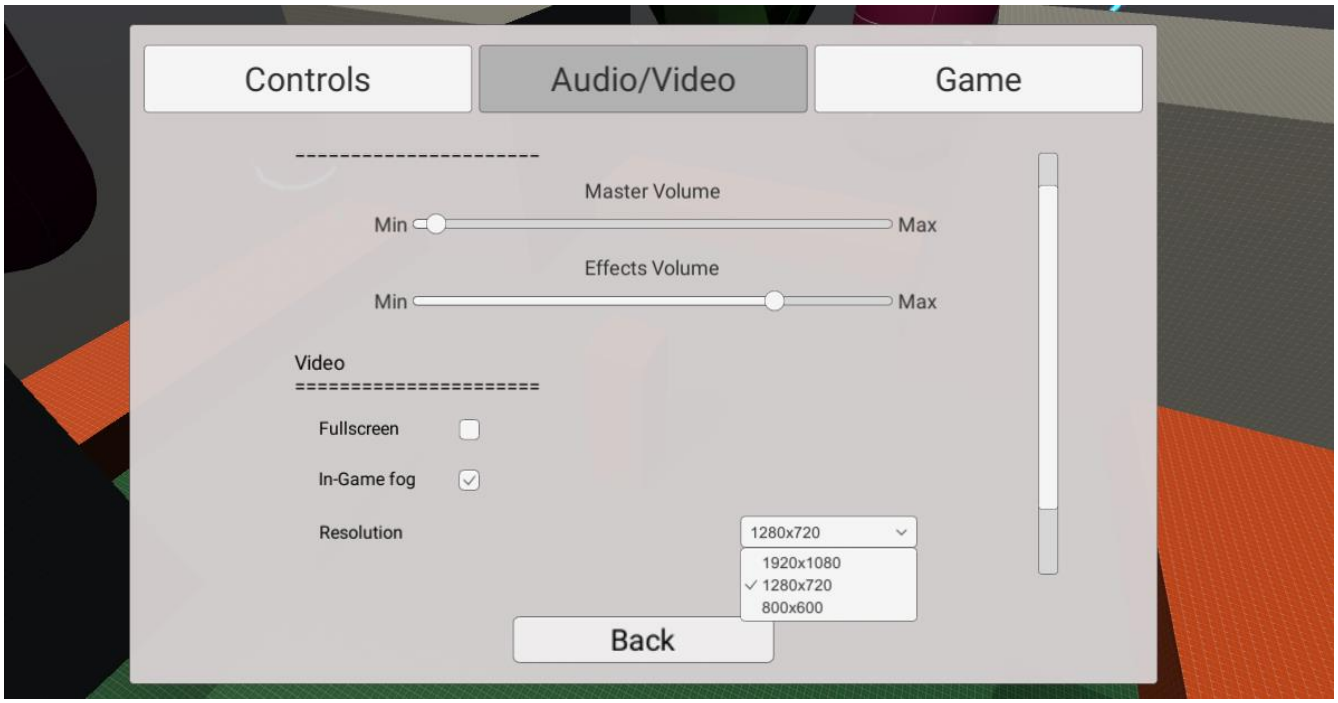


Figure 88 - Audio/Video, with resolution dropdown extended.

```
34      public void SetSelected(int _select)
35      {
36          _currentlySelected = _select;
37          EnableSelected();
38
39          if (_currentlySelected == 1 && innerMenus[1] != null)
40          {
41              //Sets sliders visually to respective values
42              Slider[] audioSliders = innerMenus[1].GetComponentsInChildren<Slider>(); //Obtains the references to sliders
43              audioSliders[0].value = GlobalAudioReference.instance.GetMasterVolume(); //Master volume
44              audioSliders[1].value = GlobalAudioReference.instance.GetEffectsVolume(); //Effects volume
45          }
46
47      }
48
49      //Referenced dynamically by volume slider
        0 references
50      public void SetMasterVolume(float _volume)
51      {
52          if (GlobalAudioReference.instance != null)
53              GlobalAudioReference.instance.SetMasterVolume(_volume);
54      }
55
        0 references
56      public void SetEffectsVolume(float _volume)
57      {
58          if (GlobalAudioReference.instance != null)
59              GlobalAudioReference.instance.SetEffectsVolume(_volume);
60      }
61
```

Figure 88 - Snippet of code from new options menu.

Above snippet shows some of the logic behind the settings menu. Additional classes were created alongside the options menu for this sprint, such as the GlobalAudioReference class, which maintains a universal value and accessing/modifying methods for audio across the game.

Figure 89 - Menu has a script to change its colour over time.

```
1 reference
private void ColourChange()
{
    //Debug.Log($"R: {(int)(image.color.r * 255)} G: {(int)(image.color.g * 255)} B: {(int)(image.color.b * 255)}");

    if (image.color.r < tinyUnit * 10)
        rFlip = false;
    else if (image.color.r > 1.0f)
        rFlip = true;

    if (image.color.g < tinyUnit * 10)
        gFlip = false;
    else if (image.color.g > 1.0f)
        gFlip = true;

    if (image.color.b < tinyUnit * 10)
        bFlip = false;
    else if (image.color.b > 1.0f)
        bFlip = true;

    int _path = Random.Range(1, 4);
    switch (_path)
    {
        //Red
        case 1:
            if (rFlip)
                image.color = new Color(image.color.r - tinyUnit, image.color.g, image.color.b, image.color.a);
            else
                image.color = new Color(image.color.r + tinyUnit, image.color.g, image.color.b, image.color.a);
            break;
        //Green
        case 2:
            if (gFlip)
                image.color = new Color(image.color.r, image.color.g - tinyUnit, image.color.b, image.color.a);
            else
                image.color = new Color(image.color.r, image.color.g + tinyUnit, image.color.b, image.color.a);
            break;
        //Blue
        case 3:
            if (bFlip)
                image.color = new Color(image.color.r, image.color.g, image.color.b - tinyUnit, image.color.a);
            else
                image.color = new Color(image.color.r, image.color.g, image.color.b + tinyUnit, image.color.a);
            break;
```

Figure 90 - Code snippet from CoolMenu.cs.

The code shown above shows the logic for changing the colour of the menus background smoothly over time, giving it an interesting and arcade feel. The background to the menu is a 3D rendering of the map, with a script attached to a camera making it orbit around map.

Conclusion

The implementation chapter has covered the development of the game. The purpose of SCRUM methodology in this project should now be clear. The sprints kept development concise and focused. There were clear goals set out and achieved during the sprints.

The implementation chapter has also covered the projects approach to bug fixing and testing. Some of the usage of GitHub has been discussed, as well as the role third party software such as Blender played in the development.

The game has a heavy focus on a multiplayer experience, which can be seen throughout this chapter, the game is focusing on an older style of game such as previously mentioned well respected titles Team Fortress 2, and Quake, with similar themes and concepts.

Sources

1. https://guntherverheyen.com/wp-content/uploads/2013/01/takeuchi-and-onaka-the-new-new-product-development-game.pdf
2. D. (2020, November 5). What is Scrum Methodology? & Scrum Project Management. Digite. https://www.digite.com/agile/scrum-methodology/
3. CP_Orange, Team Fortress 2 map, https://wiki.teamfortress.com/wiki/Orange_X

5 Testing

5.3 User Testing

1. Participant tasks

The participants to the usability surveys will be asked to perform a set of tasks to better understand the strengths and weaknesses of the design of the program.

- Launch a singleplayer game (Player Vs Bots). And then return to the main menu. This will test the ease of navigation for the core functionality of the game.
- Change the setting for the number of bots present in singleplayer. This is aimed at testing the ease of use of the settings panel in the game.
- Join a multiplayer server with an I.P address and send a message to the multiplayer chat box. This will test the more complicated functionality of the user interface.
- Lastly, as an extra challenge, participants were asked if they could manage to get on top of the spire in the multiplayer game mode. This is aimed at testing if the user intuitively grasps the concepts of momentum in the game, as reaching this location is only possible with a solid understanding of the core physics mechanisms present in the game.

2. Type of participants required

The participants required for this survey need to have some experience with gaming and with computers. If someone has very little knowledge of videogames or computer conventions, I would expect that it would take quite long to understand how to interact with the game. This is a problem for most videogames, they typically have a learning curve, and take some time to understand.

To reduce this learning curve, the game has stuck to conventions that are typically present in other games, so that gamers with experience will have transferable knowledge. To test that these conventions have come across, the game should be tested by people who play videogames.

The demographic for the participants is ideally within the age range of 21-50 years of age, which is the range most videogame players reside.[1]

3. Ease of use or ease of learning

The usability test will focus on ease of learning, since it is a videogame and not a website, some concepts need to be explained to the user. Judging how easily and fast the participants take up information while doing the testing will be the most valuable information.

The reason for focusing on ease of learning over ease of use is because since it is a unique videogame, there are concepts which cannot be understood instinctively by the user, and instead need to be taught by the game.

5. Participant environment

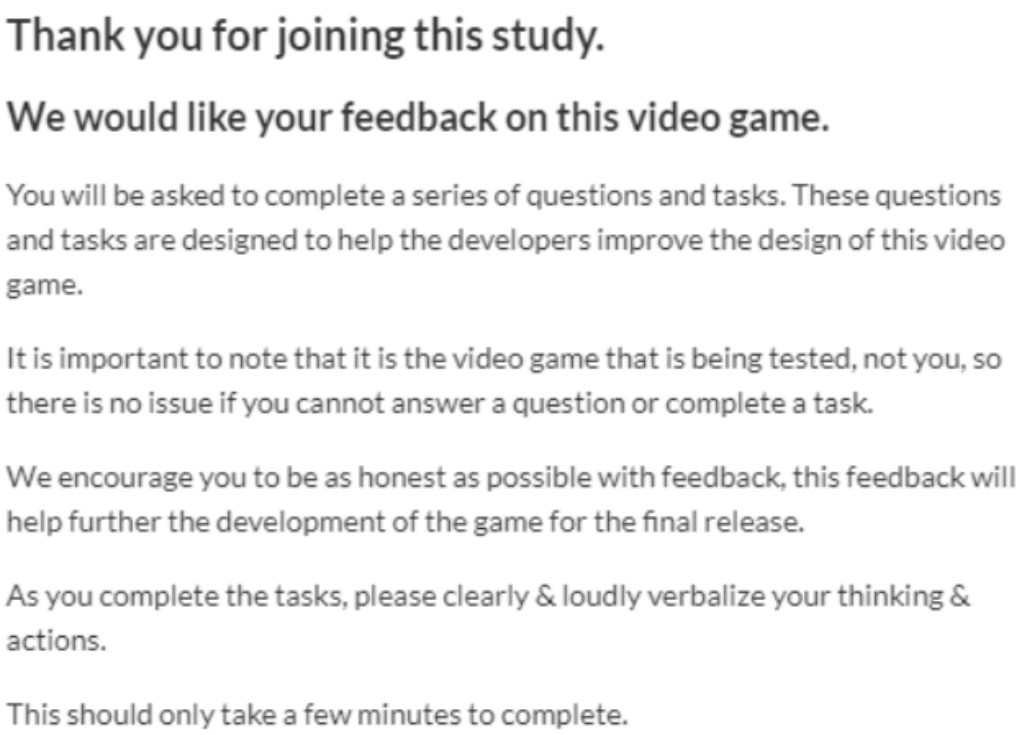
To keep the testing accurate, it is best that the testing environment for the participants be as like the real environment that a user would play the game in would be. This ideal environment is at home, and for our game, at a home computer.

Due to COVID-19, the testing was conducted online. However, working through this, the game was built into a multiplatform redistributable .exe file, which was compiled through Unity's build feature. This allowed the participants to simply download the game and run it on their own machine. The file was hosted as a release on GitHub, a link was provided to the participants to download from.

Once downloaded they could play the game from their own home on their own computer, which is the ideal environment as it is the same as the environment that most of our end-users would be in. All our participants were in the same environment that they would be for performing similar tasks.

Other methods such as hosting the game on a website were explored, but to get the full experience of the game as it would be for an end-user, the executable file was preferred.

6. Documents



Thank you for joining this study.

We would like your feedback on this video game.

You will be asked to complete a series of questions and tasks. These questions and tasks are designed to help the developers improve the design of this video game.

It is important to note that it is the video game that is being tested, not you, so there is no issue if you cannot answer a question or complete a task.

We encourage you to be as honest as possible with feedback, this feedback will help further the development of the game for the final release.

As you complete the tasks, please clearly & loudly verbalize your thinking & actions.

This should only take a few minutes to complete.

Figure 89 - Introduction text

The introduction was read verbally to the participants, followed by the consent form.

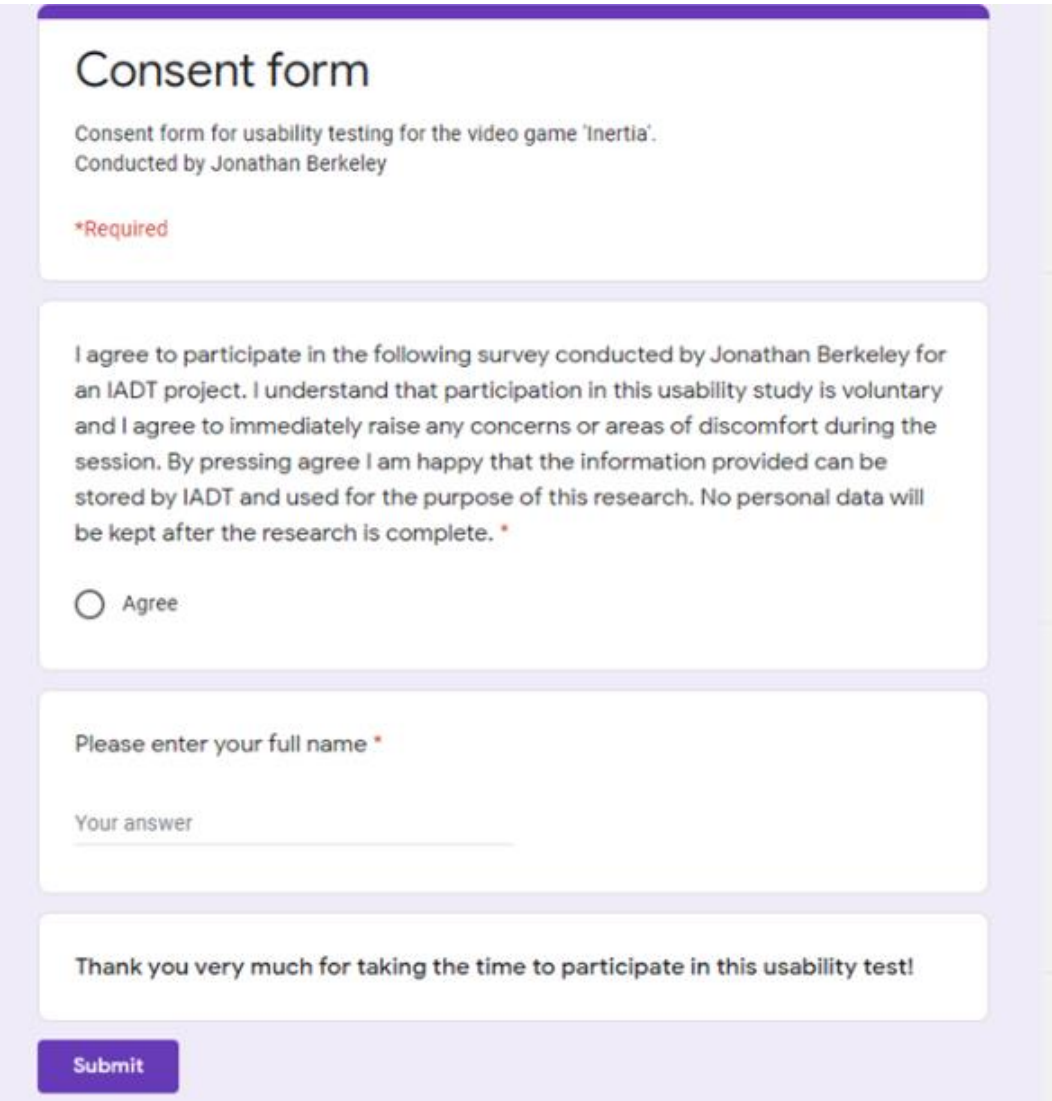


Figure 90 - Consent form

Google forms was used for my forms. After I verified that they had consented to continuing the usability test, the pre-test/demographic survey was sent to them.

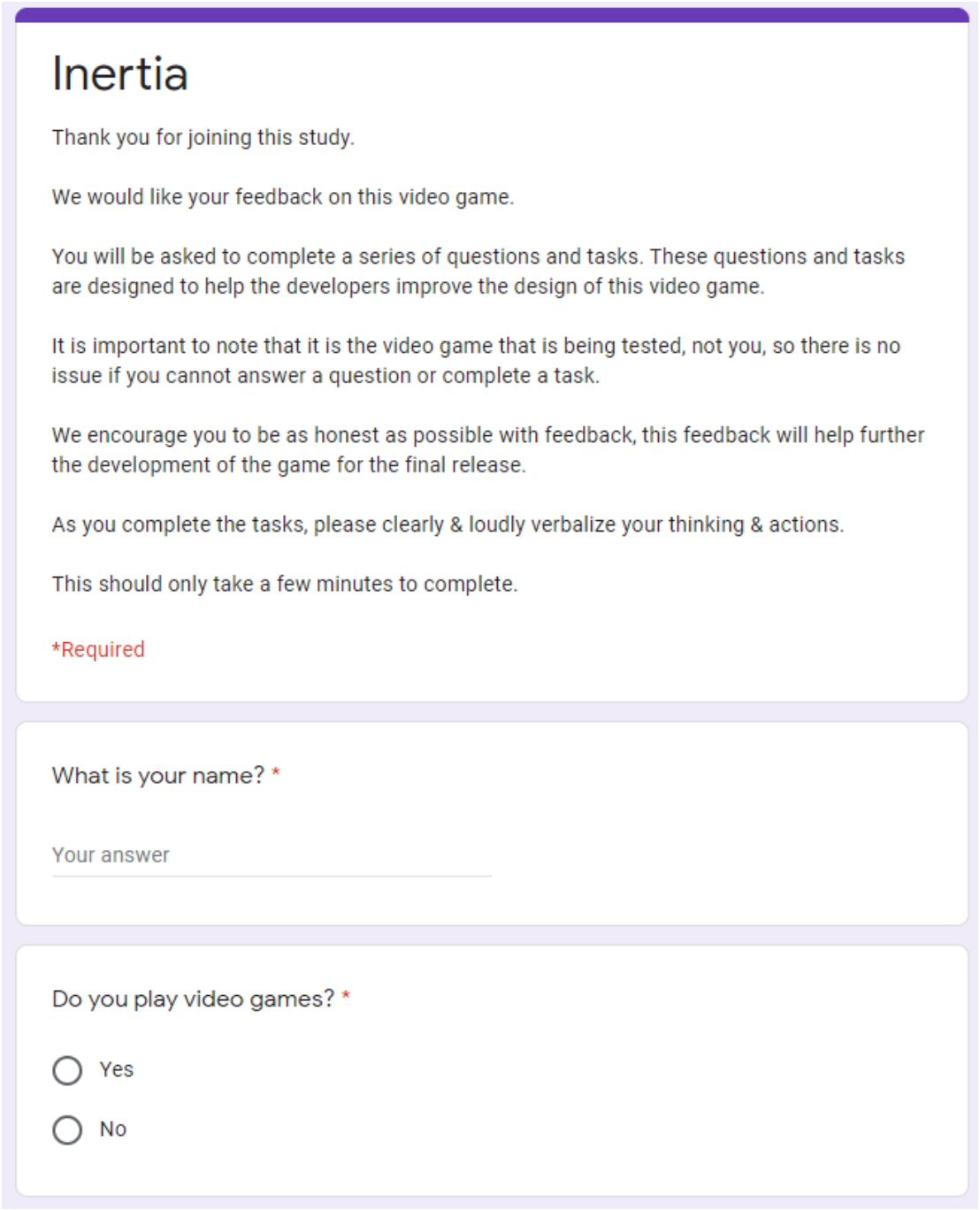


Figure 91 - Demographic questions

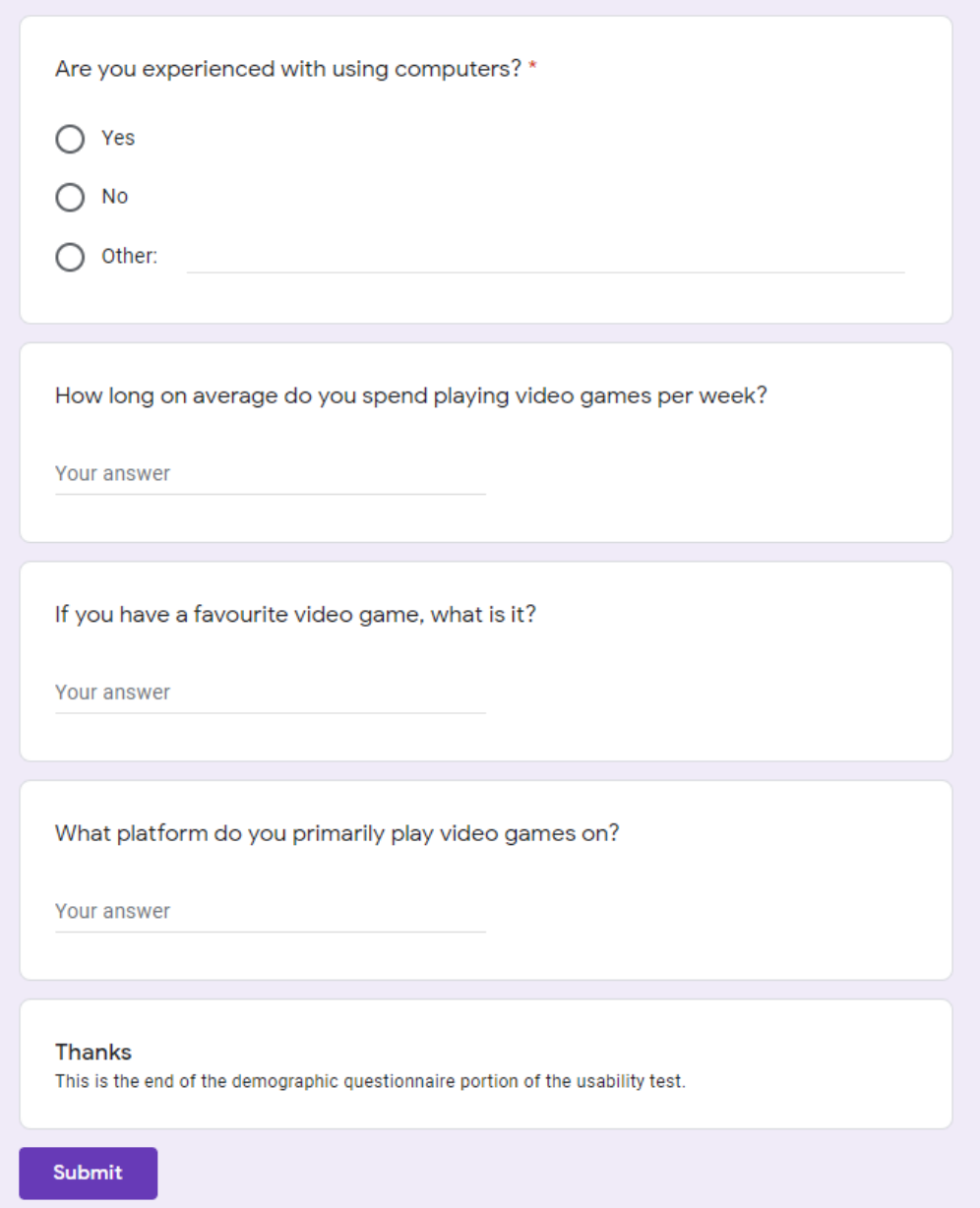


Figure 92 - Demographic questions

These questions were asked to get a better understanding of the experience of the participants, ideally this game should be tested by someone that has some prior gaming experience as that is our target market for the game.

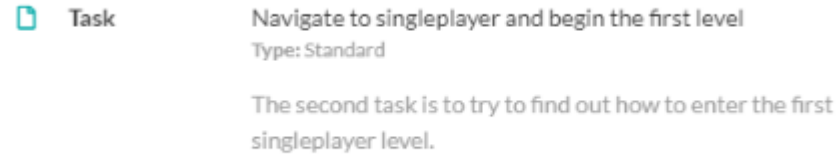


Figure 93 - Task 1

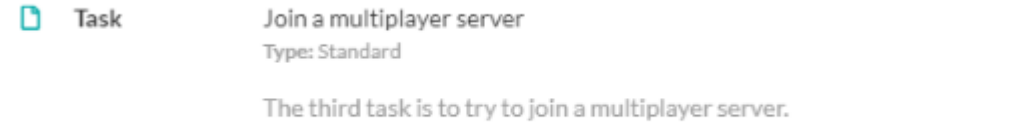


Figure 94 - Task 2

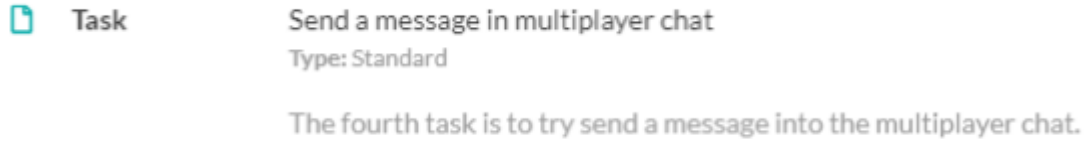


Figure 95 - Task 3

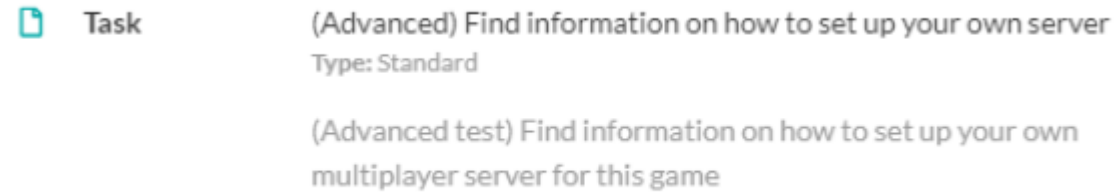


Figure 96 - Final task

The final task was substituted for a different task during testing, due to the task's ambiguity. The task was replaced with an in-game test to see if the participant could perform a "Rocket-Jump", a more advanced concept in the game where the player uses an in-game item to reach an otherwise inaccessible area by using physics.

The tasks were designed to test various important aspects of the user interface design and game functionality, as well as the ease of learning of the controls for the game.

The post-test for these usability tests were conducted verbally. The participants were asked the following questions:

1. Did you find any aspect of the game confusing?
2. Did you experience any issues or notice any design flaws with the user interface?
3. Did you find the game overwhelming?
4. Were there any controls or keybinds not where you expected them to be?
5. How was your experience with the game overall?

7. Data summary

7.1 Consent form

All participants consented to the usability test.

7.2 Demographics form

Figure 97 - Demographic data

All five of the participants to the usability survey indicated that they both had experience playing videogames, and they had experience with computers. This is exactly the type of participants that was desired for this usability test, as it is the target demographic for the game.

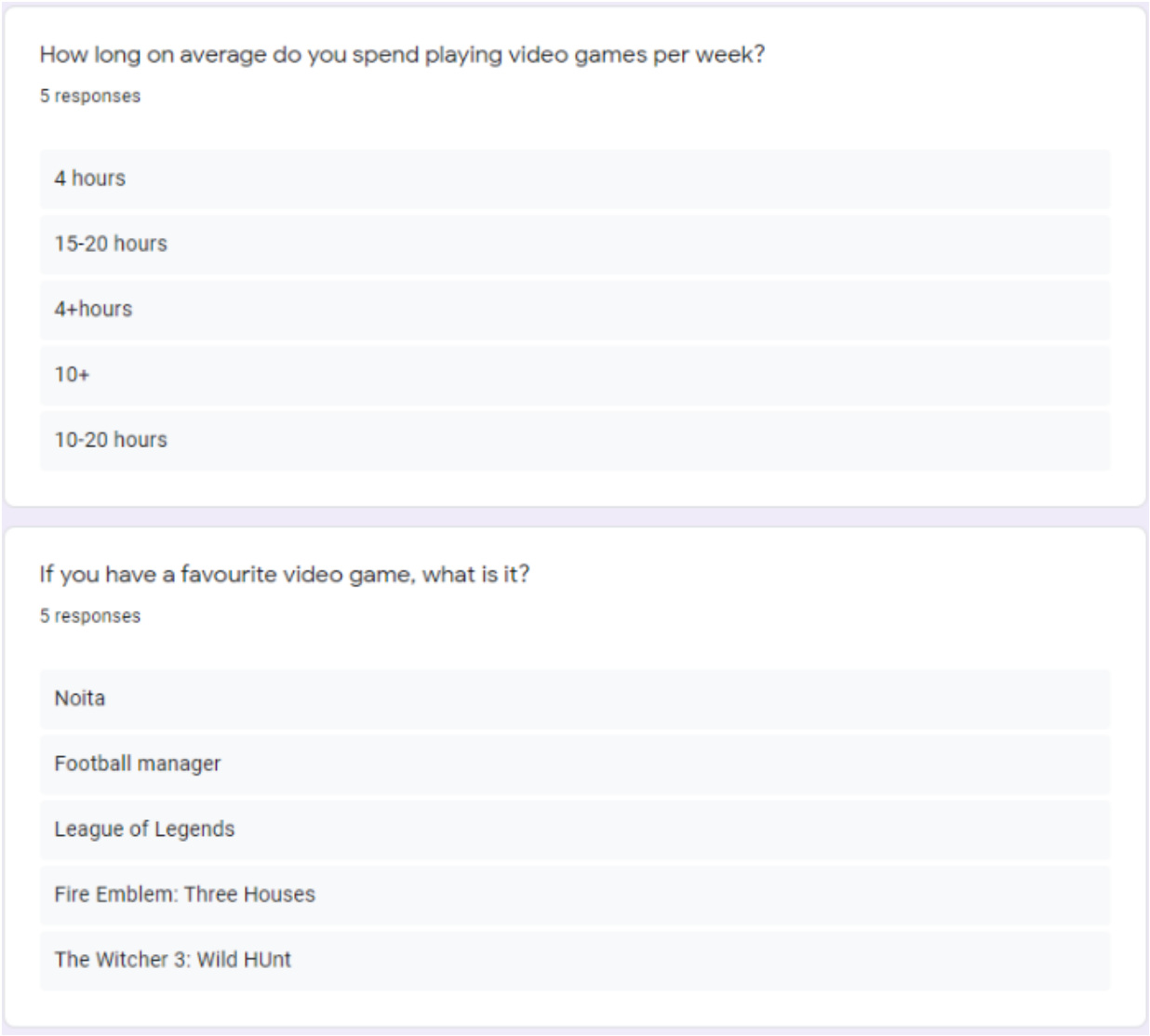


Figure 98 - Demographic data

According to the survey responses, our participants spend an average of 10 hours a week playing video games. This is roughly 1 hour and 30 minutes more per week than the average gamer.[2] The participants were in the perfect range of experience for this testing.

The favourite video game information was less favourable, the games mentioned were far different to genre of our game. Ideally, the participants would be interested in First-Person Shooters (FPS) or physics puzzle games like Portal.

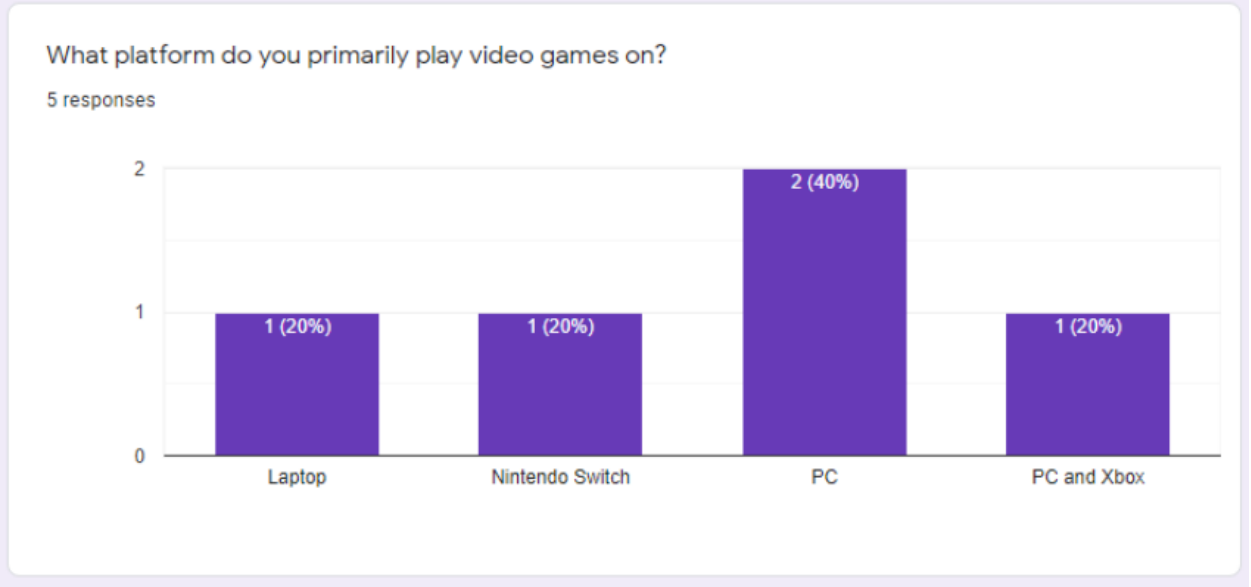


Figure 99 - Demographic data

This was most positive information for the testing, all but one of the participants listed a PC as their main gaming platform, which is the platform that our game is targeting.

7.3 Post-test questions

This revealed the most useful information out of the testing. Issues with the game's design were identified and discussed here. I think that it was a good decision to conduct the post-test verbally as the less formal setting made it easier to discuss design issues and understand the problem. The information obtained can be summarised as follows:

1. The design of the menus was inconsistent.
2. The settings menu design needed to be improved.
3. The controls/keybinds for the game were not always intuitive.
4. The default volume for the game is far too loud.
5. The actual gameplay was interesting and enjoyable.
6. The movement mechanics and concepts needed a dedicated tutorial.
7. The models and projectiles in the game can be difficult to see, especially when on a dark background.

8. Conclusions and design changes

The testing was very helpful to identify issues with the game that had not previously been known.

- Immediately a bug was discovered and fixed due to this testing.
- The default sound has been reduced and is now controllable.
- The design of the main menu will change and be merged so that the design is consistent across the game.
- The movement mechanics have a tutorial in development.
- The settings menu will be changed to be more intuitive; a design has already been wireframed and now needs to be implemented in code.
- The projectiles will have a particle trail to make them more visible.
- There will be a dedicated section to the settings for changing controls.

6 Project Management

6.1 Introduction

This chapter goes into detail on the topic of project management, focusing on areas such as how the team worked together, and what third party services were used to aid development. The design, implementation and testing phases are discussed here in relation to how the team operated through these phases.

6.2 Project Phases

6.2.1 Proposal

The proposal was one of the most cohesive stages, at the beginning there was meetings to discuss the projects development trajectory, the theme of the game, and the ideas and aspirations for the final product. The meetings, which were conducted online due to COVID-19, were very helpful in outlining the beginnings of the project. In these meetings, we discussed in detail all the different features that needed to be developed to make this game a success.

The initial idea was to go for a 3D movement game similar to that of 'Refunct'. The game is a simple game made by one developer, focusing on simplistic beauty with elegant movement through a changing landscape. There were a set of ideas and fallback ideas discussed during this period. The final idea at this stage was to create a game with a lab-like feel, similar to the game series 'Portal'. Where the player does not know their own backstory or understand anything but learns as they go along through the world.

After all the proposal discussion, it was decided to create a 3D motion game with grapple hooks and other movement mechanics as a focus. I created a GitHub repository and a Trello during this phase to track the progress of the game, as well as make it easier to log issues and pitch ideas.

6.2.2 Requirements

The team worked similarly for the requirements phase as it did for the proposal phase. It was early days still, and the development of the game had begun in a very early fashion. The very start of the scripts were being developed, and assets imported during this stage.

The idea for the development during the requirements stage was to first develop a foundation of a game, then aim high, and if things went wrong or the game was too difficult to develop, then there would be the foundational game to fall back on.

As we both play video games as a hobby, we already had extensive knowledge of the field of videogames. And so little research was needed to find competitors and similar games, as we already had a lot of experience with similar games and their concepts.

This made the requirements phase easier, as we had a consensus on what we wanted to make, yet had a fallback option if things proved to be too intense.

6.2.3 Design

There was good teamwork for this phase of the development. There were regular online voice meetings being held where the design of the game was being discussed, we created paper prototypes for what we thought the game should look like.

As we are developing a game, the design is less important than that of a website. In order to make best use of this phase, the design focused on level design as well as user interface design. The general layout and logic for the levels was discussed and determined during this phase.

The final design for the game was not determined, but after receiving the survey results that were conducted during this phase, we had a clear direction to go in development.

6.2.4 Implementation

The implementation phase went well, but it was largely independent. There was an initial avoidance to editing each other's work which caused a larger fracture in the game's development. We were never working on the same assets or level for the sprints, it made a fracture whereby I had my assets and levels and scripts, and my project partner had his own separate level and assets.

This led to an issue of there being duplicate menus, where we had different visions for how the main menu and settings should look, so our ideas conflicted during this stage.

The workload was also off-balanced, I am personally responsible for the vast majority of the scripts. The multiplayer mode and combat mode was developed entirely independently.

It would have been better if there was more teamwork and interaction during this phase.

6.2.5 Testing

The testing phase was conducted almost entirely individually. I had created my own survey and consent forms, I conducted my own usability tests, and my project partner conducted their own. At this point it did not make sense for me to conduct usability tests for levels and designs that I had no part in authoring, and the same for my partner who had not developed my section.

The cut between the game had become obvious by this stage, there was a portion of the game that my partner had created, with the indoor labs, and the part that I had created, with the explosive combat mechanics.

There was essentially no teamwork for the testing phase of the game, and it was instead performed individually. This was not initially planned but occurred due to my partner not attending the pilot usability tests, so I then decided to continue with the testing by myself.

6.3 Teamwork

6.3.1 Roles

I created the vast majority of code for the game, as well as the multiplayer code and server. I create various assets for the game such as the multiplayer arena level, which is the same as the singleplayer arena level. I developed some of the assets for the game, and a portion of the user interface which has been discussed in more detail in the implementation chapter of this report.

I have maintained the Trello board, and the GitHub repositories. We controlled a shared Miro board, which was used for the design stage of the project.

My partner created the grapple hook levels, as well as the menus and some of the scripts that go with it. My partner also sourced assets to download from the Unity asset store for the game.

6.3.2 Communication

Communication was very good at first, there were frequent meetings and open discussion. The communication worsened as the year progressed, at one point there was a period of silence as my partner took some time out for personal reasons, but communication was restored afterwards.

My project partner has always been very friendly and open to talk about ideas which has been encouraging for the development of the game, the input and feedback has been important for the games progress.

6.3.3 Difficulties

The main difficulties in this project stemmed from the complication of the server code, it was a huge time sink. It was like an entire project on its own. Getting simple features working for the server could take several days due to the several moving parts involved. It was compounded by the fact that the foundational code, that I had watched a tutorial to create, was very buggy and unstable.

The main issue for the group was the early fracturing of code and ideas, instead of ending up with one cohesive project there is instead two different version with different design and focus.

6.3.4 Resolving difficulties

Nearly all the server code was modified and improved in some way making it more stable and usable. From there features were built on top and tested rigorously before continuing to develop more features.

There was discussion between us to talk about potential solutions to the difference in appearance to our different sections of the game. The differing sections of the game was soothed by using the same font, graphics, and image type in the menus across the different game-modes.

6.4 SCRUM Methodology

I believe the SCRUM methodology worked well. It was a great way to keep development focused, concise and on point. The sprints were crucial in structuring the development of the project. Without it, it would have been more likely that things got left until the last minute, or things that needed to be developed would have been forgotten about.

The meetings with the supervisor were crucial in self-reflection on the project, discussing changes and plans and hearing continual outside feedback was invaluable for the development.

The SCRUM methodology works well with my preferred mode of working as well, if you focus on a very large end goal it is difficult to motivate yourself to work on it at

the start. However, when it is split up into smaller chunks over a period of time it is far more motivating to work on since you have visible goals which are being completed along the way.

Additionally, it is good as it guarantees you breaks if you fulfil your sprint requirements. For instance, I did not need to work over Christmas, but had I been developing without structure, I may have needed to in order to have the project ready on time.

6.5 Project Management Tools

6.5.1 Trello

Trello is a website that has cards which you can add code ideas, concepts, TODOs, or really anything to. It provides an excellent way to structure what needs to be done for a project, as well as for throwing out new ideas into the group, easily declare to other members what has been finished, and what is being worked on.

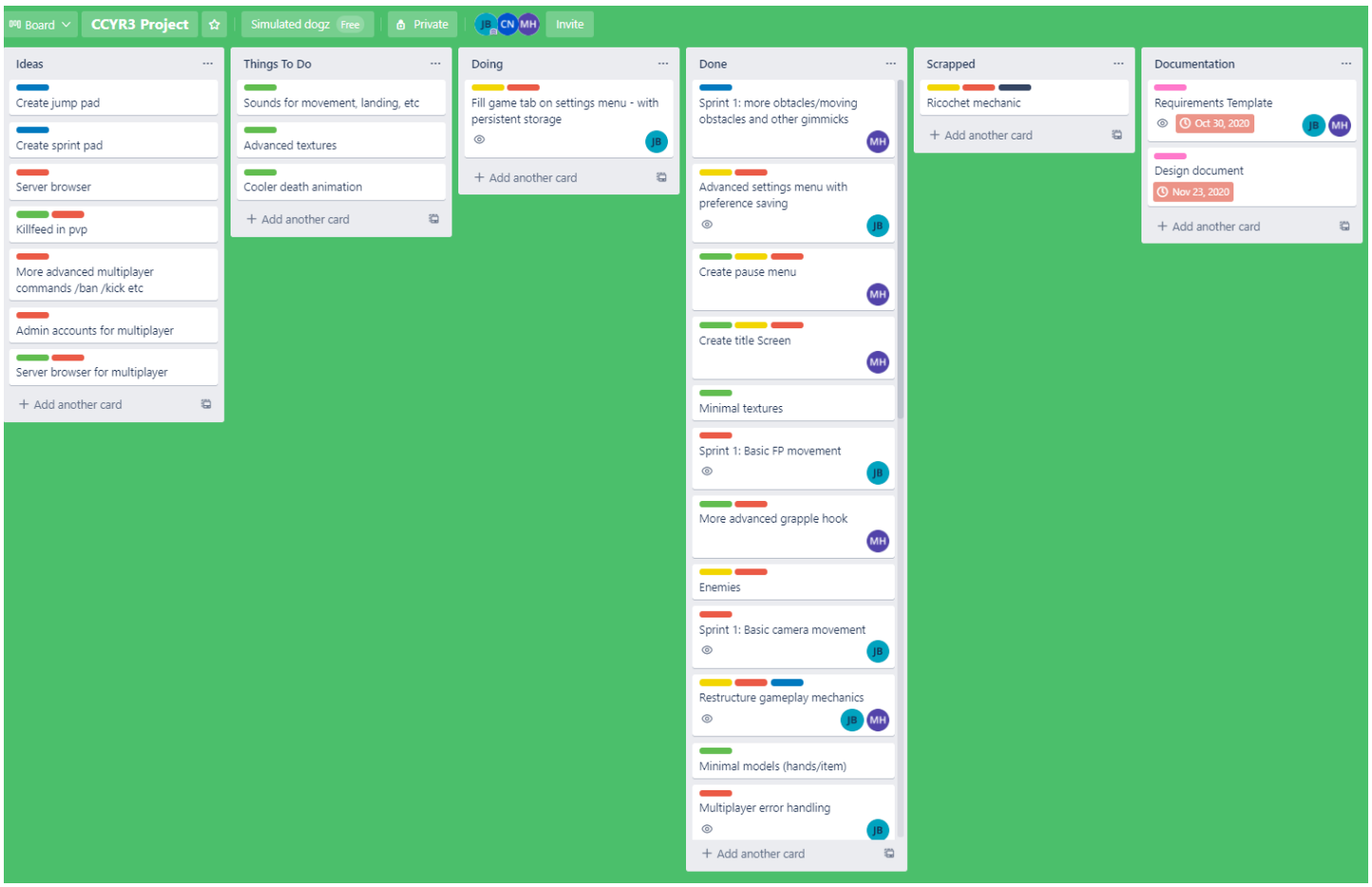


Figure 100 - Trello board for project

The Trello board worked quite well; however, it was only me updating it after a while. I still got good use out of it, sometimes I would forget an idea that I had, and would check the Trello board to remember. It was good for planning what had been done and what needed to be done next. When I went to plan the next sprints, I would often go to the Trello board and decide on the next features that needed to be implemented.

Bugs relating to the code were not put on the Trello board, but instead on the GitHub issues page.

6.5.2 GitHub

GitHub is a cloud-based version control platform. It empowers developers to upload their code online, giving both a backup point and a place to revisit historical versions of the software. GitHub was used as the primary location for the logging of bugs and issues for the game. It was just me logging bugs there, though.

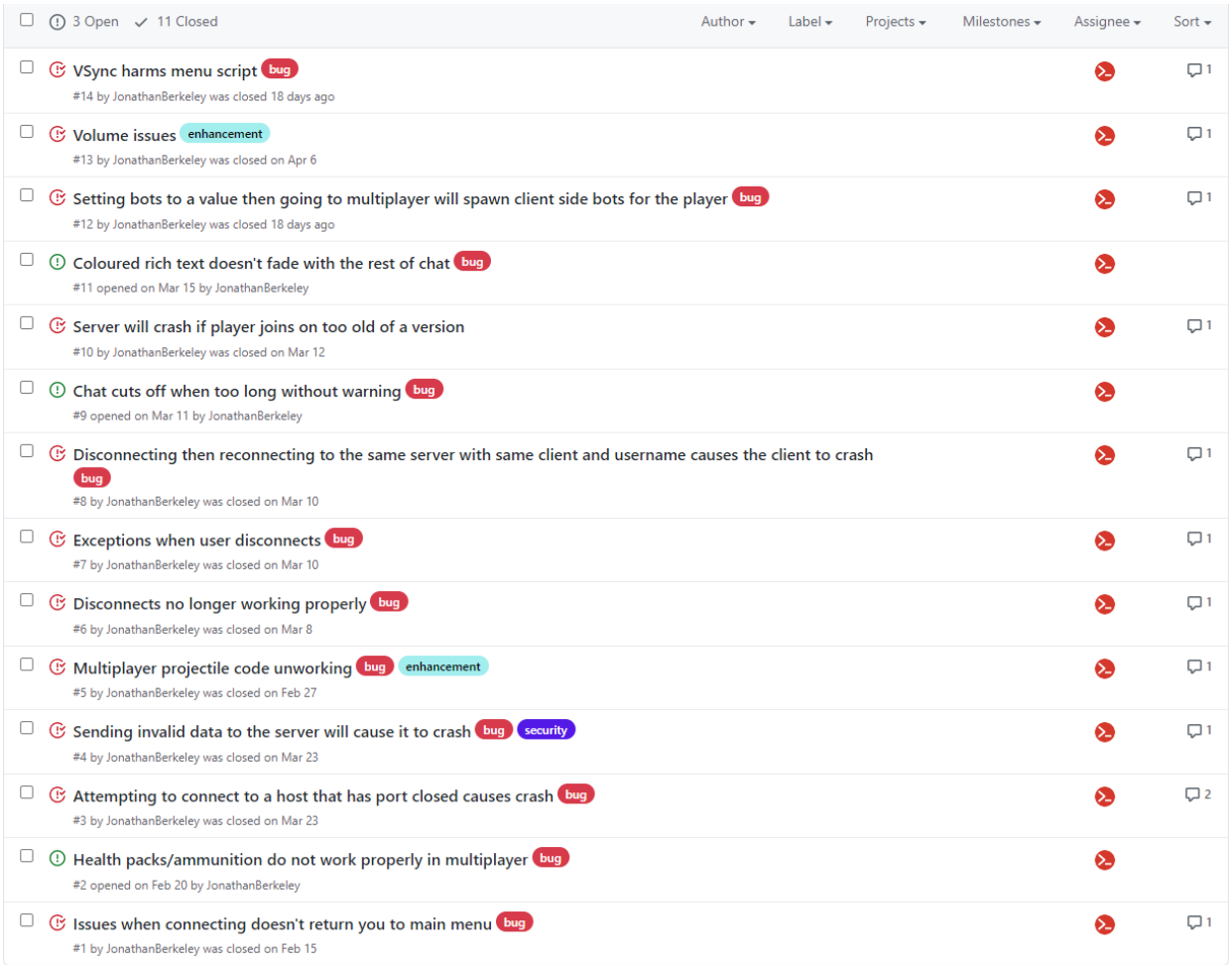


Figure 101 - Portion of issues page on GitHub

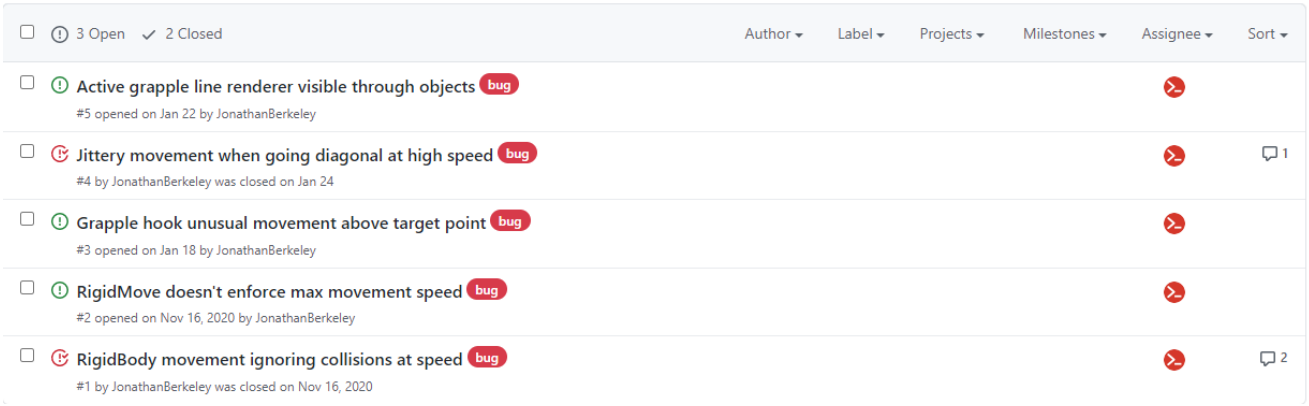


Figure 102 - Separate page on another repository for combined project.

GitHub was used extensively during this project and has just shy of 200 commits in total across the 3 repositories.

The version control software Unity Collaborate was also used extensively, since it allows rapid syncing of data to team members without hassle. There are over 100 commits on Unity Collaborate.

6.5.3 Journal

I kept a very detailed journal for this project. I updated it as often as I could think to after I did work. It was very useful in writing this report, so I could recall the exact date things were fixed. The journal is 1,363 words in length and has been kept up to date since the start of the project.

I found it easy to write a small summary of what had been changed at the end of each development session, it made it easier if I came back to the code a period of time later to know exactly what I was working on when I left off.

I considered pasting the Journal here into the report, but it took up far too much space.

6.6 Reflection

6.6.1 Views on the project

I think that the project went very well, I learned an incredible amount from the development of this game. Before this I had not ever tinkered with networking code, and by the end I had a working server with multiple players playing at the same time.

It was such a satisfying feeling to finally get it working, and to invite my friends on to play something that I had put time and effort into creating.

There have been many mistakes along the way, which I will undoubtably learn from in my future endeavours.

It was my first-time developing software under this SCRUM methodology, but I believe that it went very well. I had a lot of responsibility in the group, but it was an accomplishing feeling to get things completed for the project.

6.6.2 Completing a large software development project

I learned the methodology and the systems used in developing a large software project. Especially important I found it was to maintain a rigorous attention to the neatness of both the code, and the folder structure of the project. I regularly made sure that the code was neat and commented and deleted unused assets and files from the game. The clutter makes it harder to navigate in the future and is a nightmare to debug messy code months in the future if there is no comments or proper formatting.

I have covered the full stack of development for this project, covering design and low-level code, and I learned about the level of development that goes into projects of this size.

6.6.3 Working in a team.

Although the teamwork could have been better for this project, I think that my commitment to the GitHub and Trello, as well as clear commented code demonstrates that I am capable of working in a team. Following through with these actions gave me an appreciation of the extra work and effort it takes to work in a team.

6.6.4 Working with a supervisor

This has been a great experience; I really enjoyed the supervisor meetings. It is very encouraging to have someone to talk about your progress with, especially when you are enthusiastic about the product. The added support and information was invaluable during the projects development and I learned about the communication requirements from developer to supervisor from this project.

6.6.5 Technical skills

Prior to this year, I had little Unity experience. I had absolutely no Blender experience. I had no C# experience, nor any networking code experience or knowledge.

Through the modules such as Game Development, and Computer Networks, I learned rapidly the technical skills needed for this project. I continued my learning in the specific area of game networking code in online tutorials, and I now have a firm understanding of C#, video-game networking code, and Unity. I have a moderate understanding of Blender now; but it is still quite difficult to use.

6.6.6 Further competencies and skills

Learning how to use GitHub issues tab and tracking problems that way. This would have been alien to me before this year, but I am well used to it now. Alongside this, I kept a log of versions of the program with what had changes. I decided to version my code and document the changes after I kept getting confused which version was which on the remote Google server. I learned to appropriately version and document code.

6.7 Conclusion

This chapter, in conclusion, has discussed the intricate details of the project management for this game. The issues and strongpoints have been discussed, as well as the tools that were crucial in the development of the program.

The learning outcomes for the project have been discussed alongside the teamwork aspects. There has been a lot to learn from the project and about large-scale software development in general.

7 Business Opportunities

1. Executive Summary

Purpose of the plan

The purpose of this business plan is to find and analyse all the opportunities and pitfalls of this business. It will aim to figure out how feasible it is to start up the business.

Product or service and its advantages

This game is a 3D motion-based physics game with a focus on using momentum to reach otherwise inaccessible areas of the game. The player's objective is to reach a specific objective in the level, they will use tools such as a grapple hook to achieve this. At its core, it is a singleplayer/co-op puzzle game designed for entertainment. It has direct advantage over other games in the market as it is cheaper and contains more content with unique features that cannot be found on the market currently.

The name of the game is Inertia, this name is a reference to the momentum aspect of the game which is required to progress. It is short and memorable, following the naming conventions of similar games of this genre such as Refunct, Portal, and Karlson. The name Inertia also has a subtle reference to the skill requirement of the game, as the word derives from the latin *iners*, meaning unskilled (wikitionary, n.d.).

Market opportunity

Our target customers are gamers, people that play videogames, especially targeting gamers that are interested in variety gaming (playing many different games rather than just one). There is not much currently in the market for cheap movement-based video games, particularly ones that focus on popular movement mechanics that feature as side mechanics of other mainstream games. The games that do exist have a more limited scope than this game and focus on other aspects such as graphics.

Management team

This business is being managed by primarily two people, Jonathan Berkeley, and Mark Hurley. This small management team is typical of indie game development studios, which typically consist of less than 10 people. This is also advantageous as it reduces the cost of salaries massively.

Financial projections

	Year 1
Sales	€60,030
Net Profit before Tax	€8,455
Investment	€2,000
Employment	€39,500

The figures and the origins for these figures are explained in the Cash Flow Profit Loss excel sheet shown in Chapter 9 of this document. They are based on games of a similar scope and development (Refunct, Karlson), by taking their initial sales and price ranges into account, as well as the advertising effort behind the games.

Funding requirements

The game will need 3,500 Euro in funding for the first year, this funding will go towards professional assets and models for the game, as well as for the initial funding of the game's multiplayer servers, after this initial funding period, the sales from the game should fund further development of the game so further funding should be unnecessary. This funding also covers the cost of getting the game on to the Steam store, which is approximately 100$ (82 Euro) (Xsolla, 2020).

2. Problem & Solution

Products and services

The product being developed is a single player 3D movement-based physics game, where the player assumes control of an unnamed test subject who is armed with a grappling gun or rocket launcher. This unnamed test subject has the objective of escaping the laboratory they are trapped in by solving various puzzles using the tools available to them and their wits.

The game is very beginner friendly but can also be a challenge for those who have are interested in beating the game in the fastest time possible. The game requires the player to make their way from one end of the level to the other while jumping across and around platforms. It will be a relatively short game with the main focus being on the unique gameplay and level design. The game is targeting the entertainment market, specifically the video gaming sector.

This game will be made in the Unity game engine, a free development environment for video games that has been used for many Triple-A video game releases such as “Hearthstone”, “Cities: Skylines” and many other major video games (https://en.wikipedia.org/wiki/List_of_Unity_games). The game will be coded in the C sharp (C#) programming language, targeting Windows OS.

Compared to our competitors our product is cheaper and has a high level of quality for its price.

“Pain” and ‘Cure’ provided by this game

The pain we are curing with our product is the lack of innovative puzzle games that are both challenging and easy to pick up.

Another problem we are solving is for the people who are bored playing their current games and are looking for variety.

There are few games with this focus on momentum mechanics, the unique gameplay will attract gamers that are looking for a change from the regular game releases.

Features and Benefits

This game will feature detailed levels, 3D models, objectives, progression, 3D movement mechanics, grapple mechanics, an interactive UI, and a compelling story.

The benefits of this product are that it is cheap and easy to learn but hard to master and it is entertaining.

Another benefit of this game is the long-term support we will be giving it, we will be adding more content after the release, following the successful business model of other games in the market.

Business Model and competitor profile

The main revenue will be generated from the initial purchases when the game releases. The game will have an initial price of 4 euros, available on the Steam store (https://store.steampowered.com/). This price is intended to drop to 2 euros during the Steam summer sales event, and to 1 euro during the Steam Christmas sales. At all other times, the game will remain at the 4-euro price point, keeping it under the 5 dollars category for the American market. There is a tax of 10% on the sale of games on the steam market for indie games, which has been factored into pricing (Simon Carless, 2020).

The pricing for this game was decided based on similar games with a similar scope, comparing our game to other independent game releases such as Refunct and Karlson, which were inspiration for this game. The pricing for Refunct is 2.99$ (approx. 2.50 Euro), it was mainly developed by one person, and is much shorter than our game with far less game time. This game was very successful with between 200,000 and 500,000 owners (steamspy, n.d.) (steamdb, n.d.), at a conservative estimate, and higher estimates at up to 1,300,000 (playtracker, n.d.). Karlson, an unreleased game with a similar theme and just one developer, is on the Steam wish list of 100,000 gamers (DaniDevYT, 2020), indicating intention to buy on release, at a price point of around 10$ (https://store.steampowered.com/app/1228610/KARLSON/).

Later down the line after the release of the game, further content would be added through DLC (Downloadable Content) as is very common in the gaming market. These DLCs would add extra content for current owners of the game such as levels and mechanics, without the need for an entire new game, for a low price.

Further down the line a sequel to the game would also be considered, depending on the success of the first one. The sequel could be a direct improvement building on the success of the first, with a larger budget behind it. Since the game has a multiplayer aspect to it, there will be some cheap micro-transactions available to the players, where they can pay a small price for in-game items that would be displayed to other players.

The players can then trade these items between each other. This concept is taken from games such as Counter Strike: Global Offensive, Team Fortress 2, and Fortnite which does this exact business model and generate billions of USD in revenue (Nick Yopko, 2020) (Christina Gough, 2018).

Unique Selling Points

The unique selling point is that this game has a unique gameplay experience compare to other games in the market. There is no other game offering this exact type of gameplay. It has great value, it's far cheaper than the competitors' products as it is costing us much less to produce. It will also run-on lower end machines than alternative games that focus on graphics, giving us a competitive edge in the budget gaming sector, especially the under 5-dollar section in the Steam store (https://store.steampowered.com/search/?filter=ut2).

Disadvantages and weak points

The main disadvantage of this game is that it is an indie game targeting a niche market, this is both a blessing and a curse. It is an advantage in some sense as it's more likely to get traction with the gamers that enjoy that niche, but also a disadvantage as it will not have the same general appeal that other games do.

Innovation evaluation matrix

	Time	Cost	Potential Income	Monetary Impact	
Criteria Coefficient	5	4	3	4	
Post launch content(DLC)	3	2	4	5	
Weighted Rating	5x3=15	4x2=8	3x4=12	4x5=20	
(cosmetics) Microtransactions	1	1	4	4	
Weighted Rating	5x1=5	4x1=4	3x4=12	4x4=16	
Multiplayer	5	3	2	5	
Weighted Rating	5x5=25	4x3=12	3x2=6	4x5=20	

3. Target Market & Competition

	Your Idea (Name of Idea)	COMPETITOR 1	COMPETITOR 2
SUMMARY describe what you already know about your competitors	Inertia	Portal 2	Refunct
TARGET CUSTOMERS	All kinds of gamers	All kinds of gamers	Casual gamers
Feature 1 (Price)	€3.99	€8.19	€2.52
Feature 2 (service)	Video Game	Video Game	Video Game
STRENGTHS	• Cheap • Casual experience • Unique gameplay	• Unique gameplay • Compelling story • Cinematic	• Beautiful visuals • Cheap and Cheerful • Relaxing
WEAKNESSES	• Not completed • Low Budget • Very Simplistic	• Fixed story • Limited amount of playtime • Short game	• No story • Very short game • Not exciting
COMPETITIVE ADVANTAGE	• Unique gameplay (grapple hook, rocket launcher) • Cheap • Indie game appealing to variety gamers	• Unique gameplay (portal gun) • Memorable characters and story • Has a cult following	• Cheap • Unique gameplay (satisfying movement) • Memorable visuals

Target Market

The target market for this game is anyone who has an interest in games as this is very beginner friendly but also offers a challenge to more experienced gamers. There is a particular market of gamers that shops for variety games, games that differ from the typical beaten path of game creation. These gamers want to play games with unique attributes and features, which the game Inertia contains in abundance.

This game specifically fits in to that niche and appeals to those types of gamers by being low risk to try, Steam offers a customer-first refund system, so being a lesser-known game development studio will not be an issue.

To be more specific on demographics of the target market, the target market for this game is younger audience, from 13 (The lowest age a user can own an independent Steam account) – 35. This is the general age range for gamers in skill-based games such as Counter Strike: Global Offensive (Christina Gough, 2020) (espn.co.uk, 2017).

Going by the recommendations of similar games such as the Portal series, it would be recommended to be only allowed for players of ages 11 onwards (imdb, 2007).

There are many gamers that prefer to play Indie games over mainstream releases due to the unique factors that indie games often have; this business would intend to tap into that market with this unique game.

The niche area of this game is growing, along with the rest of the gaming market which will be further developed upon under the next heading. Other available games with a focus on skill-based movement mechanics have had a recent huge surge in popularity, showing the markets interest in this area (Nathaniel Mott, 2019).

Market trends

The gaming market is rapidly growing (GrandViewResearch, 2020), which in turn gives the game longevity to keep generating revenue even after development on the game has ceased. The amount of servers hosted can decrease or increase depending on how many users are currently online, which means the cost of continuing to host the servers will always be profitable, as if there were no users online in our game, there would be no servers hosted, and therefore no hosting costs. The server hosting pricing is done on a usage basis.

The game genres that are experiencing the most rapid growth share very similar attributes to our game, being generally 3D first person games, containing some shooting mechanic (statistica, 2019).

As the market progresses, we intend to follow up with DLC (Downloadable Content) that targets any new progression in technology or the market. Such as possible support for virtual reality devices

Competitors

The main competitors would include the games Portal/Portal 2, Refunct, the Stanley Parable, Karlson and The Witness. These games are the most comparable in genre and the type of gamers that they would attract. The games Refunct and Karlson are particularly focused on in the market research for this business, as they are both developed by 1 developer, giving a more realistic analysis for market share than, for example, the Portal series, which has many developers creating the game under a multi-billion-dollar studio.

What strategies do they use to compete in the market?

1. They have the resources to invest in large advertisement campaigns for their product
2. They have much more resources and time to craft an experience for the players

The competitor profile for these games was explored previously, under the Business Model / competitor profile header. These games directly affected how we priced and targeted the game.

Pitfalls / Weaknesses

1. There are better equipped competitors.
2. This business has a smaller advertising budget.
3. There is an overabundance of indie games, will require an advertising effort to stand out.

Pros:

- Low risk.
- Main investment is time, not a large monetary investment.
- Easy to expand development on in future.
- Easy to roll out to users with the available launchers and assistant software for video games (Such as Steam).
- Potential for sequels and remakes of the game in future for further business potential.

Cons:

- Not likely to generate a large amount of revenue.
- Video game market is very saturated.
- This video game occupies a niche market.

4. Marketing/Sales Plan

GOAL	TARGET	STRATEGIES	TACTICS / MESSAGES	CALENDAR	MEASUREMENT
Develop a game that can be enjoyed and played by casual and hardcore players	International people who are young, male/female who are looking to have fun and are interested in gaming	Deals and special offers on industry leading platforms such as Steam to attract variety gamers.	We are using the industry leading platform Steam to bring our game to a wider audience. 2 man Indie game dev team that are making a modern and fun game for all to play and enjoy casually or competitively. The affordable cost and easy to jump into game will hopefully persuade our targets to check the game out.	Daily discussion and development takes place where we add new components into the game in a series of sprints, these sprints are bi weekly where we complete a major game component.	We would track our results through user surveys, feedback and interviews. We meet with our project supervisor to deliberate features and discuss feedback.

Marketing strategy

The marketing strategy for this game is simple, the game will be published on the Steam store, which incurs a fee of 100$ as previously mentioned. The game will aim at the cheap indie game market, this is where gamers will go to buy cheap games that they can play for a few hours to get an entertaining break from the regular Triple-A releases with some unique concepts and gameplay. The customers are aware that if the game is not as advertised, they can get a refund through steam easily, so there is no issue of trust.

The game will go along with the Steam sales in summer and winter, Steam features games that are on sale, essentially giving free advertising. The more players that play our game and positively review it, the more Steam will recommend the game to other players.

It is crucial to get an initial player-base to kick off this domino-effect spread of this game, and that is where the advertising budget will be used. The main advertising method for the game will be to pay popular online streamers to play the game on stream for a certain amount of time. This is an effective method of advertising used by many games such as Fortnite and Apex Legends, among many others, to rapidly spread popularity of a game (Vincent Genova, 2019). This is effective as it is exposing the game to people that are already interested in gaming, as they are watching a streamer play video game. Paying extremely popular streamers such as Apex Legends did is far outside of the budget, and our advertisement would focus on smaller, yet still popular, streamers.

5. Operations

Operations

- To get this game operational we first bought an internet domain name for the game.
- This establishes a central location for customers to find information about our company and its products.
- We also paid for internet hosting for 1 year to host the site and game servers.
- Our plan is to set up a private limited company for the furthered development of this game and future games/content.
- To do this we will need to pay the registration fee, which will cost approximately 150 to do online.
- It is nearly 600 Euro to cover the costs of the formation of the company with the help of specialists.
- Having a limited company will reduce risk involved.
- There is lower tax on profits for a limited company as well since sole traders are taxed on an individual income tax rate.
- The game will be developed from home, and there is no central office. This cuts down on unnecessary overhead. Indie game development thrives in work from home settings.
- The current testing game servers are being hosted on Googles VM Instance service, running Windows Server 2012 with custom security settings. This is temporary but gives an idea for server running costs.
- There is no intention to buy servers, only to rent server slots with an on-demand basis, this ensures that the game does not go into loss by overbooking (or under booking) servers.

Production workflow

Our product will be made using the Unity game engine. The player takes control of a character from the first-person view and must solve various puzzles using a grappling hook to advance in various levels. We create the game by using and placing assets in the game engine and then applying code to those assets. This game will be coded using the C# sharp programming language. The various problems we could encounter would be game breaking bugs that stop the player from playing/advancing in the game in one way or another.

The production of the game is managed through a private GitHub repository. This allows for logging of issues and more rapid development of the game, with coordinated elimination of bugs and issues as they appear. The game also has a Trello board with staged deadlines, encouraging continued development and preventing the games development from stagnating.

Industry association memberships

Our business would be set up as a Limited Company.

Supply chains:

Our suppliers would include Unity who provide us with the Engine we need to create our game and are also the ones that supply us with assets used in said game. Unity provides these to us for free so long as our company does not make more than $100,000 a year and we include the unity logo in our games intro. If our supplier Unity lets us down, we can switch development to the Unreal Engine for a price.

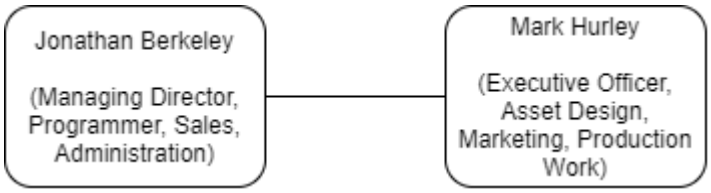
6. Team and Company

Ownership Structure

We are operating our business under a limited liability company known as **Earthshatter Entertainment.** The ownership is split 50/50 between the two owners of the company and any change of that would be due to future investments.

Management Team

Organization Chart



Jonathan Berkeley is the Managing Director, Head Programmer, Sales Head and Administration Officer. Jonathan Berkeley previously volunteered in Coder Dojo, teaching kids to code for a year. Then he worked in paid employment teaching kids to code at a summer camp in Dublin city for the duration of summer. This has given him a lot of experience in programming, and teaching programming to others. He currently is in education at the Institute of Art Design and Technology Dún Laoghaire (IADT), studying Creative Computing, to further develop his programming and asset development skills.

Mark Hurley is the Executive Officer, Asset Design Lead, Marketing Head and Production Work Officer. He is currently a student in IADT studying creative computing but has already taken on the responsibilities that come with running a business as he is a fast learner and is eager to tackle any challenges that may arise.

Board of Advisors:

Tim McNichols - Financial advisor

Susan Reardon – Design advisor

Catherine Noonan – General advisor

For the future, there is currently no plan to hire additional staff. The scope of this game and its forecasted income would not support additional fulltime staff. Instead, further development would be driven by the existing staff, pushing updates and new content to the game to keep it relevant on the gaming market. Accounts and finance are managed internally in the company for the first year, later moving to being managed using online resources and budget management services.

7. Financial Projections

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1						**CashFlow Forecast (1 Year)**									
2		*Pre Start*	*Apr*	*May*	*Jun*	*Jul*	*Aug*	*Sep*	*Oct*	*Nov*	*Dec*	*Jan*	*Feb*	*Mar*	*Total*
3	*Income*														**€ -**
4	Sales		€ 3,200	€ *1,840*	€ *4,000*	€ *9,000*	€ *7,500*	€ *4,800*	€ *6,200*	€ *4,200*	€ *9,800*	€ *3,090*	€ *2,500*	€ *3,900*	**€ 60,030**
5	Loans	€ 2,500				€ 500			€ 500						**€ 3,500**
6	Grants						€ 2,000								**€ 2,000**
7	Capital Introduced	€ 2,000													**€ 2,000**
8	**Total Income**	**€ 4,500**	**€ 3,200**	**€ 1,840**	**€ 4,000**	**€ 9,500**	**€ 9,500**	**€ 4,800**	**€ 6,700**	**€ 4,200**	**€ 9,800**	**€ 3,090**	**€ 2,500**	**€ 3,900**	**€ 67,530**
9	*Expenses*														
10	Setup costs	€ 100													**€ 100**
11	Materials	€ 200												€ 50	**€ 250**
12	Capital Equipment	€ 2,000													**€ 2,000**
13	Rent & Rates	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	€ 40	**€ 520**
14	Power, Heat, Light	€ 60		€ 60		€ 60		€ 60		€ 60		€ 60			**€ 360**
15	Steam sales tax		€ 320	€ 184	€ 400	€ 900	€ 750	€ 480	€ 620	€ 420	€ 980	€ 309	€ 250	€ 390	**€ 6,003**
16	Server costs/ISP hosting	€ 250	€ 250	€ 200	€ 250	€ 400	€ 350	€ 250	€ 350	€ 250	€ 400	€ 250	€ 250	€ 250	**€ 3,700**
17	Travel/ Vehicle Cost	€ 20			€ 20			€ 20		€ 20			€ 20		**€ 100**
18	**Advertising/ marketing**	€ 300		€ 300			€ 300			€ 300			€ 300		**€ 1,500**
19	Legal Fees / refunds		€ 20	€ 10	€ 20	€ 40	€ 40	€ 20	€ 30	€ 20	€ 50	€ 30	€ 30	€ 30	**€ 340**
20	Bank Loan		€ 208	€ 208	€ 208	€ 252	€ 252	€ 252	€ 296	€ 296	€ 296	€ 296	€ 296	€ 296	**€ 3,157**
21	Bank Loan Interest		€ 11	€ 11	€ 11	€ 22	€ 22	€ 22	€ 33	€ 33	€ 33	€ 33	€ 33	€ 33	**€ 297**
22	Drawings (Salaries)	€ 1,500	€ 3,000	€ 3,000	€ 3,000	€ 3,000	€ 3,000	€ 3,000	€ 3,000	€ 3,000	€ 5,000	€ 3,000	€ 3,000	€ 3,000	**€ 39,500**
23	**Total Expenditure**	**€ 4,470**	**€ 3,849**	**€ 4,013**	**€ 3,949**	**€ 4,714**	**€ 4,754**	**€ 4,144**	**€ 4,369**	**€ 4,439**	**€ 6,799**	**€ 4,018**	**€ 4,219**	**€ 4,089**	**€ 57,827**
24	Net Cashflow	30	-649.3	-2173.3	50.7	4786	4746	656	2331	-239	3001	-928	-1719	-189	
25	Opening Balance	0	30	-619.3	-2792.6	-2741.9	2044.1	6790.1	7446.1	9777.1	9538.1	12539	11611	9892.1	
26	Closing Balance	30	-619.3	-2792.6	-2741.9	2044.1	6790.1	7446.1	9777.1	9538.1	12539	11611	9892.1	**€ 9,703**	
27															

Figure 103- Cashflow forecast

Q	R	S	T
Profit & Loss A/C			
Sales		€ 60,030	
- Cost of Goods Sold		-€ 100	
Gross Profit		**€ 59,930**	
-Expenses		€ 57,477	
Net Profit (Loss)		**€ 2,453**	

Figure 104 - Profit & Loss A/C

	A	B
28	*Sales Assumptions	Refunct, a competitor game to ours with a similar theme has between 200,000 and 500,000 owners, at a conservative estimate.
29		Other sources report up to 1,300,000 owners of the game.
30		src: https://playtracker.net/insight/game/5114
31		src: https://steamspy.com/app/406150
32		src: https://steamdb.info/app/406150/graphs/
33		
34		Refunct sells for $2.99. (Approx 2.50 euro)
35		Our game would be priced at 4.00 euro. Steam takes a 10% tax on sales which we accounted for in the Steam sales tax column.
36		
37		Using information from https://steamdb.info/app/406150/graphs/ , a good way to get an approximate ownership
38		count for a game on steam is to multiply the review count at that time by 20 for a lower range estimate (multiply by 60 for high range estimate).
39		In the first month Refunct was released, it received 84 reviews. 84 x 20 = 1,680. (Multiplying by price gives us 1,680 * 2.99 = $5,023.2 (4,200 euro))
40		This gives us an approximate number of players for the first month of release of a similar indie game. And some sales information for a similar scope of game.

Figure 3 - Sales assumptions

The next month after Refunct's release (1st January 2016 - February 1st 2016), there was 49 new reviews, giving us an estimate of 980 new players. (2,450 euro in revenue)

The following month (February 1st 2016 - March 1st 2016), there was 94 new reviews, giving us an estimate of 1,880 new players. (2,450 euro in revenue)

From (April 1st 2016 - May 1st 2016), there was 345 new reviews, giving us an estimate of 6,900 new players. (17,250 euro in revenue)

From (December 1st 2017- January 1st 2017), there was 951 new reviews, giving us an estimate of 19,020 new players.
However, the game was on sale for half price. (23,775 euro in revenue)

From (February 1st 2018 - March 1st 2018), there was 160 new reviews, giving us an estimate of 3,200 new players. (8,000 euro in revenue)

From (August 1st 2020 - September 1st 2020), there was 121 new reviews, giving us an estimate of 2,420 new players. (6,050 euro in revenue)

Unlike other European countries, Ireland does not have any national organisation or body providing specific supports for the games sector. A recent government report highlighted this deficit and recommended the establishment of a game prototyping fund.
https://fiercefun.com/game-development-funding-in-ireland/#:~:text=Unlike%20other%20European%20countries%2C%20Ireland,of%20a%20game%20prototyping%20fund.

Steam tax on sales is between 10% - 20%
https://www.gamasutra.com/blogs/SimonCarless/20200817/368366/Game_refunds__the_hidden_costs_of_getting_to_net_on_Steam.php#:~:text=But%20actually%2C%20it's%20pretty%20straightforward,a%20player%20in%20their%20country.

Steam Publishing Costs
Steam charges a fee of $100 for each game you submit on Steam Direct. Although nonrefundable, this fee can be recouped once your product has generated at least $1,000 adjusted gross revenue from the Steam Store and in-app purchases.
https://xsolla.com/blog/monetization/2206/self-publishing-on-steam-the-ultimate-guide#:~:text=Steam%20Publishing%20Costs,Store%20and%20in%2Dapp%20purchases.

Loans are for marketing & assets from the unity asset store for use in the game.

Comparable competitor game link on steam store: https://store.steampowered.com/app/406150/Refunct/

Figure 4 - Sales assumptions

Refunct, a previously mentioned competitor game with a similar theme, has between 200,000 and 500,000 owners, at a conservative estimate.

Other sources report up to 1,300,000 owners of the game. (playtracker, n.d.) (steamspy, n.d.) (steamdb, n.d.)

Refunct sells for $2.99 (Approx 2.50 euro).

Our game would be priced at 4.00 euro. Steam takes a 10% tax on sales which we accounted for in the Steam sales tax column.

Using information from https://steamdb.info/app/406150/graphs/ , a good way to get an approximate ownership

count for a game on steam is to multiply the review count at that time by 20 for a lower range estimate (multiply by 60 for high range estimate).

In the first month Refunct was released, it received 84 reviews. 84 x 20 = 1,680. (Multiplying by price gives us 1,680 * 2.99 = $5,023.2 (4,200 euro))

This gives us an approximate number of players for the first month of release of a similar indie game. And some sales information for a similar scope of game.

The next month after Refunct's release (1st January, 2016 - February 1st, 2016), there was 49 new reviews, giving us an estimate of 980 new players. (2,450 euro in revenue)

The following month (February 1st, 2016 - March 1st, 2016), there was 94 new reviews, giving us an estimate of 1,880 new players. (2,450 euro in revenue)

From (April 1st, 2016 - May 1st, 2016), there was 345 new reviews, giving us an estimate of 6,900 new players. (17,250 euro in revenue)

From (December 1st, 2017- January 1st, 2017), there was 951 new reviews, giving us an estimate of 19,020 new players.

However, the game was on sale for half price. (23,775 euro in revenue)

From (February 1st, 2018 - March 1st, 2018), there was 160 new reviews, giving us an estimate of 3,200 new players. (8,000 euro in revenue)

From (August 1st, 2020 - September 1st, 2020), there was 121 new reviews, giving us an estimate of 2,420 new players. (6,050 euro in revenue)

Karlson is another previously mentioned unreleased game with a similar theme and just one developer. Karlson is on the Steam wish list of over 100,000 gamers (DaniDevYT, 2020), indicating intention to buy on release, at a price point of around 10$ (https://store.steampowered.com/app/1228610/KARLSON/). Although this game has a much more complex advertising campaign, with the developer offering free tutorials in exchange for publicity to his game, so that was taken into account.

“Unlike other European countries, Ireland does not have any national organisation or body providing specific supports for the games sector. A recent government report highlighted this deficit and recommended the establishment of a game prototyping fund.” – Quote from (gamesdude, 2019)

As previously mentioned, Steam tax on sales is between 10% - 20%, which is where the figure for steam taxes comes from, approximating that our game would be on the lowest end of the scale due to no additional tax on games for the Irish market, and that our game is an indie game. (Simon Carless, 2020)

Steam Publishing Costs

Steam charges a fee of $100 for each game you submit on Steam Direct. Although non-refundable, this fee can be recouped once your product has generated at least $1,000 adjusted gross revenue from the Steam Store and in-app purchases.

(Xsolla, 2020)

Loans are for marketing & assets from the unity asset store for use in the game.

Links to competitor games:

https://store.steampowered.com/app/406150/Refunct/

https://store.steampowered.com/app/1228610/KARLSON/

8. Funding requirements

The game will require 3,500 Euro of initial funding. This will be spent on advertising, assets for the game, server hosting costs, as well as costs of publishing the game on Steam. Firstly, the business will attempt to receive a grant from the government of Ireland through schemes such as Enterprise Ireland.

However, the business has already prepared for this to fall through, and the financial cash-flow sheets display a 1-year bank loan at 13% interest for the initial 2,500. Along with a following two sums of 500 in July and October, these will cover the advertising leading up to the off-school holiday periods that are hotspots for game sales in Summer and Winter, those are calculated at a 9.9% interest as a 1-year loan each of 500. Ideally, this can be avoided, and external investment can be secured for the game at lower rates.

In the worst-case scenario, where the game cannot secure any funding, the game can still be released in a non-multiplayer state, then multiplayer can be rolled out as revenue from sales comes in.

8 Conclusion

The background for this project comes from Team Fortress 2, and the game Quake 2. These are classic games that inspired the development of this game. The concept of firing an explosive at your feet in a game to gain momentum comes from those games.

The aims for the project were to make a fun multiplayer game with the same spirit as the games previously mentioned, one that focuses on momentum and speed in a 3D world. As well as being more accessible than those games, by being simpler yet still enjoyable.

Research played a key role in the development of this game; without it the server code would not be anywhere near the stage it is. The free informational resources of the incredibly generous online developer community has been a key factor in the development of the game.

Design was important for the game; the design of the game shapes the personality of it. By not taking itself too seriously and adopting surrealistic and cartoonish appearance, it can be made clear that the game is not for realism but for enjoyment.

The implementation stage for this game was the most work intensive and demanding stage, here the game was put together piece by piece through the SCRUM methodology, with frequent reviews, meetings, updates, bug tracking and fixes which were essential to the game's successful completion.

The testing stage for the game made the problems in the games design and playability obvious, issues which had not previously been seen or even thought of while developing had been discovered through the usability tests. After this testing period, they could be fixed so that the end user would receive a more polished product.

The overall result of the project was positive, the game is enjoyable, multiplayer games tend to have a lot of replayability especially when played with friends. A lot was learned during the development of this project and the skills developed are very useful.

The project management was handled well through the combined methods of GitHub's issue pages, the Trello board, the Miro board, the project Discord chat, and meetings, and the Unity Collaborate system. The SCRUM methodology made the large workload manageable and prevented the project form becoming overwhelming despite the large amount of work to be done.

So many useful skills were learned from this project. The main thing is the improvement of coding skills, it is a very good skill to have currently. The development of Unity games requires lots of code, and the experience gained from coding this game is very useful. With that are other skills such as code documentation, using Blender, developing networking code, using the Unity editor, and more.

The project could be further developed by adding more maps and game modes. New weapons could be added, a friend system could be introduced, where players could add friends that they play with. The servers could be hosted, and the game could be published on the Steam launcher to have real players try the game out and give feedback.

Bibliography

Christina Gough. (2018, October 26). *Counter-Strike: Global Offensive (CS:GO) revenue worldwide from 2015 to 2018*. Retrieved from Statistica.com: https://www.statista.com/statistics/808773/csgo-revenue/#:~:text=It%20won%20the%20fan's%20choice,U.S.%20dollars%20worldwide%20in%202018.&text=CS%3AGO%20is%20the%20fourth,as%20well%20as%20on%20PC.

Christina Gough. (2020, July 20). *Share of shooter video gamers worldwide as of December 2018, by age*. Retrieved from statistica.com: https://www.statista.com/statistics/1129298/age-distribution-shooter-video-gamers/

DaniDevYT. (2020, February 18). *twitter*. Retrieved from twitter.com: https://twitter.com/DaniDevYT/status/1229720467604418560

espn.co.uk. (2017, September 17). *Average age in esports vs. major sports*. Retrieved from espn.co.uk: https://www.espn.co.uk/esports/story/_/id/20733853/the-average-age-esports-versus-nfl-nba-mlb-nhl

gamesdude. (2019, March 9). *Game Development Funding in Ireland*. Retrieved from fiercefun: https://fiercefun.com/game-development-funding-in-ireland/#:~:text=Unlike%20other%20European%20countries%2C%20Ireland,of%20a%20game%20prototyping%20fund.

GrandViewResearch. (2020, May). *Video Game Market Size, Share & Trends Analysis Report* . Retrieved from grandviewresearch.com: https://www.grandviewresearch.com/industry-analysis/video-game-market#:~:text=The%20global%20video%20game%20market,12.9%25%20from%202020%20to%202027.&text=The%20rising%20inclination%20from%20physical,on%20hardware%20compatibility%20and%20efficiency.

imdb. (2007). *Parents Guide - Portal series*. Retrieved from imdb: https://www.imdb.com/title/tt1127708/parentalguide

Nathaniel Mott. (2019, February 12). *Apex Legends, Titanfall 2 Score Impressive Player Counts*. Retrieved from tom's hardware: https://www.tomshardware.com/uk/news/apex-legends-boosts-titanfall-2-player-count,38593.html#:~:text=Titanfall%202's%20Resurgence,million%20in%20less%20than%2072.

Nick Yopko. (2020, May 17). *EDM*. Retrieved from EDM.com: https://edm.com/industry/fortnite-one-billion-in-game-sales#:~:text=The%20battle%20royale%20heavyweight%20brought,sensation%20and%20EDM%20crossover%20platform.&text=Despite%20being%20completely%20free%2Dto,mobile%20version%20of%20the%20game.)

playtracker. (n.d.). *Refunct - Playtracker*. Retrieved from playtracker.net: https://playtracker.net/insight/game/5114

Simon Carless. (2020, August 8). *Steam tax on sales*. Retrieved from gamasutra.com: https://www.gamasutra.com/blogs/SimonCarless/20200817/368366/Game_refunds__the_hidden_costs_of_getting_to_net_on_Steam.php#:~:text=But%20actually%2C%20it's%20pretty%20straightforward,a%20player%20in%20their%20country.

statistica. (2019, May). *Genre breakdown of video game sales in the United States in 2018*. Retrieved from statistica: https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre/

steamdb. (n.d.). *Refunct - Steam DB*. Retrieved from steamdb.info: https://steamdb.info/app/406150/graphs/

steamspy. (n.d.). *Steamspy - Refunct*. Retrieved from steamspy.com: https://steamspy.com/app/406150

Vincent Genova. (2019, March 13). *EA allegedly paid Ninja and Shroud an insane amount of money to play Apex Legends*. Retrieved from dexerto.com: https://www.dexerto.com/entertainment/ea-allegedly-paid-ninja-and-shroud-an-insane-amount-of-money-to-play-apex-legends-449904/

wikitionary. (n.d.). *Inertia - wikitionary*. Retrieved from wikitionary.org: https://en.wiktionary.org/wiki/inertia

Xsolla. (2020, May 19). *SELF-PUBLISH ON STEAM: THE ULTIMATE GUIDE*. Retrieved from xsolla.com: https://xsolla.com/blog/monetization/2206/self-publishing-on-steam-the-ultimate-guide#:~:text=Steam%20Publishing%20Costs,Store%20and%20in%2Dapp%20purchases.