Lost Horizons - 3D Unity Game 2023

Game and Multimedia Environments (GAME)

Informatics Department

Name: Thanh Ngo Trung

Candidate Number: 230956

Supervisor: Dr Paul Newbury

Statement of Originality

This report is submitted as part requirement for the degree of Games and Multimedia Environments at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged. I hereby give permission for a copy of this report to be loaned out to students in future years

Signed: 

Abstract

This project aims to create a fun experience for various levels of video gamers by producing a space-themed survival game with the Unity game engine [1]. The major aspects of video games, which are game features, multimedia and art design, and game design, are analyzed in existing survival games where key ideas are used to influence this project. These game features consist of gathering and crafting items, building structures and fighting enemies. The multimedia and art design of games are studied and discussed of their effects on the nature and feeling of the game and game design concepts incudes the importance of storylines and objectives in games or designing an open world map with the use of procedural generation with voxels or the marching cubes algorithm. This report outlines the game's requirements to become complete and methods of implementing them with limiting factors such aa time constraints and limited resources compared to a team of developers. The project resulted in a complete and functioning game with all the essential traits of a survival game while leaving a lot of room for additional features to be added.

Acknowledgements

I would like to thank Dr Paul Newbury, my project supervisor, for his guidance and feedback throughout the project, as well as the participants who volunteered to test my game and provided significant feedback to create something I thoroughly enjoyed planning and making.

Content

1 Introduction

1.1 Project Aims and Objectives

In this project, a 3D survival video game based in space will be designed and implemented using the Unity game engine and will run on the Windows 10 operating system. In a survival game, the player's main goal is to survive, and most games even include a story line for the player to complete along with surviving, giving the player a purpose to survive and explore. In these games, the player can generally gather resources from the environment and use them to their benefits such as crafting other items to fight with or food to nourish the player. Another main aspect is exploration where the player can discover new things such as landmarks, resources, key story components or even enemies. The survival aspect is defined by enemies or hazards that slow the player down by posing a risk where the player will have to find methods to deal with or avoid them.

Using a game engine to create a game will help accelerate the development of the game as the depth of knowledge required to make a game is reduced by using one. This means that the developer will mainly just need to focus on writing the gameplay code and creating the game with the built-in tools. Game engines also include graphics rendering for 2D or 3D, a physics engine that supports collision detection, an audio engine to load and play sounds, scripting support to implement gameplay logic, a world object model, animation handling, and much more [2].

1.2 Problem Area

When creating a game, there are several aspects that may become a problem when trying to create the game such as having limited time to implement the desired features and not having the skill set to implement a certain feature of the game. These skill sets may include complex programming such as visual programming, modelling, and texturing, animating, creating sound effects and music, and designing and writing a story for the game.

1.2.1 Practical Skill Problems

Achieving high-level visual effects can be made easier using the Unity VFX graph or shader graph which uses a node graph architecture rather than programming which is easier to learn therefore more helpful when designing and implementing a game within a time limit. I have a decent amount of experience in modelling but generally, modelling can take a long time due to the mass number of objects needed in a game and the level of detail considered per object depending on the game, however, when designing a game with a low poly art style, modelling and texturing is a lot quicker which is beneficial. Creating music may also be a challenge as all the sound effects should generally have the same style of sound. I also have some experience in making soundtracks/music or sampling audio but for the sake of time, royalty-free sounds will mainly be used, and some may be sampled and changed to fit with the game.

1.2.2 Game Design Problems

The game will also require a purpose for the player to play whether that be a storyline to fulfill or to just have fun. Writing even a short story for a game can give the player a purpose for playing the game and "give them context as to what they are doing" [3]. When implementing a storyline, the first problem to overcome is to design one that fits with the game style and one that is not too long to implement. Derk talks about diverse ways to implement a storyline into a game [3, Step 6] such as cutscenes which may require animating and camera work knowledge, environments such as a crashed ship to give the history, enemies, allies and loading screens.

Designing the world for the game requires strategic placement of certain objects such as storyline checkpoints or puzzles or even enemies. Placing everything and deciding where to place it can be increasingly time-consuming when developing more world objects and is also essential when designing an open-world game as the player will be able to see everything as they can roam anywhere on the map other than beyond the "bounds of an open-world game will players be limited by geographic features" [4] which should also be considered.

1.3 Report overview

Chapter 2 reviews the BCS Code of Conduct and criteria from the Ethical Compliance form and how the project follows these guidelines when looking into testing methods and participants.

Chapter 3 outlines effective components and ideas from similar games that create a successful survival game and issues that would not be ideal for the game. This chapter also reviews the development environments and software that will help create the game.

Chapter 4 covers all the mandatory and desirable requirements for both the functional and non-functional requirements. Each requirement has a brief description to explain what they are and their importance in the game.

Chapter 5 discusses the overall design of the game's macrostructures and different software design patterns used to facilitate faster and easier development.

Chapter 6 outlines the implementation and purpose of each main feature in the game and how they were implemented.

Chapter 7 highlights the multimedia and overall art of the game and how they were created and the reason they were created in such ways.

Chapter 8 covers unit testing, requirement testing and user testing to ensure a functioning and entertaining game.

Chapter 9 examines the project's final state by discussing what went well, what did not go well and how a different approach could have resulted in a better game.

Chapter 10 discusses ideas that can be implemented along with or changed in the game and how this may be done for a better product.

2 Professional Ethical Considerations

2.1 BCS Code of Conduct

This project will follow the BCS Code of Conduct where the relevant sections are as follows:

1. Public Interest

> **a. have due regard for public health, privacy, security and wellbeing of others and the environment.**

The game poses no harmful or disturbing content as the theme of the game is light and family friendly. The only data held will be the player's saved world which they may choose to do so or not.

> **b. have due regard for the legitimate rights of Third Parties.**

Most of the game's assets will be self-made and very little will be from third parties. Those that are third-party content will be free and properly referenced.

> **c. conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement.**

The game character will have no previously mentioned characteristics, so no discrimination is possible. Any testing user willing to participate will be accepted.

> **d. promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.**

As the game takes place in outer space where the background may be dark and light in random areas there may be a small risk of epilepsy triggers, however, the game will be designed to be always bright and have no flashing lights. Voice acting will not be present to accommodate deaf users.

2. Professional Competence and Integrity

> **d. ensure that you have the knowledge and understanding of Legislation* and that you comply with such Legislation, in carrying out your professional responsibilities.**

No legislation will be broken in the project's lifetime. The PEGI rating of this game will be 7 as the game will have some mild forms of violence but no gore or any other adult content that is unsuitable for children.

> **e. respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.**

The game will be tested by many users often and feedback will be presented. Drafts of the report and prototyping of the game will be shared with my supervisor.

f. avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction.

The user testing will remain anonymous, and their feedback will not be commented on, only presented and used to refine the game. Playing the game does not require any intensive physical activity, only the use of a keyboard and mouse.

g. reject and will not make any offer of bribery or unethical inducement.

Feedback from user testing will be obtained without conversation or incentives.

2.2 Ethical Consideration

Ethical considerations will be required as this project will require feedback from tests from human participants. The project will meet all 12 criteria in the Ethical Compliance form and a signed copy is included in the appendix section of the report.

1. **Participants were not exposed to any risks greater than those encountered in their normal working life.**

The game does not expose any significant risks as there is only mild violence towards inanimate objects.

2. **The study materials were paper-based, or comprised software running on standard hardware.**

There will be a questionnaire that the participant can fill out after playing the game.

3. **All participants explicitly stated that they agreed to take part, and that their data could be used in the project.**

Participants that agree to take part in the user test will be informed what data will be collected and stored.

4. **No incentives were offered to the participants.**

Participants will not be incentivized to participate in the user test and will purely be voluntary.

5. **No information about the evaluation or materials was intentionally withheld from the participants.**

When the participants play the game, the game will have small tutorials that will lead the player to most functionalities. There will be some functionalities that the player will have to explore in the game to uncover, however, the player has all rights to do so and is not held back by the game.

6. **No participant was under the age of 18.**

All participants will be friends and family that are over 18 years old.

7. **No participant had a disability or impairment that may have limited their understanding or communication or capacity to consent.**

All participants will be friends and family with no known disabilities or impairments.

8. **Neither I nor my supervisor are in a position of authority or influence over any of the participants.**

Participants will be friends and family which I have no position of authority over. The test is purely gameplay which will rely on the participant alone and I will not have any influence over it (No information or instructions given prior to and during the test).

9. **All participants were informed that they could withdraw at any time.**

All participants will be informed that they may withdraw at any time they wish.

10. **All participants have been informed of my contact details, and the contact details of my supervisor.**

All participants will be offered the emails of both me and my supervisor after the test.

11. **The evaluation was described in detail with all of the participants at the beginning of the session, and participants were fully debriefed at the end of the session. All participants were given the opportunity to ask questions at both the beginning and end of the session.**

A script for briefing and debriefing will be used and include time for questions before and after the test.

12. **All the data collected from the participants is stored securely, and in an anonymous form.**

An online questionnaire will be used where all data collected will be kept anonymous, and the data will be saved on my secure personal computer.

3 Research and Related Games

A survival game generally means the player is set in a hostile or unknown environment where the player must find means to survive with what they can, but many existing games evolved this idea by adding some unique ideas and features and so research is done to explore the possibilities for the project. Different games take different approaches to the survival aspect of a game and multimedia such as the art style, sound design or game design including unique core functionalities. These attributes and ideas are assessed for their benefits for the project and development environments are discussed to help decide the most appropriate or desired approach to implementing these.

To get a clearer idea of what this project will be and include, similar projects are investigated, compared, and evaluated to see which would be beneficial in the project. The three games researched are The Forest, Subnautica and Astroneer where their full research and findings can be found in Appendix E.

3.1 Functionality and Features

3.1.1 Difficulty

One common feature that The Forest and Subnautica shared was the ability to select a difficulty where each difficulty provides varying gameplay. One may be relaxing with no threats and allows the player to freely explore the story and world and another where the survival aspect is amplified where the player must be more cautious and strategic.

Scaling a game's difficulty can be difficult as many distinct aspects that affect the player must be considered and balanced. For example, changing the number of maximum enemies or their health can affect the player's confidence and require more power to fight. Changing the resources spawn frequency or rarity will affect the frequency of the player scavenging hence progressing slower or faster.

3.1.2 World Saving

Another key functionality that all researched games implement is saving and loading. This feature is crucial for the game's longevity as it allows the player to make meaningful progress and not be forced to repeat what they do every time they play.

When creating a story-based survival game, the game's completion playtime is generally longer than a healthy amount of time on the computer. As the player may spend a lot of time gathering resources and building structures, they may need to take breaks in between sessions so it is important for the game to be able to save progress made by the player allowing them to take frequent breaks and continue playing from how they left off when they want. Saving progress can be done by saving the state of every (or at least the most important) object in the world, the player's stats, and their inventory.

3.1.3 Item Manipulation

A critical feature that all three games include is the use and versatility of items or resources where the player may collect these resources through means and use them for their benefit. This could be crafting other items with existing ones, building structures with items, or storing these items to be used later.

3.1.4 Combat

Another key survival aspect that is present in The Forest and Subnautica is the ability to combat other entities to protect themselves or for rewards. Each game consists of multiple types of enemies with different behaviours and effects enhancing the combat aspects of the games as the player would need to learn to adapt to each

3.1.5 Advanced Movement

Since the game worlds are large and open, navigating around the world at a fixed speed can be burdensome to the player. Subnautica facilitates this issue by adding vehicles that have higher speeds than the character and includes storage space for expeditions. Specific vehicles even allow the player to transport other vehicles for long expeditions and to explore areas where it may be too risky or harmful to go without a vehicle.

The player can also swim around the world where they are able to move freely in all axes and when in the appropriate environment such as in a building or an island, they may only move horizontally while still being able to jump vertically. This effect can be replicated for movement in space where the player may move in all axes in space and only horizontally in a building.

3.2 Art and Multimedia

The visual arts of a game are the first impressions of a game and can define the type of game it is from a hyper-realistic looking game which can give a more immersive experience or a soft, colourful low poly style game which may be more child friendly and fast-paced.

3.2.1 Art Style

Going for hyper-realistic graphics like The Forest may be harder, more time-consuming, and costly, and give the game the impression of a simulator or a triple-A game. It would also make the game cater towards higher-end computers which not a lot of people own.

The project will opt for a low poly art style with enhancing visuals, similar to what Astroneer achieved. A guide by PontyPants gives some information on how to make a low poly object look more complex by taking a simple geometry, rendering it in super high resolution then applying a flat colour as a texture but using a complex shader with specularity, refractions, ETC then rendering the shapes with the latest

technology in lighting [10]. A low poly model with the use of techniques is made more appealing shown in Figure 3.1.

Figure 3.1 Light House by Jeremy Edelblut

3.2.2 Theme

The theme can help set the mood of the game, from serious to casual or dark to lighthearted, so picking an appropriate theme is important to match the mechanics of a game [11]. Objects and the environment can give the player some context to the story depending on how it looks. For example, in The Forest, the game is heavily survival-themed, so the player has a survival journal as an interface, uses handmade tools, ETC to emphasize the survival theme of the game. When deciding a theme for the game, a tip is to choose an appropriate theme that helps explain the rules of the game. If the rules involve gathering resources and building, a theme of building a city, a boat, or a spaceship will make the rules easier to understand and accept [11].

As the game will be set in space, items and models will have a futuristic look to set the theme of space exploration in the future and most structures will look besieged to give the 'crashed and lost in space' theme.

3.2.3 Sound Design

Sounds are important when relaying information to the player, especially in survival or intense situations. Josh Bycer explains the importance of background music and sound effects and how they can convey emotions, a sense of realism and authentic interactions and how all these factors create an immersive experience [18].

In space, as there is no medium for sound to travel through, there are no sounds, however, a game without any sounds could imply a more serious type of game or a horror game as it may give an unsettling realistic feeling floating in space in silence. A more casual approach while not ruining the immersion is to add sounds with a muffled effect or edit the audio clips to give an outer space effect. One way to achieve this is by adding a low-pass parametric filter to the soundtracks.

3.2.4 Interface Design

An interface communicates with the player displaying everything they need to know to make decisions such as the player's health, therefore being critical in a survival game where constant feedback is required to show the player's status and inventory.

An interface may be implemented in many ways like in The Forest with the survival manual or in Subnautica, a grid is used to represent the space in their inventory and items can fill these grids with different sizes depending on the item, for example, a water bottle can fill a 1x1 grid or a tool can fill a 2x3 grid as shown in Figure 3.2. Some games allow the same items to stack conserving space, which in a resource-gathering survival game, is very convenient to save the player time from having to travel back home to deposit all their resources.

Figure 3.2: Image of grid-based inventory interface and vitals interface in Subnautica

A survival game should also display the most important statistics of the player, which are their vitals allowing the player to react accordingly to vital levels that are in critical condition.

3.3 Game Design

Games are designed to entertain the player by giving various activities to do in the game. Open-world games generally let the player discover the activities on their own.

3.3.1 Story Telling

Telling the story of the game is important so the player understands what is going on and what to do next. This is done generally through voice acting or text from characters, animations and cutscenes used in Subnautica. However, writing dialogue and acting them out may be a lengthy process and may require more than one person to do so.

A similar concept of The Forest can be used to tell the story through in-game objects and notes that the player must explore and find.

3.3.2 Map Design

All mentioned games include an open world that the player can freely explore. Procedurally generating, levels like what Astroneer does, can save a vast amount of time as you do not need to work out where to place what, especially a repeated object such as a tree. However, in a story-driven game, some unique locations/monuments should be placed in a meaningful area of the world so that the player must travel to it to progress. When procedurally generating this, it can get messy as two big monuments can spawn next to each other. Precautions with the calculations can be taken where they must be of a certain distance.

Another approach which Subnautica and The Forest implement is using procedural generation to generate resources or debris around a premade map to add some variation.

3.4 Engineering

When implementing features in a game, it is best to find the most optimal way which consumes the least amount of time but also not compromising the quality of the game while still implementing the core functionalities.

3.4.1 Software

To reduce the time taken to create a game, a pre-built game engine will be used instead of coding a game from scratch. When comparing the two most popular game engines, the Unity game engine is known to be more versatile than the Unreal engine [12]. However, the Unreal engine exceeds graphical

performance and complexity as it includes inbuilt advanced graphical settings such as post-processing whereas Unity requires you to install separate assets to enable these. Asaf Eldad summarizes that Unreal has better graphics quality and is quicker to generate more realistic, beautiful graphics, [13] which is not necessary for a low poly style game. The Unity game engine has a bigger community which may imply more online resources when implementing complex functionalities. With my experience with Unity, C#, and Java classes, it would be much more time efficient to use Unity.

Modelling low poly objects is much simpler to create than a realistic, highly detailed object so any modelling software would be reliable to use. As I have much more experience with Blender [14] than with Cinema 4D [15], which is taught in my university course, Blender will be my primary use for creating models.

If required, an audio workstation will be used to resample or create sound effects/music for the game. I have some experience in 2 audio workstations; FL Studio [16] requires a subscription otherwise the user is unable to open saved projects but may export audio, Ableton live [17] also requires a subscription to import/export tracks but allows loading and saving. A 90-day free trial is obtainable for students therefore Ableton Live is chosen if any audio work is required. A free open-source alternative can be used if both are unavailable.

4 Requirements Analysis

The game should be delivered meeting requirements to give the player a complete and playable experience.

The game's primary objective is up to the player who may pick their own difficulty level. A much easier difficulty would be more of a sandbox game where the player can create their own adventure and goals, explore the map, fight enemies, build their own structures and explore the story of the game. A much harder difficulty would make it more challenging to survive putting emphasis on the survival aspects of the game such as crafting better tools and gear, gathering food and resources, creating safe spaces or strategic buildings, fending off enemies and ultimately exploring the story.

The game's story is kept simple; As the player explores more of the map, they will find mysterious unique items with unknown purposes. As they continue to explore, they will eventually reach a final landmark where they must use those items for something which leads to the end of the game. The player may continue playing the game if they wish to continue with the other aspects of the game or return to the menu.

4.1 Performance

Games with a large open map can become increasingly CPU intensive as the map becomes more detailed. A good requirement for the game is to be as optimized as possible to give the player a playable experience and to be suitable for even low to medium end computers to engage a larger range of players.

The game will have a lot of debris on the map which may impact performance so culling them or reducing their level of detail should be considered. The target frame rate should be around 30 to 60 frames per second as Jennifer Larson from Healthline mentions that experts will tell you that the human eye can see between 30 and 60 frames per second and that some maintain that it is not really possible to see over 60 [19].

In Subnautica, the fog helps to hide the map from being seen by the player from too far. This may be used to cull objects that are far away from the player and cannot be seen to improve performance. In low poly games like Astroneer, performance is generally not an issue due to the small number of polygons in objects, hence, the less the GPU must process each frame.

4.2 Functional Requirements

4.2.1 Mandatory

No.	Requirement	Description

1	The game should have different difficulties to select from and play	There should be a peaceful mode, a normal mode, and a harder mode where the peaceful mode means no enemies and the harder mode means the enemies and player management is tougher. This allows the player to pick a comfortable difficulty and experience and fun playthrough whether they want a challenge or to explore and have fun is up to them to decide.
2	The game should have an inventory system	Items can be picked up and stored in the player's inventory to be used at any time. This is a critical requirement for the player as it aids the task of exploring and searching for items and so the player may carry these items with them to take wherever and use whenever. The inventory must have a limit to provide a challenge to the player to manage their inventory and bring the necessary items with it for the appropriate expedition.
3	The player should be able to die and lose their items or respawn with them	When the player runs out of health, the player will die and may choose to respawn or load from a save which, depending on the game difficulty, will give the player items back or drop them where they died. The ability to die will give the player a reason to look out and manage the game character to avoid the risk of losing their possessions. This only applies to some game modes to benefit players that want a more relaxed experience.
4	The player should have vitals that need to be managed	Vitals such as hunger, thirst, health, and oxygen will all have to be managed as some slowly decreases. This function gives the extra task for the player to manage themselves by making sure their body is nourished and to acquire sustenance by any means and find new ways of doing so as they may be scarce.
5	The game should have a save and load feature	The game should save the world's current state and be able to be loaded again and play from where they left off including the story. As the game may have a long full play time, players may not have enough time to complete it in one sitting without playing for an unhealthy amount of time and therefore possibly making them lose interest in repeated gameplay. The save and load feature will prevent that by allowing the player to continue where they left off.
6	The player should be able to craft items	With resources, the player can create items and tools through a crafting interface. This makes gathering resources more useful as the items have a purpose. The player can craft other tools to speed other processes up, or craft sustenance or even other crafting components.
7	The player should be able to build structures in appropriate areas	The player should be able to place, move and destroy structures in appropriate areas. The ability to create a home base allows the player to store their possessions and to have a fixed safe location that the player can relax in. Adding furniture may also give the player the ability to customize their home base for fun.
8	The player should be able to harvest resources	Using different tools, the player should be able to harvest the corresponding resource. This makes obtaining items more of a challenge rather than just being able to pick them up. This gives different tools their importance and uses.
9	The player should be able to attack enemies with several	Weapons will vary in types such as ranged, projectile, blade or even tools and will do damage. This gives the means to fight off enemies and gives variety in combat where the player may use several types in certain scenarios.

	types of weapons	
10	The game should have at least one type of enemy that can harm the player	The enemy will be able to chase the player and attack the player within range. The enemy may drop items on their death as a reward. This will give the player a challenge when exploring the world as they will have to deal with sudden threats.
11	There should be at least one type of hazard that can harm the player	Possibly a radiation zone from the debris where if the player moves too close, the player starts to lose health/radiation levels increase. This may be reduced or dismissed with an upgrade. Hazards can be used to control the player to allow or prevent them from being in places they should be yet or at all such as the border of the map or a storyline zone that is too early for the player to access.
12	The game should have puzzles that the player can complete for rewards or story line progression	Landmarks in the map may include puzzles such as mazes, logic puzzles or others that may be completed by the player for rewards or clues for the story.
13	The game should have a simple story that the player can follow and complete	There will be clues or objective items the player collects which leads to other locations with puzzles and finally reaching an end point. A story line with objectives the player can complete will give other functionalities a greater purpose such as an objective can be to acquire a certain number of specific items (items and crafting) or to eliminate enemies at a location (weapons and enemies).

4.2.2 Optional

No.	Requirement	Description
14	The player should be able to experience various levels of gravity	Depending on the location or environment, the player may alternate between experiencing gravity and no gravity. This could be inside a ship with artificial gravity or a 'magic' field. This can immerse the player into the space theme.
15	The game should have advanced weapons that perform different effects	Ideas for advanced weapons would be several types of projectile weapons like an explosive projectile, impulse, repulsion, gravitation, ETC. Maybe a teleporter that lets you teleport back to a saved location, a portal gun. Advanced weapons can be used for fun where the player may just experiment with them or may greatly help them.
16	The game should have more than one type of enemy	May include robots that can use ranged weapons or an enemy with different sets of attacks. Fast enemy but weak, and slow enemy but heavy hitting. Support enemies that heal other enemies or slow players. Adding various types of enemies can give the player the feeling of not knowing what to expect and requiring the player to figure out different means of defeating them.

17	The game should have different means of navigation and transport	This could include a faster jetpack, a small mobile vehicle with storage space and higher speed (fueled). Vehicles or upgrades can help the player get over the eventually tedious process of flying slowly to destinations.
18	The player can upgrade their stats and tools	Craft upgrades for their suit which may help with radiation, slower hunger or thirst decrease, slower oxygen consumption or higher capacity, more damage, maybe a solar energy upgrade to recharge battery powered tools. Better batteries for tools, refine tools with materials.
19	The game will have tamable or creatable pets that follow the player	The player may craft a robot dog/cat/animal that follows the player and acts as mobile storage for the player, maybe adding a buff for the player such as higher speed or damage, ridable with jet engines.
20	The key binds should be able to be remapped	The player can change each functional key to a key of their choice which may benefit experienced gamers that have their preferences.

4.3 Non-functional Requirements

4.3.1 Mandatory

No.	Requirement	Description
21	The game should run at least above 30 frames per second	30 frames per second will be the minimum the game will run and must not drop below 30 to fit the comfortable natural frames per second the human eyes are used to.
22	The game will run on Windows 10 operating systems	The game will be run on the most popular desktop operating system for it to be accessible to as many players as possible.

4.3.2 Optional

No.	Requirement	Description
23	The game should reach 60 frames per second	The game should not drop below or at least average 60 frames per second or higher.
24	The game's graphic settings can be adjusted	The player will be able to modify the level of graphics and each detail to their choice such as bloom or shadows. This allows the player to turn off visual features they do not like or optimize the game for their computer specs.

5 System Design and Architecture

As this game will include many features, development can become difficult and messy without any structured approach and techniques. Software design patterns and Unity features are utilized to help structure development.

5.1 Component-Based Architecture

Unity uses a component-based architecture (CBA) where game features are broken down into subcomponents in which each has their own functionalities and can be combined and reused for other unique features. This allows for code reusability and reduces redundant code allowing more time for other development work.

A common instance, which was also used in this project shown in Figure 5.1, is breaking down the character into its main components; A player controller that manages the player's inputs, a player motor that handles moving the entity and commonly for 3D games, a camera controller that handles rotating the player and their view. To then create an enemy, the player controller component can be replaced with an enemy AI (Artificial Intelligence) component to handle inputs while still adopting the same movement logic and physics.

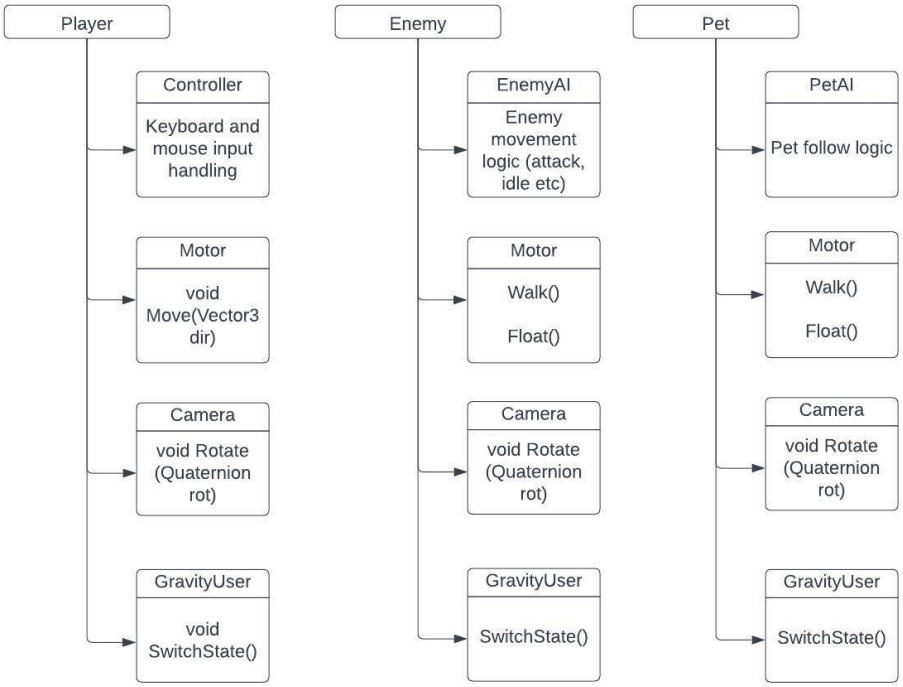

Figure 5.1 Example of components being reused

This approach comes with its drawbacks as components must be sufficiently general to cover the different aspects of their use but at the same time, must be concrete and simple enough to serve a particular requirement in an efficient way [20]. When it comes to complex and large features, developing reusable components may require three to four times more resources than developing a component that serves a specific case [20] so some compromise of reusability for time may be helpful.

5.2 Singleton Pattern

A singleton pattern ensures that only one instance of a class may exist at any given time, providing easy access to that instance. A singleton is accomplished by having private constructors so no other classes may instantiate any instances of it and a static getter or method that returns a reference to that instance.

In Unity, a singleton is created by having an instance of the class on an empty game object in the scene. When the scene is played, the Awake method, which is the first method called by every Monobehaviour when they are loaded into the scene, is called where the class ensures that there is only one instance in the scene.

Using singletons in Unity can be much more beneficial than a static class as properties in that class can be exposed in the scene inspector where developers can assign references or modify variables with ease allowing efficient prototyping and testing as shown in Figure 5.2. This makes singletons ideal for game manager classes or utility classes such as an audio manager class. In Figure 5.2, values such as volume or pitch can be adjusted to the desired sound while actively testing the game.

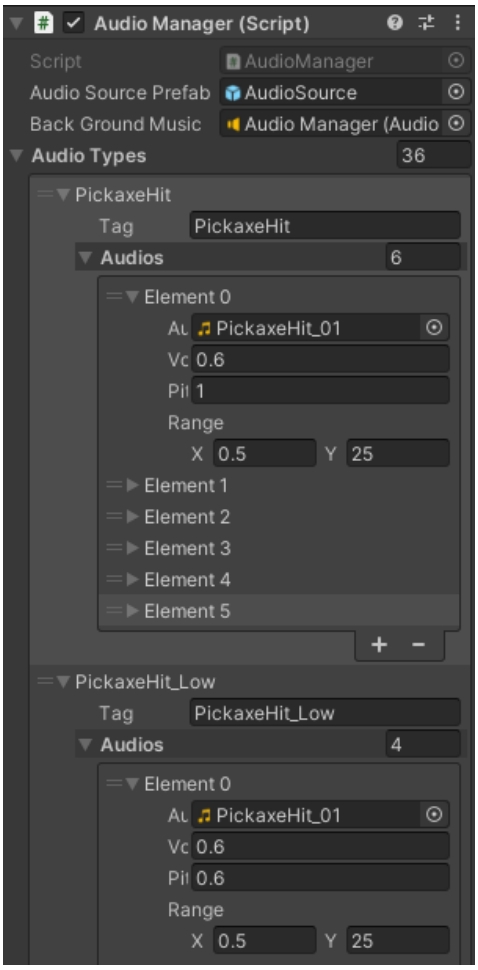

Figure 5.2 Audio Manager singleton with assigned audio clips and altered values

5.3 Delegates and Event Systems

Delegates are reference types that store references to methods with a particular parameter list and return type [21] where when invoked, all subscribed methods to that delegate instance will be passed the set parameters if any and called. If the delegate returns a value, the last subscribed method's return value will be returned from the multicast delegate.

An Event is a bit different to a delegate where Events are typically public class members. By comparison, delegates are often passed as parameters and stored as private class members, if they are stored at all [22]. They can be used to create an event system where listeners can be subscribed to a delegate event instance that may be invoked subsequently calling all methods subscribed to it.

The Unity input system is event driven so all inputs created are delivered as events allowing methods to be subscribed and called upon the player's inputs. This can be used to create player controls for movement and UI manipulation.

In this project, the player controller class handles the input system events and calls its own defined events depending on the state of the player, for example, whether the player can move or rotate or attack while in their inventory. In figure 5.3, the OnScroll event is only called if the player is allowed to which is set by manager classes.

```
public delegate void HotbarActions(int _num);
public static event HotbarActions OnScroll, OnSwitchTo;
/// <summary>
/// Toggle the player's ability to switch their hot bar
/// </summary>
/// <param name="_state"></param>
16 references
public void ToggleSwitchHotbar(bool _state)
{
    canSwitch = _state;
}
void SwitchScroll()
{
    if (!canSwitch)
        return;
    //Returns 1D vector depending on scroll dir
    float _scrollAxis = playerInputs.Player.SwitchScroll.ReadValue<float>();
    if (_scrollAxis != 0f)
        OnScroll?.Invoke((int)_scrollAxis);
}
```

Figure 5.3 OnScroll event in the PlayerController class

Upon scrolling up or down, the integer value of that direction is passed as a parameter in the HotbarActions delegate instance 'OnScroll.'

5.4 Dynamic Object Pooler

As the game contains a lot of items and manipulations with them revolving around creating, destroying, enabling, and disabling objects this can become very intensive for the CPU where objects are constantly needed to be created and generate a lot of garbage from destroying objects which leads to frequent garbage collection.

Garbage collection is Unity's automatic memory management that reduces the risk of memory leaks and other program problems [23]. However, this process requires a significant amount of CPU time, so it is best to reduce garbage collection as much as possible. One big contributor, especially in a game like this, will be item management.

An Object Pooler helps to prevent destroying objects when they are not needed anymore and rather 'pooling' them by disabling them and storing them in a data structure ready to be reused. A generic object pooler will instantiate a set amount of a specific object ready to be used whenever needed. This removes the requirement to destroy objects but as this game contains many different types of objects that need to be pooled, this could occupy unnecessary memory when having ten instances per item.

A dynamic object pooler is created where objects are not initially instantiated but will pool objects with a given tag to distinguish them from each other. When an object is required and there are no existing instances of that object, a new one is instantiated when needed which adds to the maximum number of objects of that type allowing minimal memory utilization. A string is used as a key for a dictionary of queues of game objects which are enqueued and dequeued upon spawning and pooling, respectively.

5.5 ScriptableObjects and Item System

A ScriptableObject is a data container that can store information which is independent of class instances. Using ScriptableObjects along with prefabs can help reduce memory usage by avoiding copies of values; You can use a ScriptableObject to store the data and then access it by reference from all the Prefabs. This means that there is one copy of the data in memory [24]. ScriptableObjects also supports inheritance allowing a comprehensive and optimized system. The use of an Object Pool along with ScriptableObjects for item instances greatly optimizes memory usage. The 'CreateAssetMenu' can be added to allow the player to create instances of such ScriptableObject in the editor as shown in Figures 5.4 and 5.5.

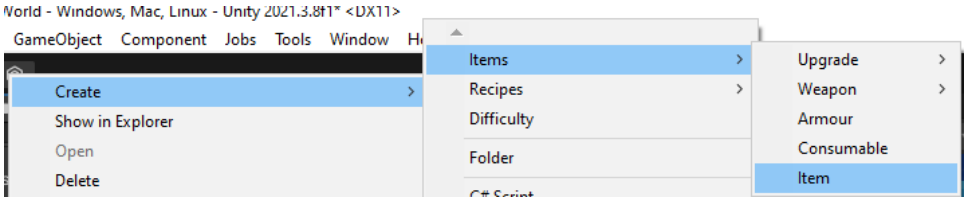

Figure 5.4 Create asset menu with option to create an ItemScriptable instance

In this game, most objects will revolve around an item system where a base 'ItemScriptable' class stores the basic information of an object in the game to be view being its name, icon, and description for it. 'WeaponScriptable' derives from this and stores basic information for any weapon such as damage, maximum range, and attack range. These ScriptableObjects are then assigned to prefabs of the following items or weapons and data can then be accessed from the ScriptableObjects by reference.

Methods may also be declared and called allowing data manipulation within the object itself. The base 'ItemScriptable' class provides virtual methods 'Use,' 'Equip' and 'Unequip' which are called when the player presses the use key on the slot with this item, equips the item to its specific item type slot and unequips it, respectively. Child classes may override these methods for specific effects such as a Consumable class may add health to the player upon use.

```
[CreateAssetMenu(fileName = "New Item", menuName = "Items/Item")]
12 references
public class ItemScriptable : ScriptableObject
{
    [Tooltip("The type of item this item is")]
    public ItemType type;
    [Tooltip("The name of this item")]
    new public string name;
    [Tooltip("The icon for this item")]
    public Texture icon;
    [Tooltip("The description for this item")]
    [TextArea]
    public string description;
    [Tooltip("This item will not be stored with other in the object pool")]
    public bool unique;

    /// <summary> Action when used inside the inventory
    3 references
    public virtual void Use()...

    /// <summary> Action when moved to its special slot
    9 references
    public virtual void Equip()...

    /// <summary> Action when removed from its special slot
    7 references
    public virtual void Unequip()...
}
```

Figure 5.5 ItemScriptable ScriptableObject

5.6 Scene Switching

The game includes multiple scenes; A main menu scene where players may start a new game or load an existing game, and a main world scene where the game is played. In Unity, upon switching scenes, all game objects are destroyed by default and when reopening the scene, new instances are created. This can be prevented by calling the DontDestroyOnLoad method from any classes deriving from Monobehaviour, the default Unity class and passing the game object to keep as a parameter.

5.6.1 Static Variables

Despite objects and instances being destroyed, as static variables do not need an instance to store a value, their values are kept between scene switches including static delegates and events. Monobehaviour provides a method ‘OnDestroy’ that is called before the object is destroyed. This can be used to reset static variables and unsubscribe methods from static events and delegates to avoid errors and null reference exceptions due to events storing references to methods from the destroyed previous instances.

5.7 Custom Rendering Passes

Unity's High-Definition Render Pipeline (HDRP) allows for custom shaders to be injected within the render loop allowing modified rendering for objects certain objects. Objects to render can be filtered with Unity's layer system where objects with a specified layer are rendered with the custom pass. The injection point for the pass may be picked from six different points with all different buffers for different types of materials and material properties [25] so depending on the custom shader, an appropriate injection point needs to be picked.

5.7.1 Always Render on Top

In this project, a custom render pass is created to render objects over everything by overriding the depth test to always as shown in figure 5.6. This tells the shader to ignore depth testing resulting in all the object's geometry to be rendered despite being behind another. Different depth test parameters lead to different effects when drawing geometry [26]. This pass will be executed within all buffers so any objects with the 'RenderOverAll' layer will be rendered on top of everything.

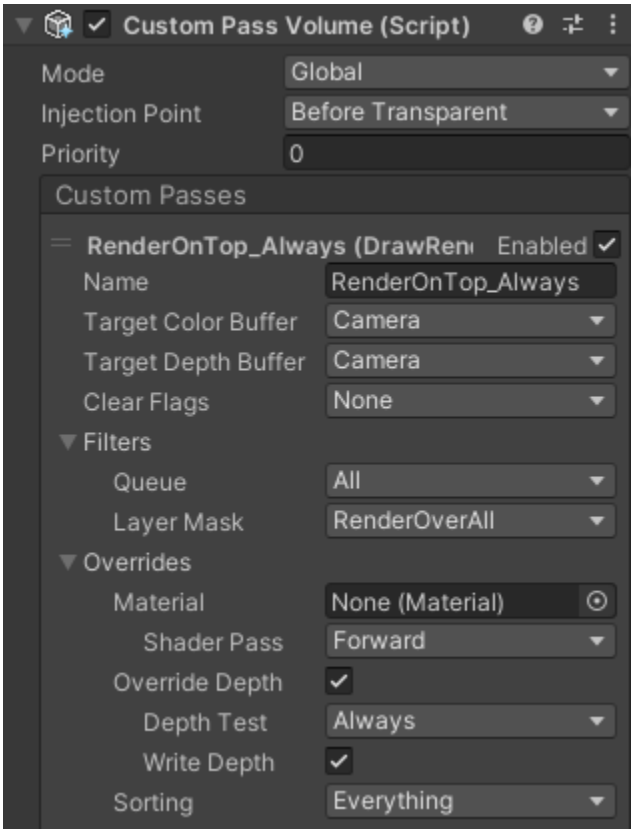

Figure 5.6 Custom 'RenderOnTop_Always' render pass

This pass is used to create a world space UI for the player to prevent the UI from clipping into surfaces that are too close to the player therefore it must be rendered over everything.

6 Implementation

The game contains many unique features; Below are all the key features and their implementation methods.

6.1 Item Types

When collecting distinct types of items in the inventory, they must be distinguished for their own usage. A base 'Item' class with the IPickable interface allows the player to pick up any object with this component attached. The Item class also stores a reference to an ItemScriptable instance where that will store information for that item. Child classes such as ConsumableScriptable can then have their own effects when used in the inventory. Figure 6.1 shows a UML diagram of different types of items being used in the same Item component.

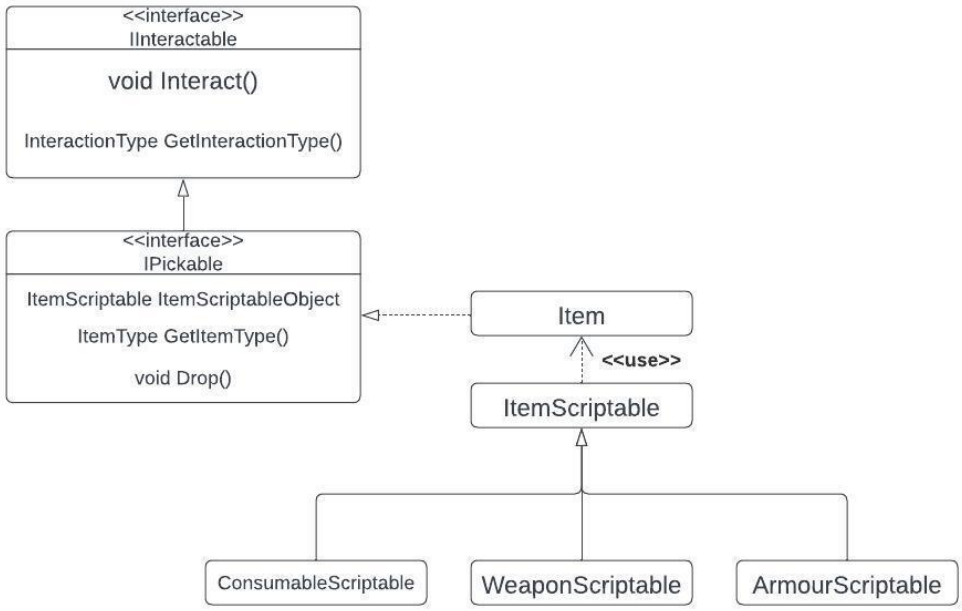

Figure 6.1 UML diagram showing item system.

6.2 Inventory System

Every inventory, including the player's and other storages, includes 'InventorySlot' instances where this class has a sole purpose to store information of an item. The player can drag a slot from one to another effectively moving items from storage to storage or back into the world. Figure 6.2 shows a UML diagram of storage using inventory slots to hold items.

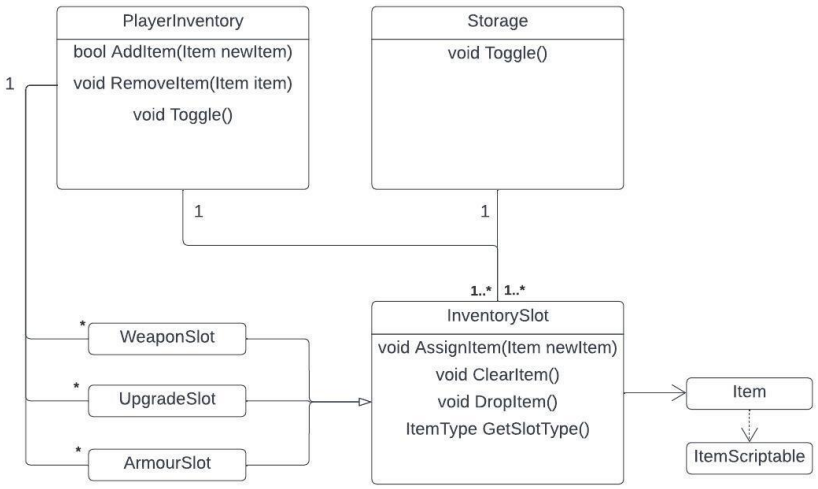

Figure 6.2 UML diagram of inventory system and storing items in slots

Using the Event Trigger component with the slot, different actions can be declared upon different input events. In Figure 6.3, when the player hovers over the slot, the item's information is displayed in the inventory item display to show its name and description about it by accessing the ScriptableObject instance stored within the Item that the slot is holding.

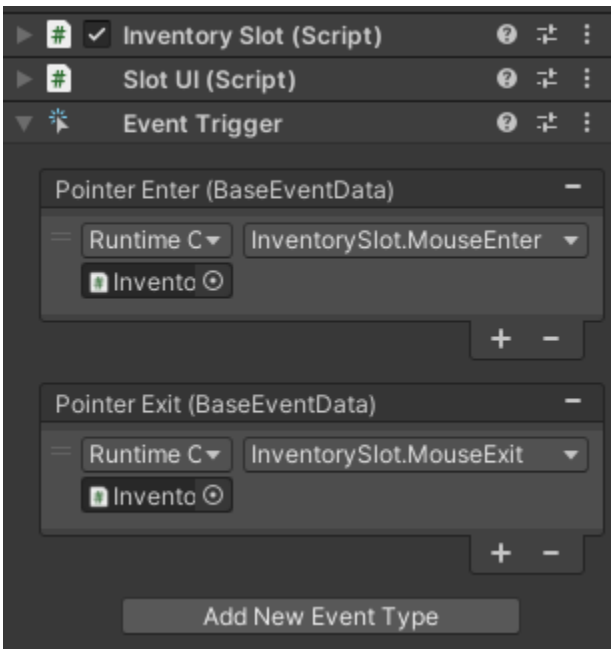

Figure 6.3 Inventory slot prefab to place wherever with Event Trigger functions

This base slot can be inherited and be made to only store a specific item type such as upgrades. This ensures that only items that are of type upgrade may only be stored in those slots. Upon equipping and removing the upgrade item, special effects are applied to and removed from the player, respectively.

6.2.1 External Storages

With the way storing items operates, it is possible to create external storage by adding this Inventory slot component with its required UI component to store any or specifically set item types by dragging and dropping other inventory slots on them. A prefab can be created to store this slot preset to be used where storing items is desired.

6.3 Crafting System

Items that are collected may be used to convert into other items. The crafting system stores recipes of these conversions and displays them in a set of other recipes. Different sets of recipes can be defined allowing different types of crafting such as a cooking station where you may only craft food or a smeltery where you may only refine resources.

Recipes are created using ScriptableObjects to store what is required and what is the product where a crafting manager checks the player's inventory for items that are required and determines if they can craft. These recipes can be easily changed by modifying the ingredients list of items and assigning these recipe ScriptableObjects to the crafting manager to the set of choices as shown in Figure 6.4. Every crafting table instance of a type will display the same set of recipes as they all store a reference of the same recipe ScriptableObjects.

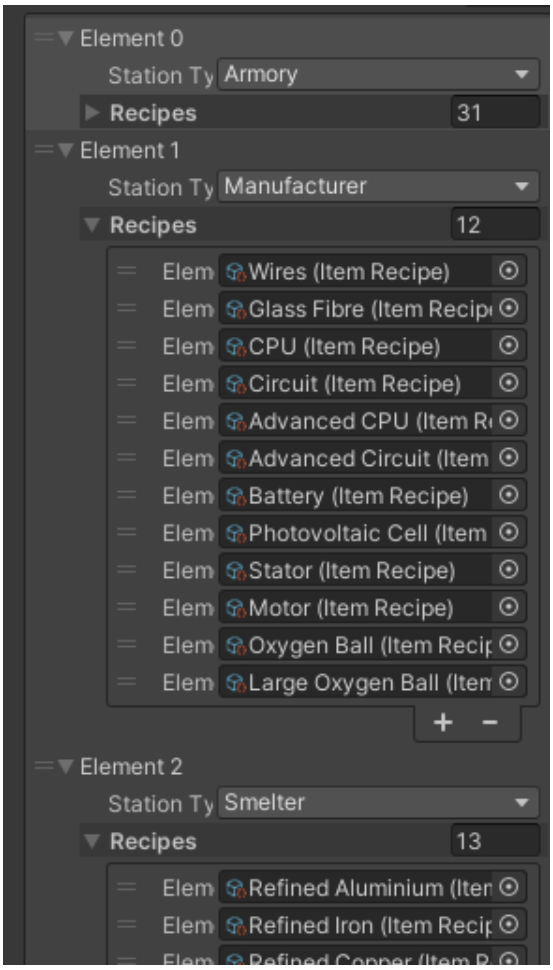

Figure 6.4 Crafting Manager with sets of recipes for a crafting table type

The product item along with its ingredients is displayed beside them to show what is required to craft the item as shown in Figure 6.5.

Figure 6.5 Crafting table displaying a pickaxe with its ingredients

6.4 Building System with Grids

The player can also use the items they collect to build structures using a similar recipe system the crafting system does. Instead of crafting the item, the player can move a blueprint around and upon confirming their placement, items are consumed. Upon hovering over a slot, the building and its ingredients are displayed as shown in figure 6.6 to show the player what is required.

The building interface is accessed through a specific tool that must be obtained. The tool allows the player to build and remove their built objects which refunds their items or drops them into the world if their inventory is full.

Figure 6.6 Building interface showing buildable with its ingredients

6.4.1 Experimenting Building Systems

At first, when each hull structure, I.e., walls and floors, were built, their positions were stored in a list of Vector3 positions where upon placing a structure, the new structure's position is checked if that already exists in the list. This is a straightforward way of detecting overlapping in a grid system. However, this provided no versatility to perform calculations such as seal checks or checking how large the overall structure was.

A new building system is designed and implemented using a 3-dimensional grid where buildings can snap to the sizes of these grids or any other set values. Floor and walls will snap to the grid sizes and other decorative buildings can snap to a desired grid size for precise placements.

6.4.2 Defining Structure Systems

Upon building floors and walls, a structure system is created to store information about the structure of each necessary grid and the origin point of that system. It uses a 3-dimensional array to store the positions of each connecting floor and wall. It can be expanded or shrunken depending on where they are placed;

If they are placed alone, a new system is created of a 1x1x1 size. If placed on a grid next to another system, that existing system is expanded along with the new structure. If removed, resulting in a smaller overall grid size, the structure system size is shrunk. When expanding and shrinking the 3D array, the indices of the existing structures must be considered. Figure 6.7 depicts an instance when a floor is added next to a system, when adding this to the array, its index will be out of bounds so the array must be resized, and existing content along with the origin point must be offset accordingly then finally adding that floor to the grid array. The same applies to removing a floor that leaves an empty row in an axis, the grid is shrunk and offset.

Figure 6.7 Sketch of adding a floor at –1,0,0

6.4.3 Connecting Systems

Multiple systems can also be connected depending on where floors and walls are placed and positioned in the world and split apart depending on how a floor or wall is removed within a system. When placing a floor between two separate systems as shown in Figure 6.8 the systems are connected into one; the maximum distance in the x, y and axis are calculated from both systems which are used to determine the size of the new system. The two old systems are then merged into the new system along with the newly placed floor.

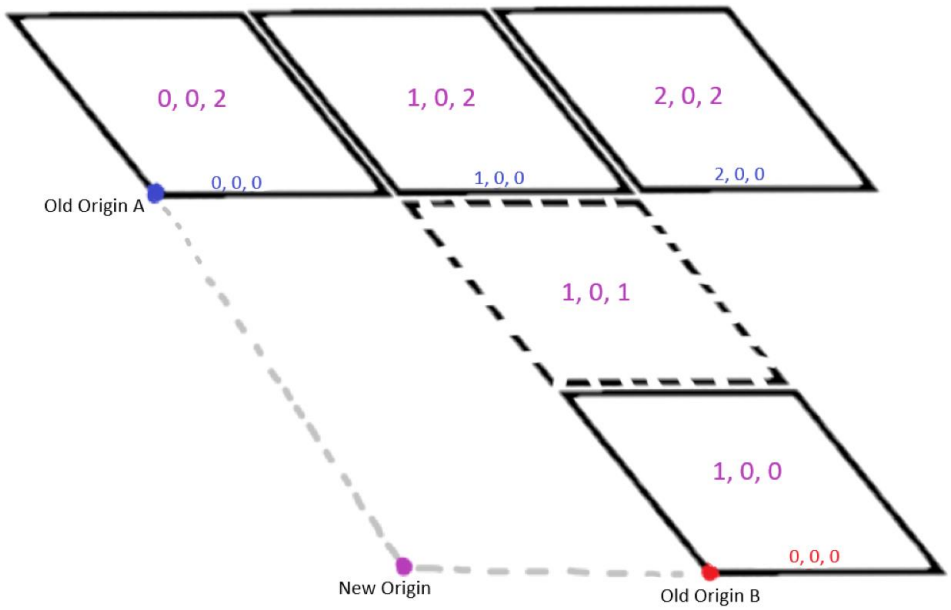

Figure 6.8 Sketch of connecting two system and applying offsets to content and origin

6.4.4 Splitting Systems

When splitting a system, a recursive traverse search is used to check for connected grids in the system and mark them as traverse. Upon traverse completion, all grids marked are then stored in a new system and this is repeated until all grids are traversed within the system to be split. For example, in Figure 6.9 this system is split in the middle resulting in four new systems with appropriate origin points. However, in Figure 6.10 the system is not split as all units are still connected.

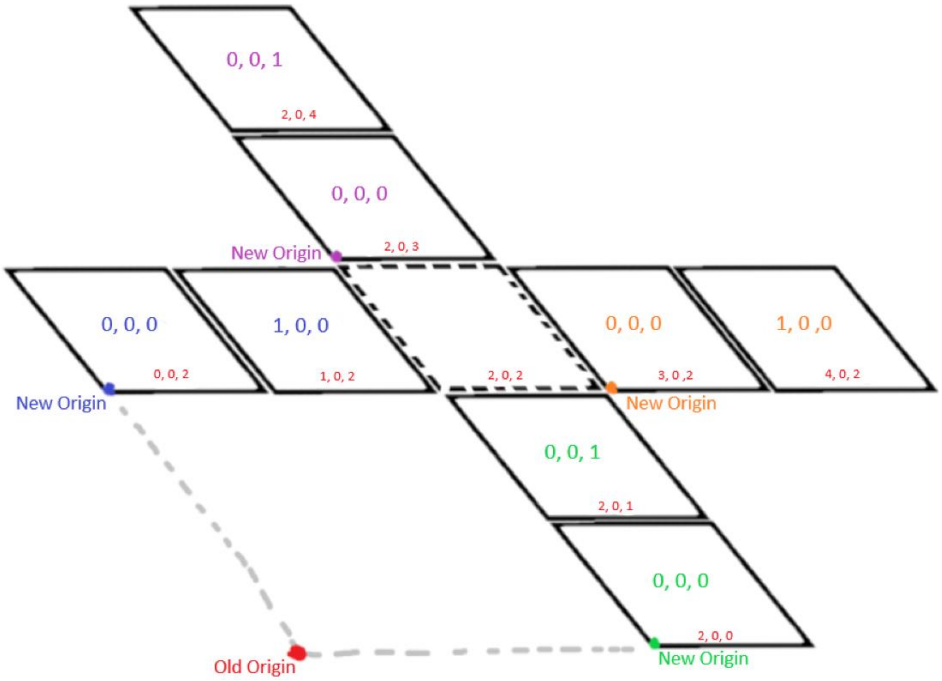

Figure 6.9 Sketch of a system being split into four

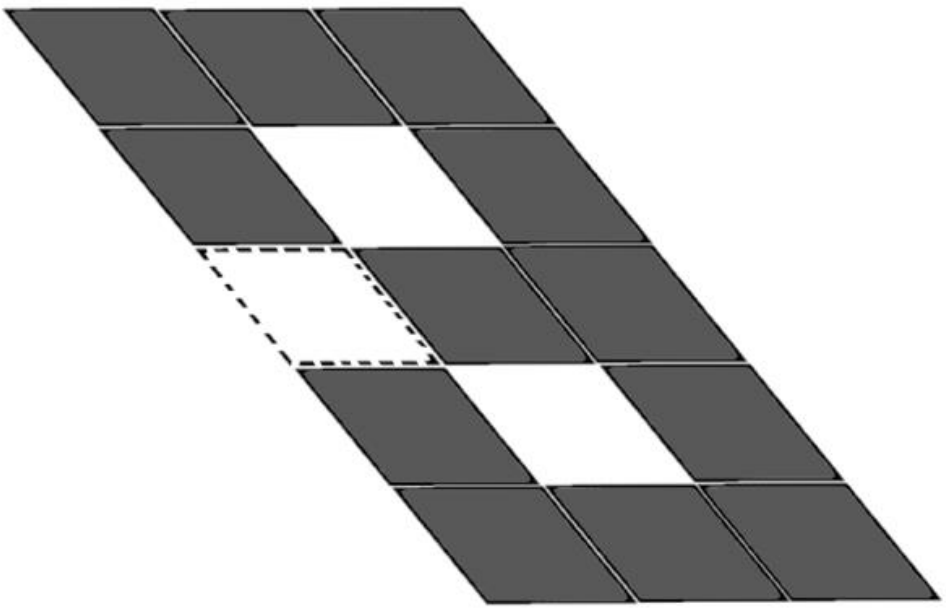

Figure 6.10 Sketch of system unchanged from a floor removal

6.4.5 Sealing Systems

This connected structure system can be used to detect overlapping when placing floors and walls instead of a physics calculation. As the system also stored the floors and walls of a grid, this can be used to check if a system infrastructure is sealed or not allowing for a complex yet modular building system. Sealed buildings can provide electricity, gravity or in this case due to time constraints, oxygen. The system uses a similar recursive breadth first traverse to check if the building is sealed or contains any breaches. Upon a successful traverse, each structure grid is marked as sealed and those that fail the traverse or empty grids within the system are marked as unsealed. The algorithm traverses through the system every time it is updated and attempts to find a breach. It checks all six sides of the cube unit and traverses each one where an empty space is found, it then attempts to traverse its way outside of the system to check if the system is open to space. This is because if the algorithm just returns when an empty space is found, this could ignore empty space inside the building itself, for example, a 3x3x3 would possibly have an empty space in the middle.

Building is also prevented around landmarks to prevent the player from exploiting the map by adding trigger colliders with the tag ‘No Build’ in areas to prevent building.

6.5 Gravity and Oxygen Detection

A dynamic oxygen and gravity system is implemented using Unity’s game object tags and trigger colliders. Each trigger collider will be marked as either ‘Oxygen,’ ‘Gravity’ or ‘OxyGrav’ for both effects to remove redundant trigger colliders. Components ‘OxygenUser’ and ‘GravityUser’ will contain the OnTriggerEnter and OnTriggerExit methods from Monobehaviour which are called when the objects enter a trigger collider. A tag comparison is used to determine if the object entered its corresponding zone, and its effect is carried out.

6.5.1 Oxygen with Buildings

When applying this to building systems, it can become expensive to have a trigger collider for each unit grid in a system as the building grows. It may also be very cumbersome to calculate meaningful resizing algorithms for the triggers. With the addition of many physical objects around the map, this can become very performance heavy leading to stutters and frame drops. Instead of using a physics approach for buildings, the grid system is used to check if the object is currently in a system or not following that a check if it is in a sealed unit.

6.6 Enemies and Finite State Machines

Unity's animation system 'Mecanim' [27] provides a finite or animation state machine which can be used to play animations for enemies based on their state as shown in Figure 6.11.

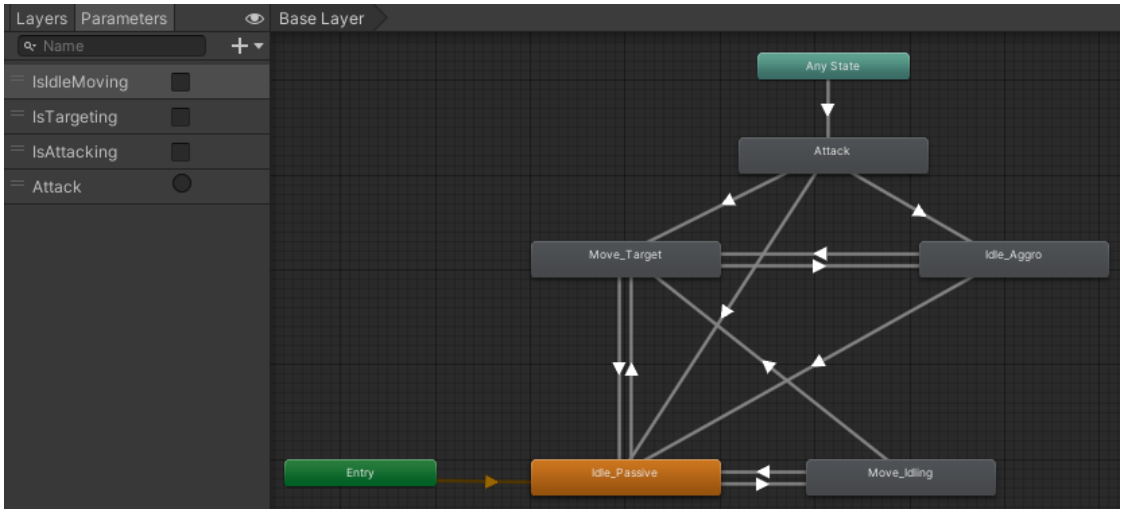

Figure 6.11 Animation State Machine for an enemy

6.6.1 States

The enemy has five states. A passive idle state where the enemy is stationary and not aggressive to the player, a passive moving state where the enemy is moving and rotating randomly to mimic it roaming, a moving aggressive state where the enemy charges the player, a stationary aggressive state where the enemy is targeting the player and is close enough to attack, and lastly an attacking state when the enemy attempts to attack the player. The animation states are transitioned using parameter conditions which can be altered in code playing the correct animation during a state. In each loop of the update method for Monobehaviour, the distance from the player is checked and the enemy's state is determined with set distance variables shown in Figure 6.12.

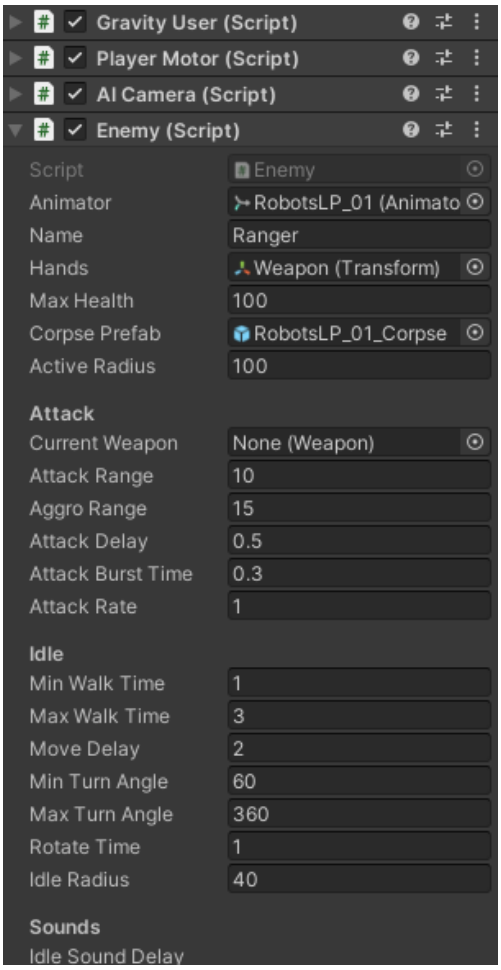

Figure 6.12 Enemy prefab with components building up the enemy

6.6.2 Delaying and Interrupting States

To delay movements and attacks, coroutines are used allowing the execution of methods to be suspended and resumed. This can be used to pause the enemy at a state for some given time, for example, the attack burst time property is used to force the enemy to continuously attack for that given time.

When the state needs to be interrupted for a much more important state for example, from idling, which is also controlled using a coroutine, to aggressive, a reference is required to the running coroutine to be passed as a parameter in the StopCoroutine method to properly interrupt it.

When the enemy dies, they are disabled and pooled and their corpse counterpart is spawned at their location playing a short animation. The corpse is then harvestable by the player granting bonus items.

6.7 Weapon System

The game contains three fundamental weapon types with all unique behaviours. The main distinctions between the three are that melee weapons do not use ammo but rather durability and have a shorter range. Ray weapons have a longer range with their attack effects being instant to a single target with the use of ammo and finally projectile weapons which also use ammo but shoot out projectiles that do damage instead causing delayed damage but generally within a radius. These weapons can be easily created and customized by creating the corresponding ScriptableObject for each and modifying values. The weapon class makes use of the Item component to access information about the weapon as well as being able to be picked up to be stored in inventory slots where Item instances may only be stored. Figure 6.13 shows the usage of the item class with the weapon class along with the interactions of different weapon types.

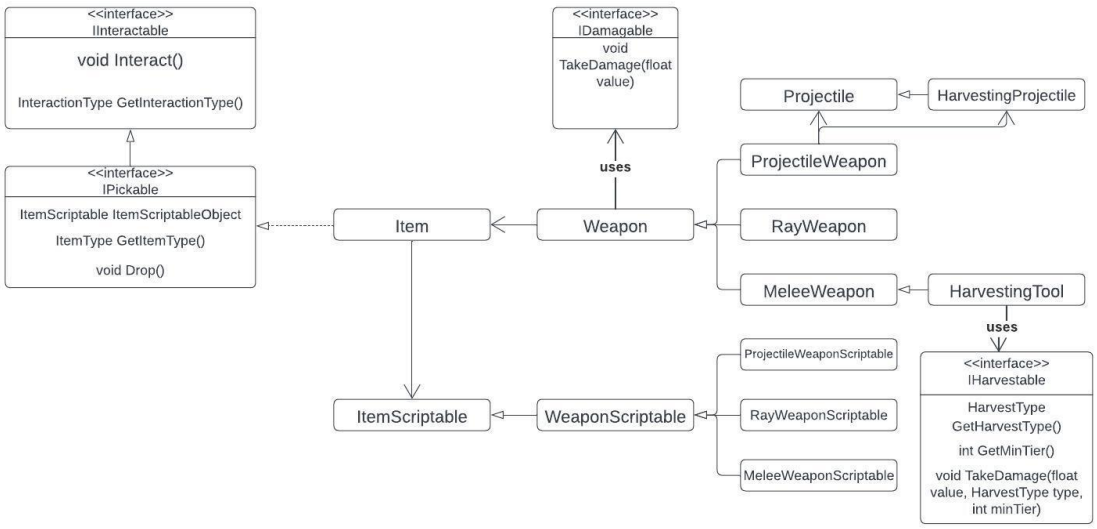

Figure 6.13 UML diagrams of weapon system

6.7.1 Harvesting

These weapon classes can be inherited to create a special weapon to destroy other objects. The HarvestingTool and HarvestingProjectile class behaves like a melee weapon and projectile respectively with the addition of the ability to harvest specific objects. An enumeration can be used to define different types of harvesting tools such as mining, cutting, ETC along with an integer to define a mining tier to create filtered harvesting systems. Upon harvesting an object, other objects can be set with a chance to spawn as shown in Figure 6.14.

Figure 6.14 Harvestable resource 'Gold ore' dropping gold ingot upon harvest

6.8 Voxels and Marching Cubes

Voxels and marching cubes are added to the game to randomly generate a collection of cubes or a low poly mesh, respectively. In addition to the harvesting system, they can be made to drop resources when the player attacks the mesh. Every instance generated can be customized allowing for dense or sparse, large, or small, round, or sporadic meshes and a harvesting type with a minimum harvesting tier can be declared. These nodes can be placed around the map to be generated for the player to collect resources.

6.8.1 Voxels Overview

Voxels can be understood as '3D pixels' where each voxel is a pixel on a 3-dimensional grid [28]. A more genuine definition can be that they are coordinates in a 3-dimension grid and a mesh is drawn based on those coordinates creating complex objects. Despite its naturally simple shape, when used at a larger scale, it can create detailed and dynamic models that can be easily edited in runtime. Atomontage Inc. showcases a voxel engine using volumetric technology which is capable of direct editing of models, editing through a 'painting' effect and even a volumetric video stream [29]. Voxels are also used at a professional level where their uses vary from medical or scientific visualization, for example, voxel-based morphometry is used to assess structural changes in the human brain [30].

6.8.2 Generation

In this project, voxels are used to create simple harvesting nodes for the player to harvest. This is done by populating a 3-dimensional voxel map with Booleans where values that are true will be used to draw a voxel. Populating is done by using a custom noise function to generate a 3-dimensional Perlin noise [31]. Perlin noise is a type of gradient noise which can be thought of as a smooth random noise as shown in Figure 6.15. When a noise value is generated, this is compared to a set threshold which determines

whether the voxel is active or not creating spaces within the voxel map. The map can also be rounded by checking if the index of the map is within a certain radius to give an asteroid look.

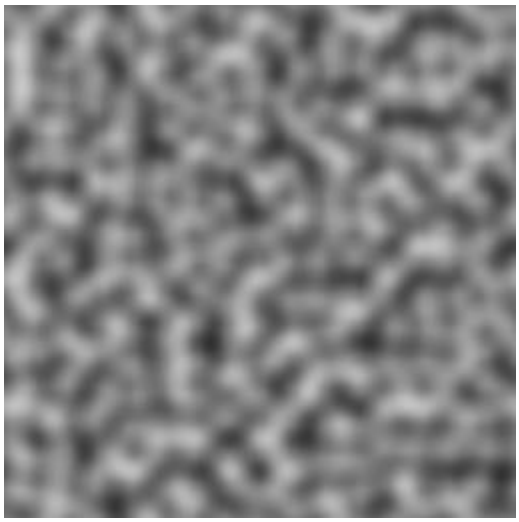

Figure 6.15 2D Perlin noise map generated with Blackears' Perlin Noise Maker (2017)

Further variables such as octaves, which control the number of extra noise layers to add upon the mesh, lacunarity which controls the increase in frequency per octave and persistence which controls the decrease in amplitude per octave [32]. This can be adjusted to create lumps and divots in the asteroids or even holes.

6.8.3 Drawing Voxels

After the array is populated, faces are drawn based on other surrounding voxels. A cube face is drawn by drawing two triangles using the local voxel coordinates. The vertices of the triangle must be selected in a clockwise order as the calculated surface normal will face forward or the 'outer side.' For example, in Figure 6.16, if a triangle with vertices 0, 1 and 3 is drawn in that order, the face drawn will face inwards and will not be visible from the current view whereas the vertices 0, 3 and 1 will produce the correct normal. The face drawn is also dependent if there is a neighbouring voxel; Faces will not be drawn to prevent unnecessary vertices and mesh which is the main benefit of using voxels.

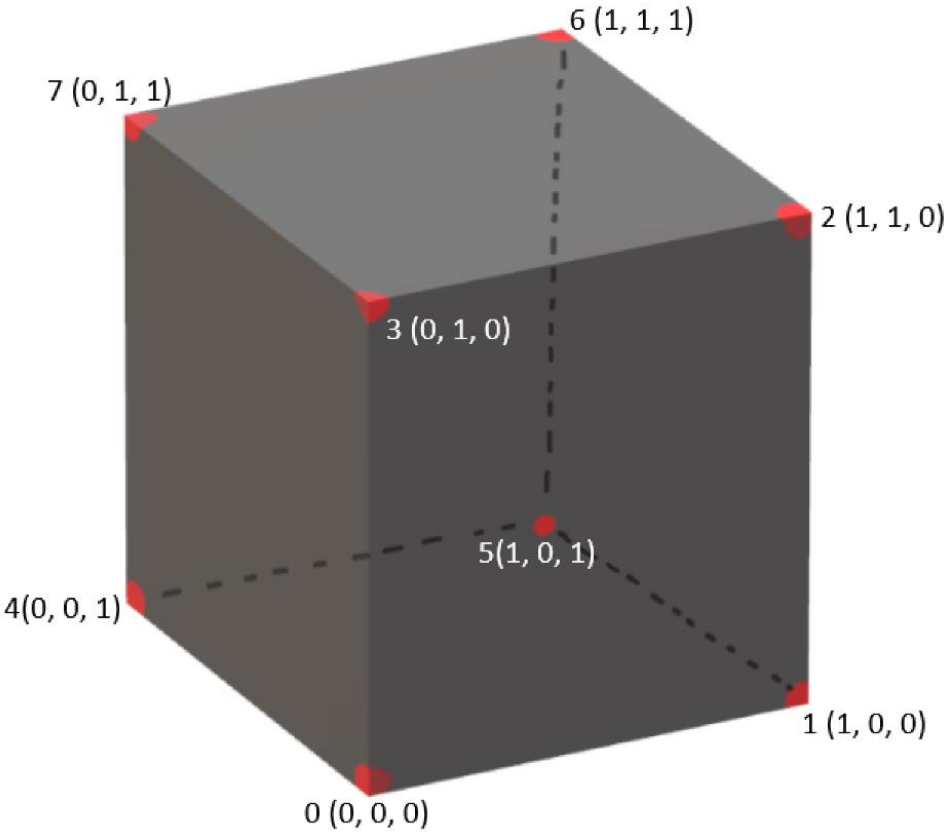

Figure 6.16 Voxel vertices coordinates

Unity allows a mesh's index buffer to be either 16-bit allowing up to 65536 vertices or 32-bit allowing up to roughly 4 billion vertices [33]. To keep the game performant, meshes will be limited to 65536 vertices as there will be a large number of generated asteroids in the map at once. When developing, the developer is notified if a generated mesh exceeds this number of vertices along with Unity not generating the full mesh itself as shown in Figure 6.17.

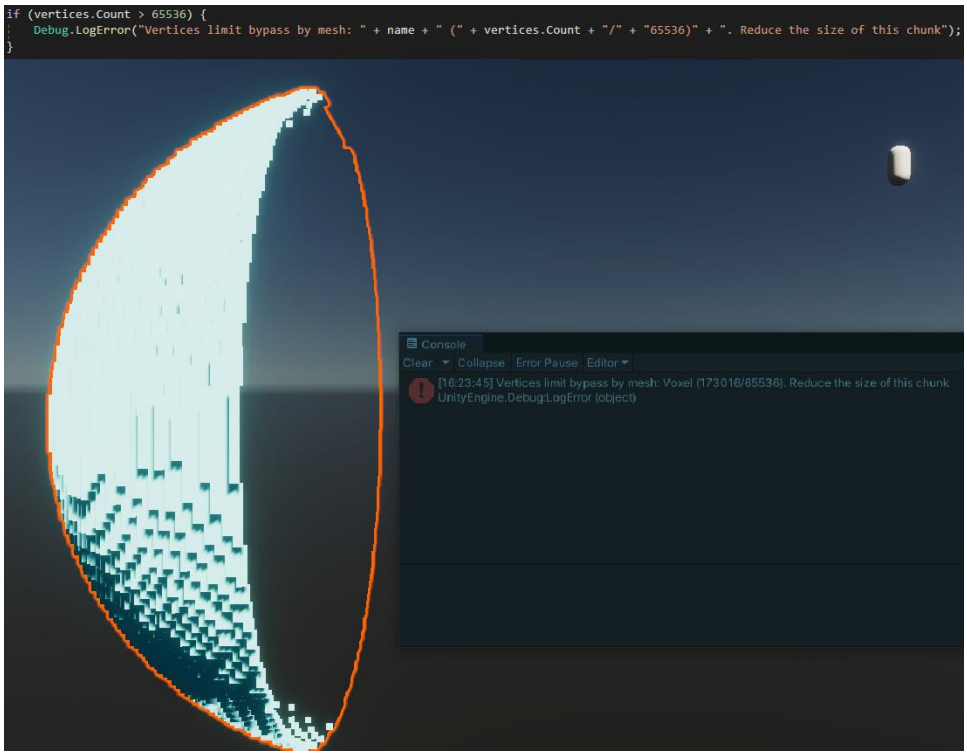

Figure 6.17 Voxel generated with too many vertices

6.8.4 Marching Cubes

Marching cubes use a similar principle as voxel where a 3-dimensional scalar field is used. But rather than storing information about the voxel itself, the vertices of the 'cubes' are stored instead and used to create a polygonal mesh. The vertex map can be generated the same way using a 3D Perlin noise function along with the random generation variables. Marching cubes algorithms are also used in medical visualizations such as thyroid computer tomography imaging [34].

As there are 8 vertices in a cube, each vertex can be considered 'inside' or 'outside' of the shape where the face normals are flipped. That makes only 14 possible unique configurations of a marching cube as shown in Figure 6.18 where triangles are drawn based on the active vertices and 256 different possible configurations where each are just symmetries of the 14 unique shapes.

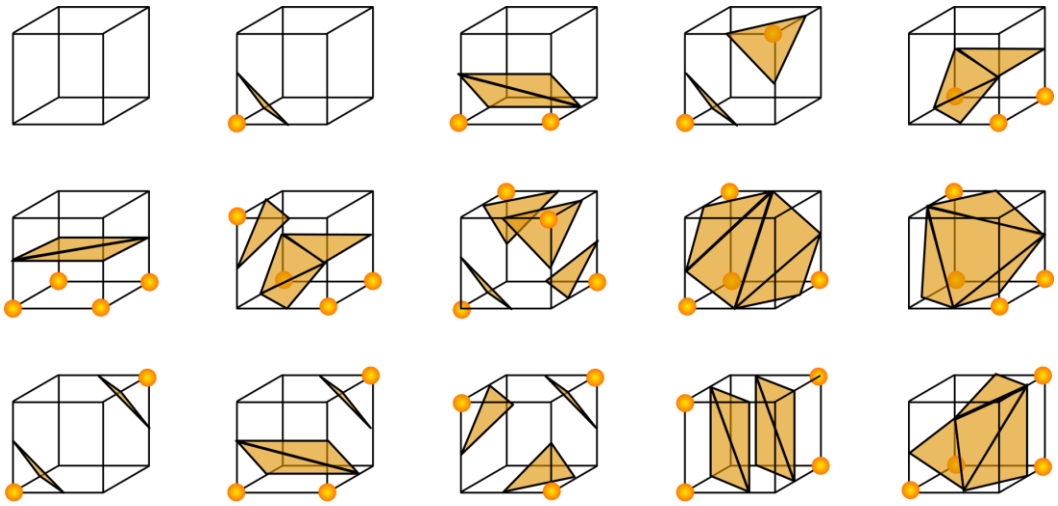

Figure 6.18 Marching cubes unique configurations

When reading the vertex map, all vertices of a cube at an index are read from the vertex map to then be used to look up a table of configurations which store indices to an edges table which is used to create the triangles by using the start and end points of the edges as the vertices order of the triangles. Instances of generated marching cubes are shown in Figure 6.19.

Figure 6.19 Marching cube instances with different generation variables

6.8.5 Texturing

Texturing voxel is done by creating UV maps for each face while drawing the face itself. When looping over vertices to create a triangle, the UV of that triangle can be mapped in a normalized 1x1 UV map. A texture can then be applied to a material that the mesh uses to display the texture in each square face. Texturing marching cubes is a bit more complex as the mesh is not cubic so when UV mapping the polygon, texture seams will become very apparent and the texture on some faces may become stretched. Instead of UV mapping, tri-planar texturing is used to texture marching cubes by projecting a texture from each axis and blending them [35]. However, as this game's low poly art style does not make use of textures, texturing both voxels and marching cubes can be done easily by UV mapping each face to a single point on a colour palette as shown in Figure 6.20. As this is a 16-bit colour palette, to map a UV at a colour, the

coordinates of the colour are divided by the image size to get a normalized coordinate of the UV map which is used to subsequently map the UV to that solid colour.

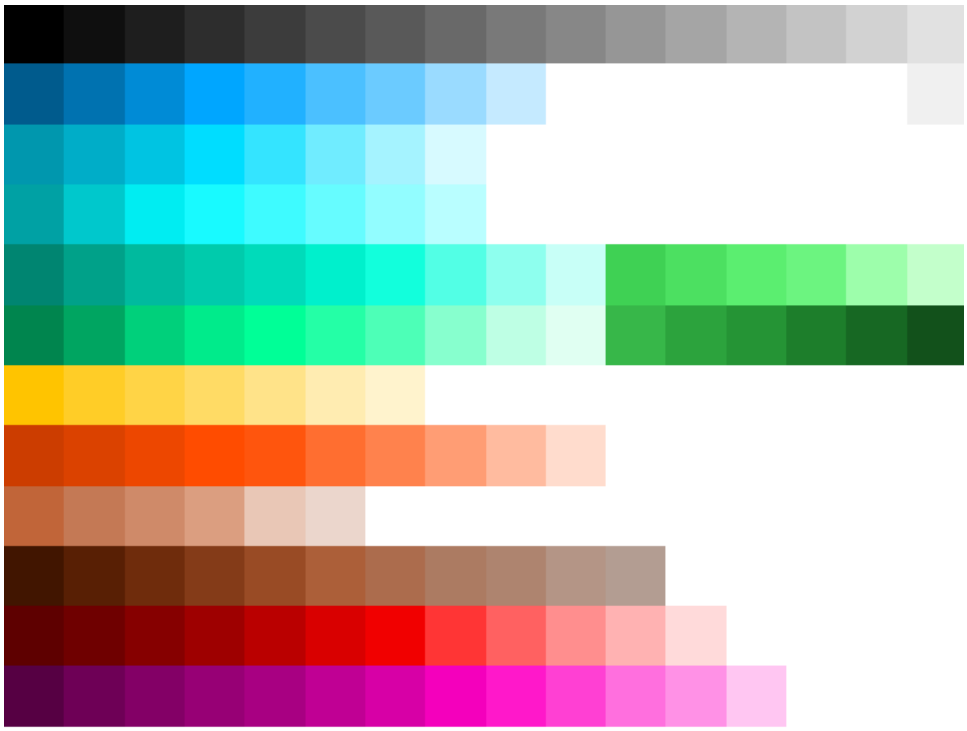

Figure 6.20 Colour palette used in this project with space for more colours

6.9 Data Persistence

A data persistence system is implemented inspired by Trever Mock (2022) to save the player's current progress I.e., changes to the world the player has made allowing the player to return to the exact same state as they left the world between different play sessions.

6.9.1 System Design

The DataPerstistanceManager class is responsible for managing the state of the game along with the current state of the saved game data. An interface is used to track objects to save and load by getting references to all objects that implement the interface and calling the appropriate method. Figure 6.21 visualizes the system in a UML diagram.

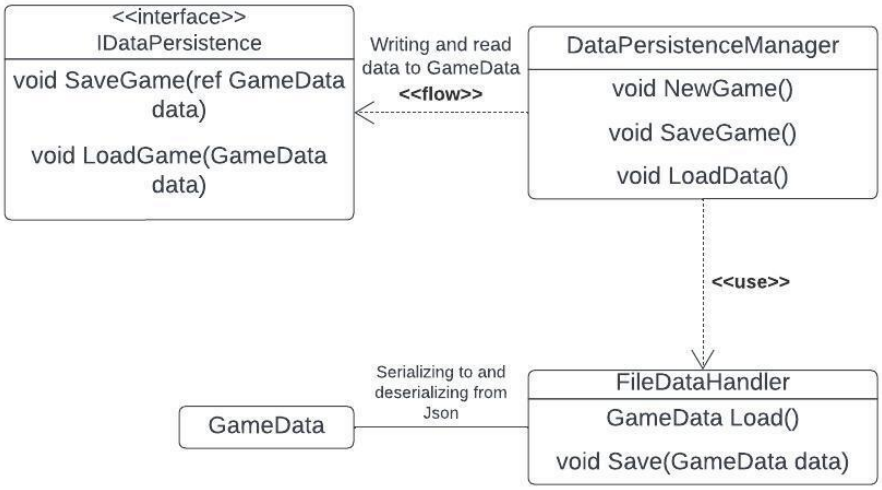

Figure 6.21 UML diagram of data persistence system

6.9.2 Serializing to JSON

When saving the game, a GameData instance is passed as a reference to savable objects where they can change and add necessary properties. Upon writing into the data, the GameData is then passed to a FileDataHandler where this class will then be serialized into a JavaScript Object Notation (JSON) format and stored on the local machine. Loading the data will deserialize this save file back to a GameData format to then be passed to all objects that implement the interface to read and load from. This can be done with Unity's JsonUtility class which provides a FromJson and a ToJson static method. JsonUtility is used over .NET JSON solutions as benchmark tests indicate faster speeds than the other [36]. The JSON serializer supports any Monobehaviour subclass, ScriptableObjects subclasses or plain classes or structs with the '[Serializable]' attribute. Unity serializes fields and simple data structures excluding dictionaries which can be replaced with a list of a custom class or struct with key and value fields.

6.9.3 Saving the Data

When saving common items such as resources, items or building structures, their position, rotation, and an identifier to tell what needs to be saved. This identifier can be the name in the ItemScriptable instance that all items and buildings will have. This can be used as a key to look up a dictionary of corresponding prefabs for the tag to load back into the world. Unique cases such as weapons or storages are different cases where a separate data structure stores the tag, position, rotation and weapon information or an array of item tags. This is to store each unique instance of each weapon and storage and load them back in correctly with the right position and right properties. Weapons will be set to their correct durability or ammo and storages will be assigned the correct item and resources in the correct order from where they were left off. Figure 6.22 shows an example of a custom class used to store information on each storage. When loading the map, the storage prefab is instantiated using its tag and its items including unique items such as weapons are placed in the storage.

```
[System.Serializable]
6 references
public class StorageData
{
    public string tag;
    public Vector3 position;
    public Quaternion rotation;
    public string[] items;
    public WeaponData[] weapons;

    1 reference
    public StorageData(string _tag, Vector3 _position, Quaternion _rotation, string[] _items, WeaponData[] _weapons)
    {
        tag = _tag;
        position = _position;
        rotation = _rotation;
        items = _items;
        weapons = _weapons;
    }
}
```

Figure 6.22 StorageData class storing persistent data of a storage

6.9.4 Security

The saved data is stored in Unity's standard directory for persisting data which is returned by the Application.persistentDataPath property in the Application class. This file can be encrypted to protect against unwanted modifications to the save file. An easy encryption method that is built into C# is the binary formatter which takes data and serializes it into a binary format. Although simple to implement, Microsoft highly advises against using the binary formatter as it is insecure and cannot be made secure [37]. Another encryption method that was attempted to be used is the XOR encryption algorithm where each character in the file undergoes an XOR operation with another character which may be a key resulting in a random string of characters that is illegible. This is much more secure and simple to implement, however, due to the current number of characters stored within the file this process takes exceedingly long to execute so for now it is left unused until a more efficient way of saving data is implemented.

6.10 Difficulty Levels

The game has five different difficulty levels, and each has varying settings to affect gameplay. Figure 6.23 shows all the variables that affect the gameplay which can be edited for each difficulty. Each difficulty is stored in a ScriptableObject to be used to read from and initialize values of shown variables for the appropriate classes. The player can select their desired difficulty when starting a new game and this difficulty is persistent throughout the game save.

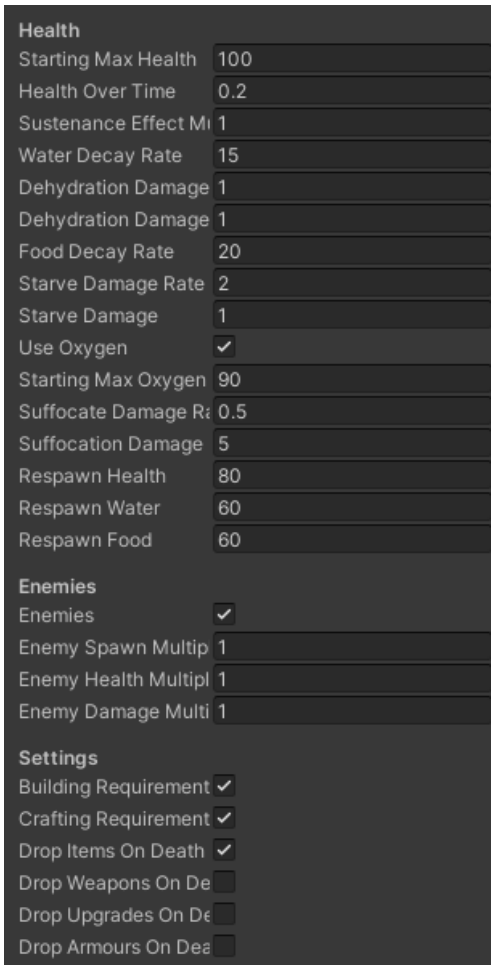

Figure 6.23 Intermediate Difficulty ScriptableObject

7 Art and Design

The game follows a simple art style using low poly models and simple solid textures to provide an appealing and friendly theme.

7.1 Animation

Various aspects of the game include animations such as enemies, the player and even the main menu. Unity's built-in animation system is used to create animations for the project allowing the use of dope sheets and animation curves. In the animation window, recording for keyframes can be enabled effectively recording and writing changes to the model when changed within the scene view. This allows for rough drafts of the animation with can then be fine-tuned with the dope sheet and curves; The dope sheet can help to adjust the timing of the animation events such as attacking, and the curves can be adjusted to calibrate smoothing.

7.1.1 Rigging

For humanoid animating, the model must be rigged for animations to control each limb of the model. This generally required an armature or a series of connected bones and aligning and calculating the weight of each bone to the target limb model resulting in the model moving along with the bone/armature. Additional techniques can be applied such as inverse kinematics to the rig which creates an effect where if the child joint is moved such as the hand, the parent joints will move and rotate automatically to achieve a believable look [38].

7.1.2 Overriding Existing Animation Controllers

Along with an Animation Controller asset that provides an animation state machine to control animation, the Animator Override Controller allows you to extend an existing Animation Controller keeping the overall structure and logic of the original controller but allowing overriding animations for states. Figure 7.1 shows the overridden animations where the 'Attack' and 'Move_Target' animations, shown in Figure 6.11, are replaced with a melee counterpart ultimately playing a different animation when attacking or moving to the player while hostile.

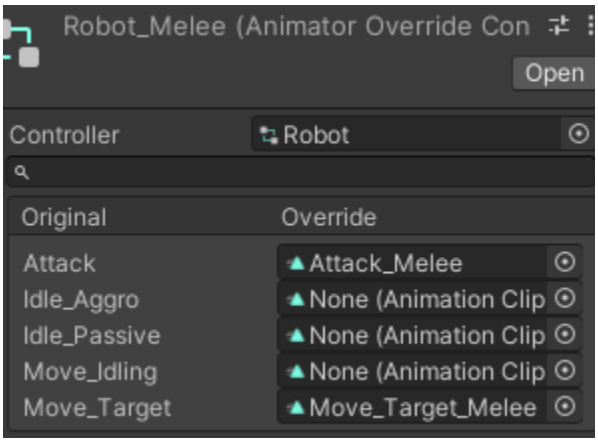

Figure 7.1 Animation Override Controller for an enemy

7.2 Modelling

Most models used in the game are modelled using Blender [14] where a low poly approach is taken where the size of the object is considered. To create a consistent look with low poly models, the number of subdivisions becomes the dependent variable when the size changes. Figure 7.2 by PontyPants [10] shows the effect of decreasing the subdivisions along with the size of the shape creating cohesion.

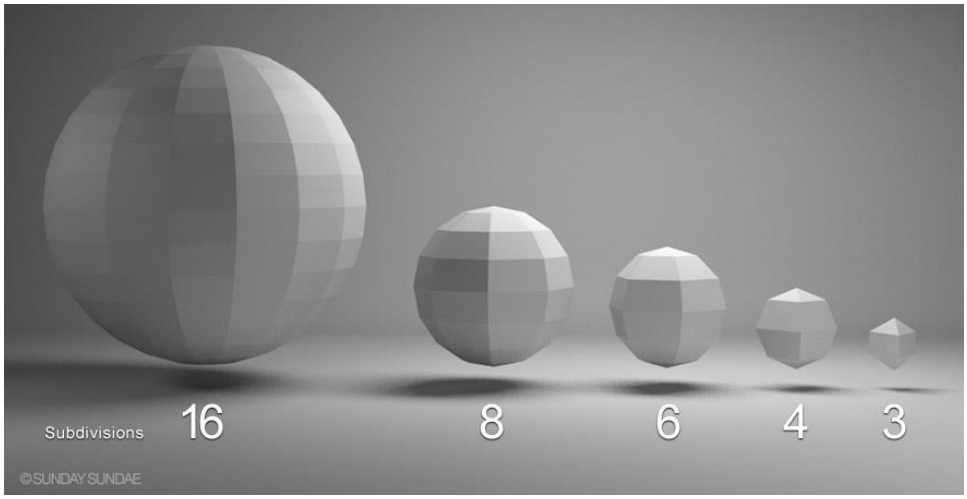

Figure 7.2 Shrinking sphere size with decreasing subdivisions

7.2.1 Remapping UVs

Colours are applied by remapping the mesh UVs to a colour in the colour palette texture as shown in Figure 7.3. When the models are exported from Blender to Unity, a new Unity material is created with the same colour palette texture and applied to the model. This allows many variations of materials with the colour palette from transparent to metallic.

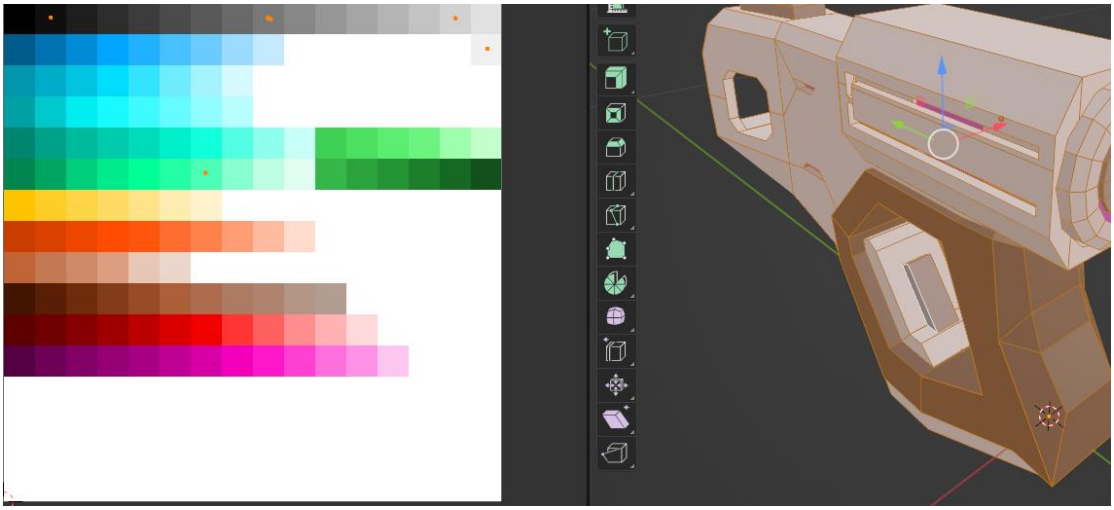

Figure 7.3 UV remapping of the energy pistol model

7.2.2 Model Cleanup

After modelling with the use of creating subdivisions, these usually leave a bunch of unwanted vertices that do not affect the model in any way. The decimate modifier can be used to eliminate these vertices without affecting the overall model. Figure 7.4 shows an ammo box model with subdivided surfaces causing redundant vertices.

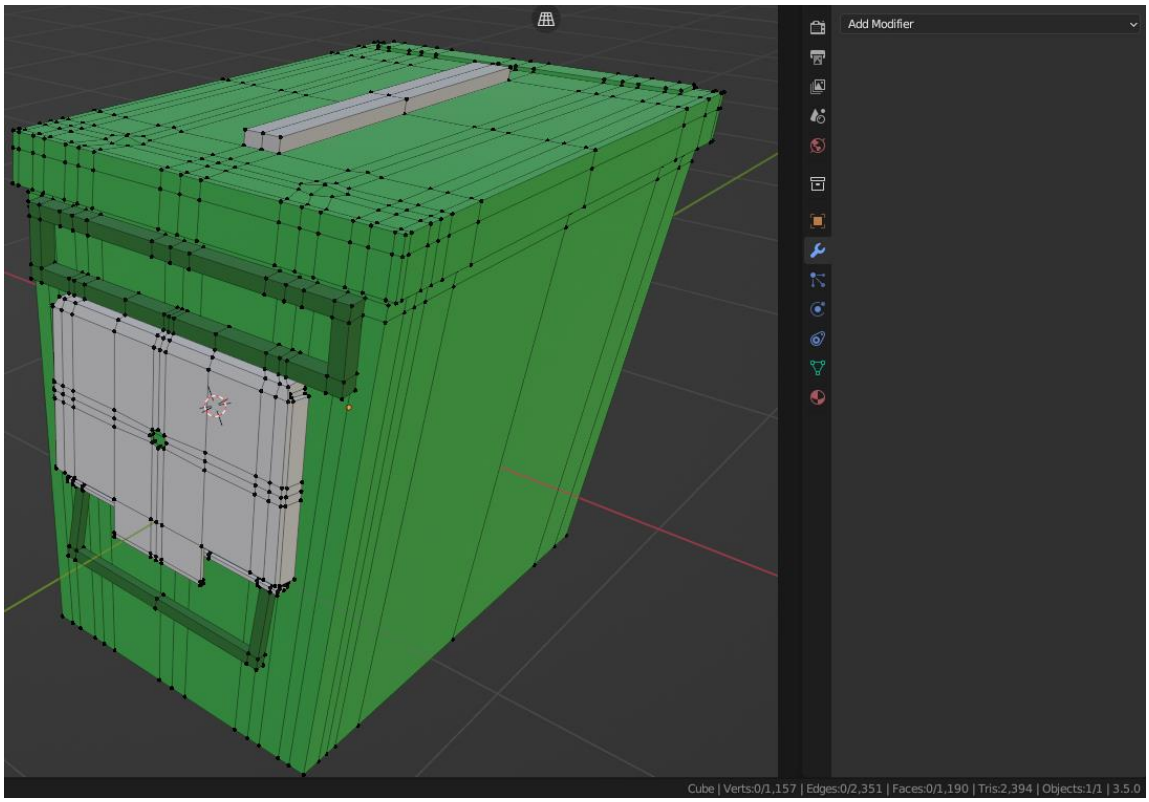

Figure 7.4 Ammo box before decimate modifier applied with model stats shown in bottom right

The decimate modifier with the planar option is applied with an angle limit low enough to not change the overall structure of the ammo box resulting in a vertex count of less than a quarter of the original model as shown in Figure 7.5.

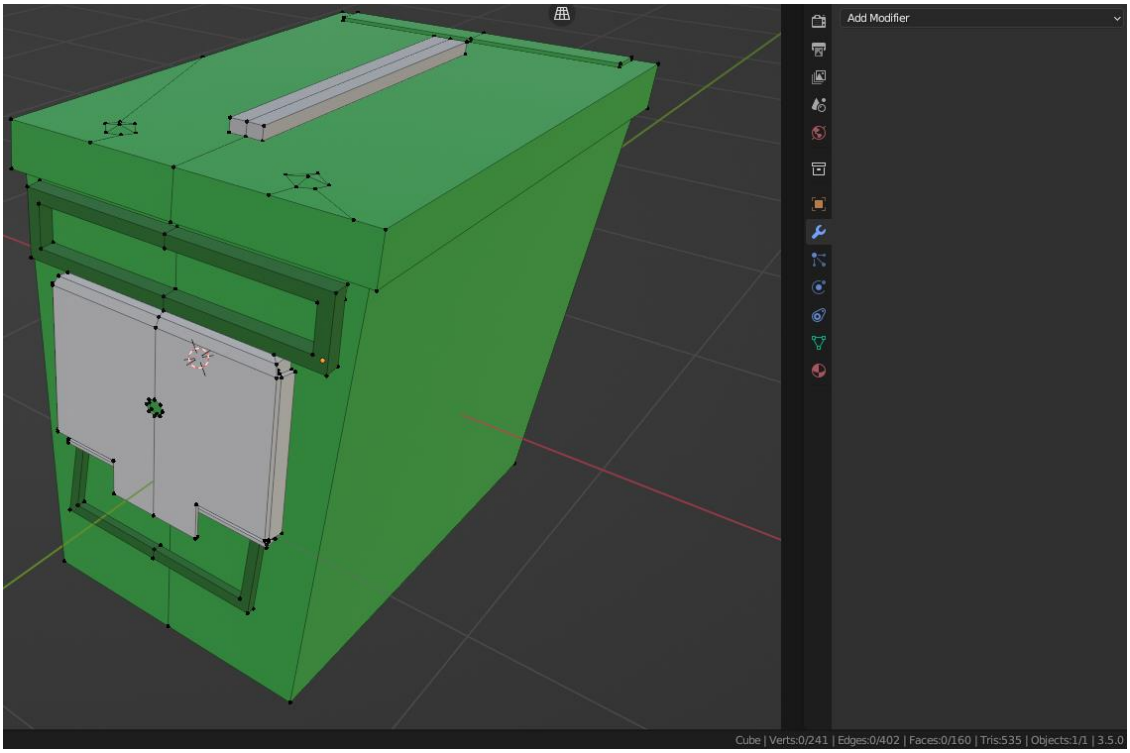

Figure 7.5 Ammo box after decimate modifier applied with stats cut down to less than a quarter

7.2.3 Level of Details

When modelling detailed models, they will generally have a high vertex count and the full mesh will be rendered when visible in the game. However, as the player gets further away from these objects, they will naturally notice fewer details of these objects, but the mesh still gets rendered in full resolution. One way to tackle this performance issue is to use Level of Details (LOD). In Blender, it is possible to automatically create a preset LOD system in the scene collection hierarchy just by naming [39]. In Figure 7.6, different models of ice are named according to their level of detail with the most detailed being '0'. When exporting this file, all the objects are selected and exported as one file and imported into Unity.

Figure 7.6 Auto LOD setup in Blender

Unity will detect this and automatically create a game object with the LOD Group component with their set references to the meshes as shown in Figure 7.7.

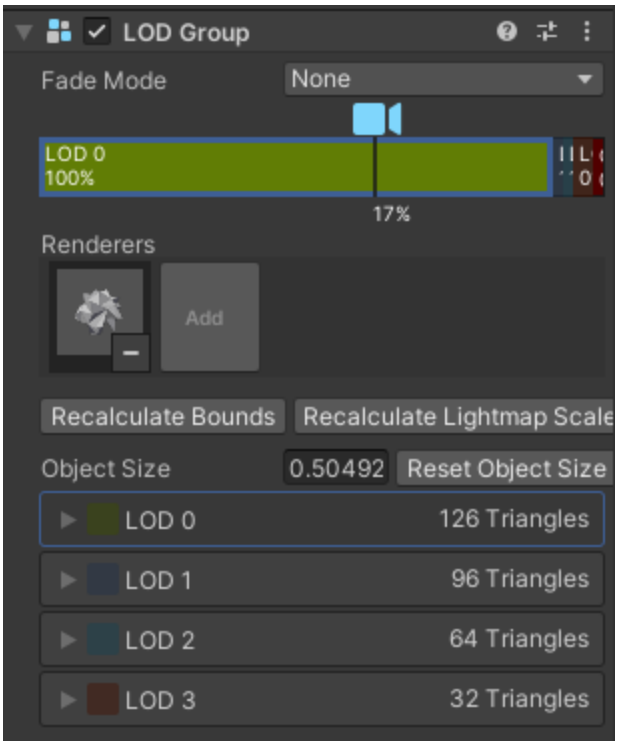

Figure 7.7 LOD Group component on imported ice Blender model

In Blender, the decimate with the collapse option can be used to reduce the model's level of detail to the desired detail level and subsequently named.

7.3 Shaders and VFX Graph

In Unity, shaders and VFX can both be created using Unity's built-in Shader Graph and VFX Graph, respectively. Both use a node graph design approach where developers can create these art assets by connecting nodes that provide different effects together. Each node has a specific functionality and can have inputs and outputs allowing for connected nodes to create an ultimate effect. Node graphs accommodate a simplified view for developers who are not too familiar with writing the source code itself. Visual scripting in general can become limited to complex functionalities as the user is relying only on the logic of connecting nodes or it may be hard to understand how to get a certain effect using the nodes. However, the user can experiment freely and swiftly with the node to explore all types of effects just by connecting random nodes and it is generally easier to understand and learn. Code Monkey created a full role-playing game using purely node-based visual scripting with many features such as lives, enemies, attacking and more [40] with Unity's Bolt visual scripting.

7.3.1 VFX Graph

In this project, VFX graph is used to create VFX used throughout the game. An instance of this is the energy orb effect which consists of multiple particle systems to create different effects shown in Figure 7.8. Properties can be set on the left side to be changed easily when inspecting an instance of this VFX for variations.

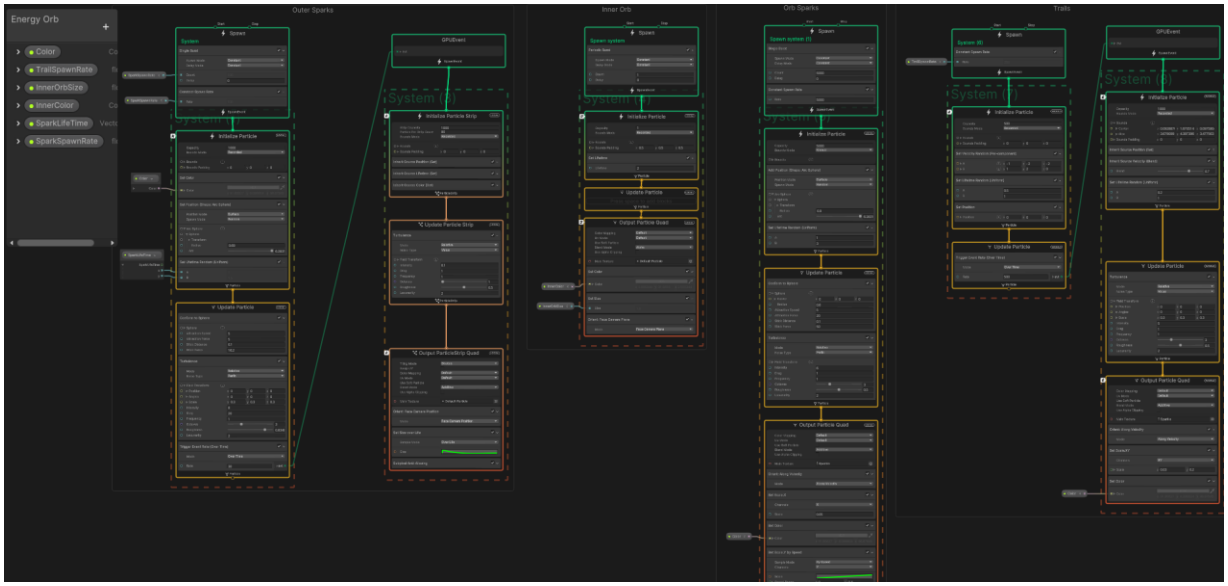

Figure 7.8 VFX graph of energy orb effect

A VFX system consists of 4 main stages; spawn, initialize particle, update particle and output particle. The spawn stage defines how many particles spawn and how often they spawn at what rate. Initialize particles control how each particle is spawned and their initial properties, for example, ‘Set position (Shape: Arc Sphere)' will set each particle spawned in a sphere shape with a set radius. Update particle will constantly apply the given effects to each particle over time, for example, the ‘Turbulence’ block will apply forces to each particle constantly making them move in seemingly random directions. Finally, the Output particle stage draws the particle. The particle can be output in many different forms such as meshes, shapes and decals and its material properties. Before outputting, particles can be made to emit further particles by adding a Trigger Event Rate block in the update particle stage and connecting that to another system to emit particles themselves. An example of this is shown in Figure 7.9 where each particle emits ‘strips’ moving along with the random moving particle creating an electrical arc effect.

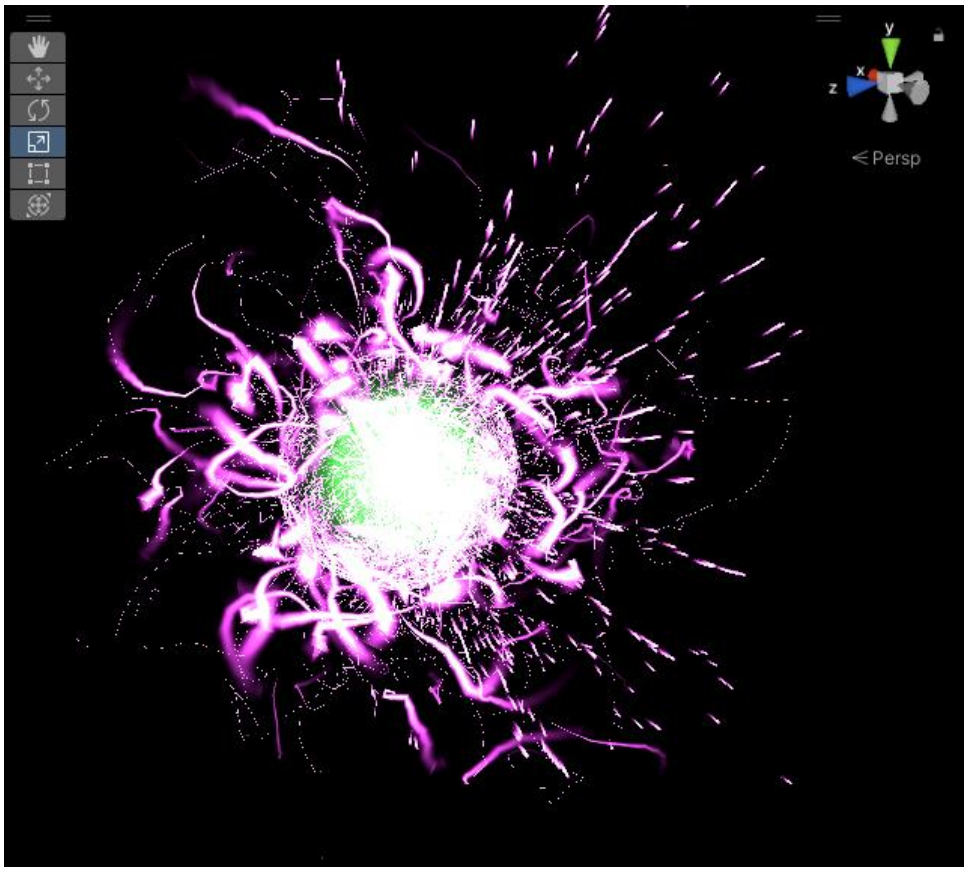

Figure 7.9 Energy orb effect preview

7.3.2 Shader Graph

Shaders are also used to create unique effects with materials. The Fresnel effect is used in the 'Glow' shader to create an outer glow based on the perspective of the observer giving a multicolour glowing effect as shown in figure 7.10. The Fresnel effect is used to calculate which colour is displayed based on the angle of incidence between the observer and the surface [41] where a smaller angle of incidence I.e., the more perpendicular the observer views a surface, the weaker the Fresnel effect resulting in the original material/colour being shown and the larger the angle on incidence for example the side of the cube, the stronger the Fresnel effect.

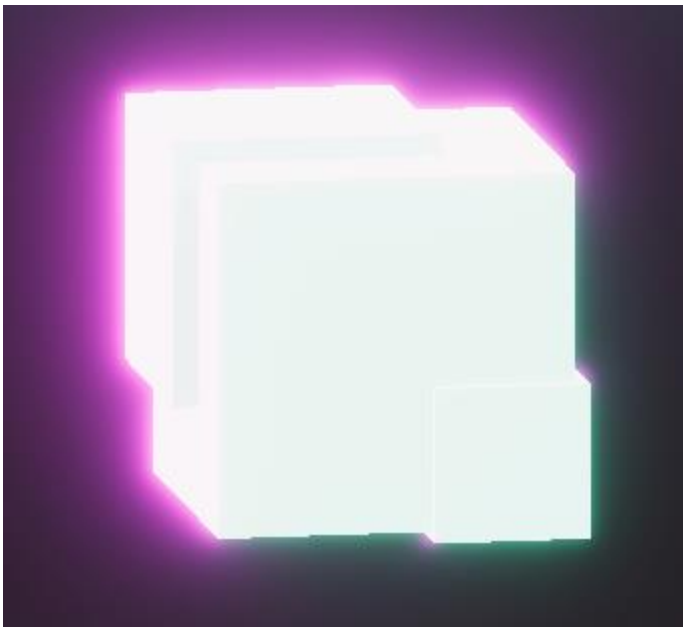

Figure 7.10 Energy cube with the Glow shader emitting purple glow along with its base green glow

This Fresnel glow effect can be replicated by multiplying a Fresnel node with an emission colour of choice as shown in Figure 7.11.

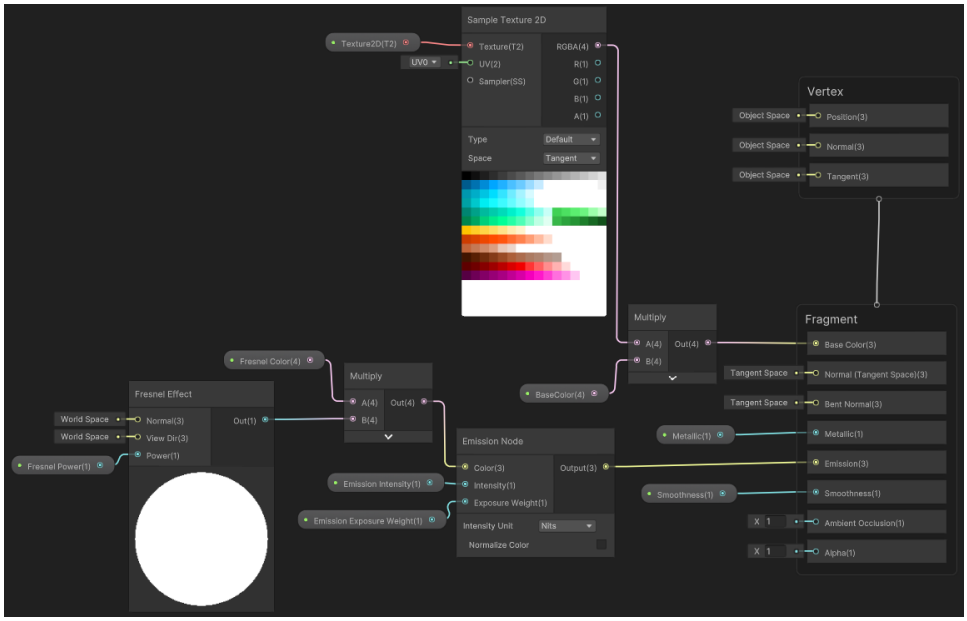

Figure 7.11 Glow shader in Unity' shader graph editor

7.3.3 Outline Shader

Shader graph is also used to create an outline shader which produces a thick line around a given mesh to highlight objects of interest easier for the player as objects are texture-less and use the same colour palette making them hard to spot. There are many different methods of creating an outline effect using

shaders [42] such as the rim effect method which uses the Fresnel effect to create outlines at surfaces with high angles of incidence, a silhouette buffer method where objects are rendered onto a separate render buffer and blurred or enlarged over the normal objects and render on top of them giving an outline effect, a jump flood algorithm which is explained in depth by Ben Golus [43], edge detection method where outlines are rendered at areas with a large discontinuity between them, or a vertex extrusion method which is used in this project.

Figure 7.12 shows the shader graph of the outline shader. Each vertex of the object is extruded along its normal and normalized to ensure that the length of extrusion is equal throughout the model. These vertices are then added on top of the original positions of the model vertices and a solid colour is applied to create the solid outline effect. The object is then rendered behind the original object to create a border outline around the original object.

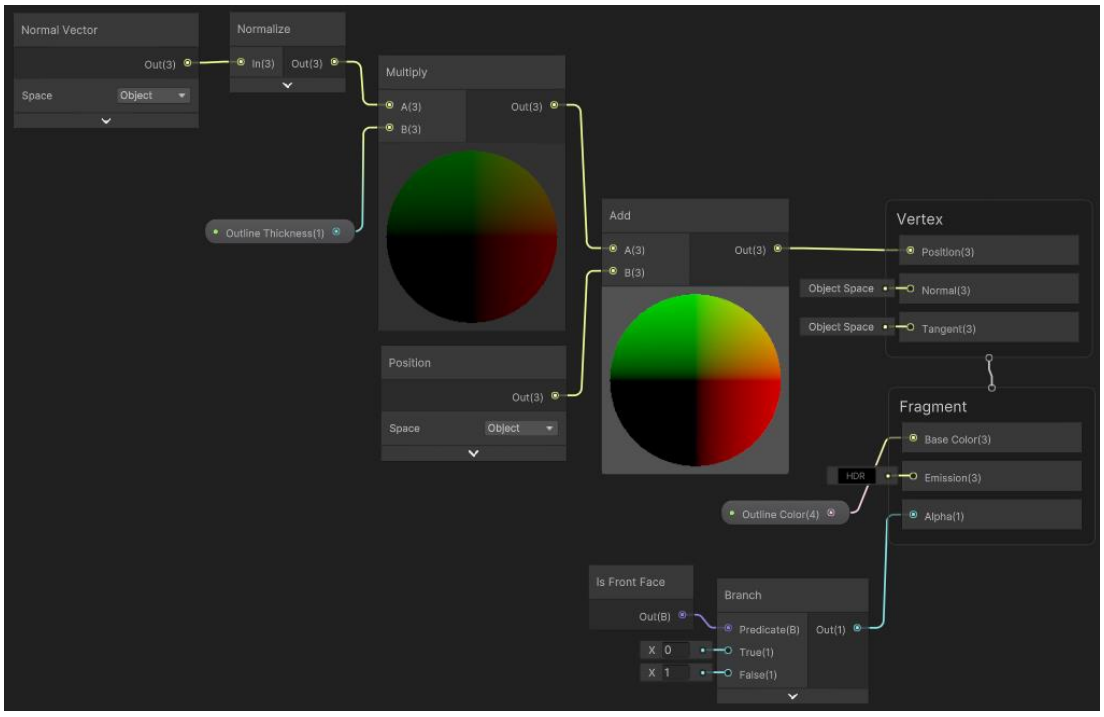

Figure 7.12 Outline shader in shader graph editor

To apply the outline, a separate child game object with the same model uses the shader and is toggled by a separate ‘Outline’ component shown in Figure 7.13.

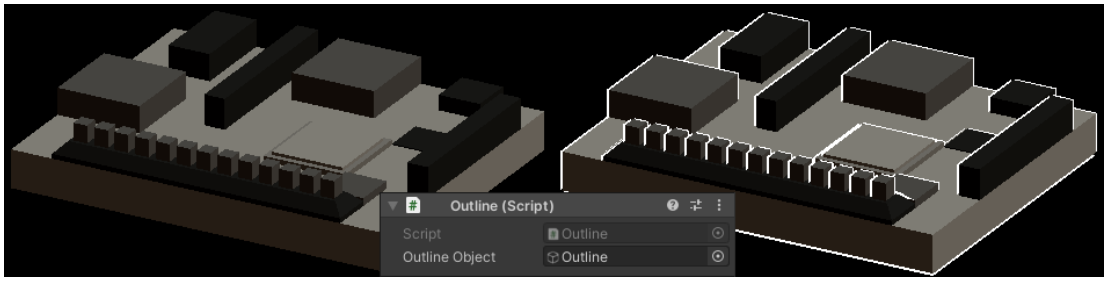

Figure 7.13 Outline example with outline component

7.3.4 Background Shader

Blender is also used to create a space background shader with a node graph system as shown in Figure 7.14. A noise texture node is used to create white noise to act as stars and larger noise to create coloured patches. This image is then rendered as a high dynamic range image (HDRI) and imported into Unity. This image is then converted into a cube map ready to use as a skybox in the scene.

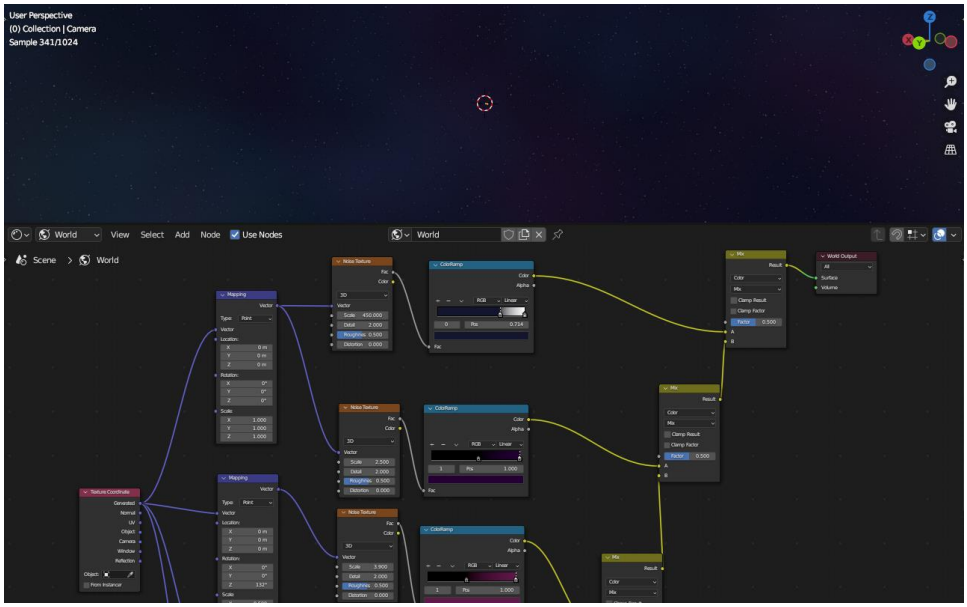

Figure 7.14 Space background shader created in Blender

7.4 Sounds

Sounds are simply added to the project by adding audio clips to an Audio Manager class instance through the inspector along with a tag to identify each clip. Multiple clips can be added to a single tag so when an audio clip is requested, a random clip is picked from the list of clips for variation as shown in Figure 6.2.

Audio clips found online, referenced at the end of the report, are trimmed using the free digital audio editor, Audacity [44]. These clips are added, and their pitch and volume can be modified directly in Unity. Unity also provides basic audio effects such as a low pass filter which removes frequencies from a sound signal above a set cutoff frequency. This can be used and applied to audio clips when the player is not within oxygen to give a muffled effect simulating them being in space.

7.5 World Design

The world is designed based on the brief drawing shown in Figure 7.15 where landmarks and points of interest are marked as shown on the right side.

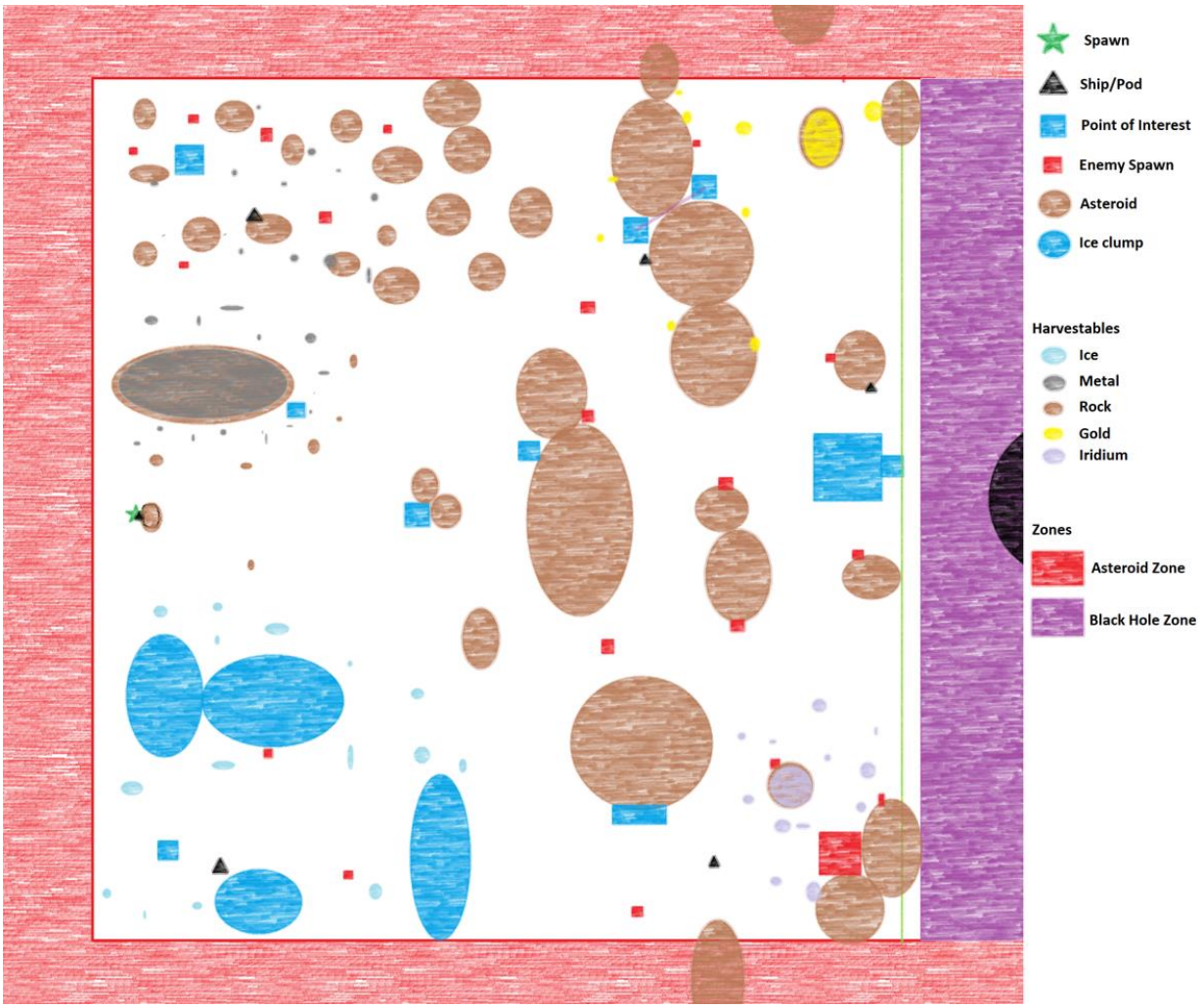

Figure 7.15 Orthographic view of world with points of interests highlighted

The world is designed to closely reassemble the drawing shown in Figure 7.16 keeping the main aspects and design for progression; Rare metals are on the further side of the map which can only be accessed through playtime and by creating safe zones to regain oxygen and stay safe in to progress there. Enemy spawns are also strategically placed to encounter the player upon their explorations.

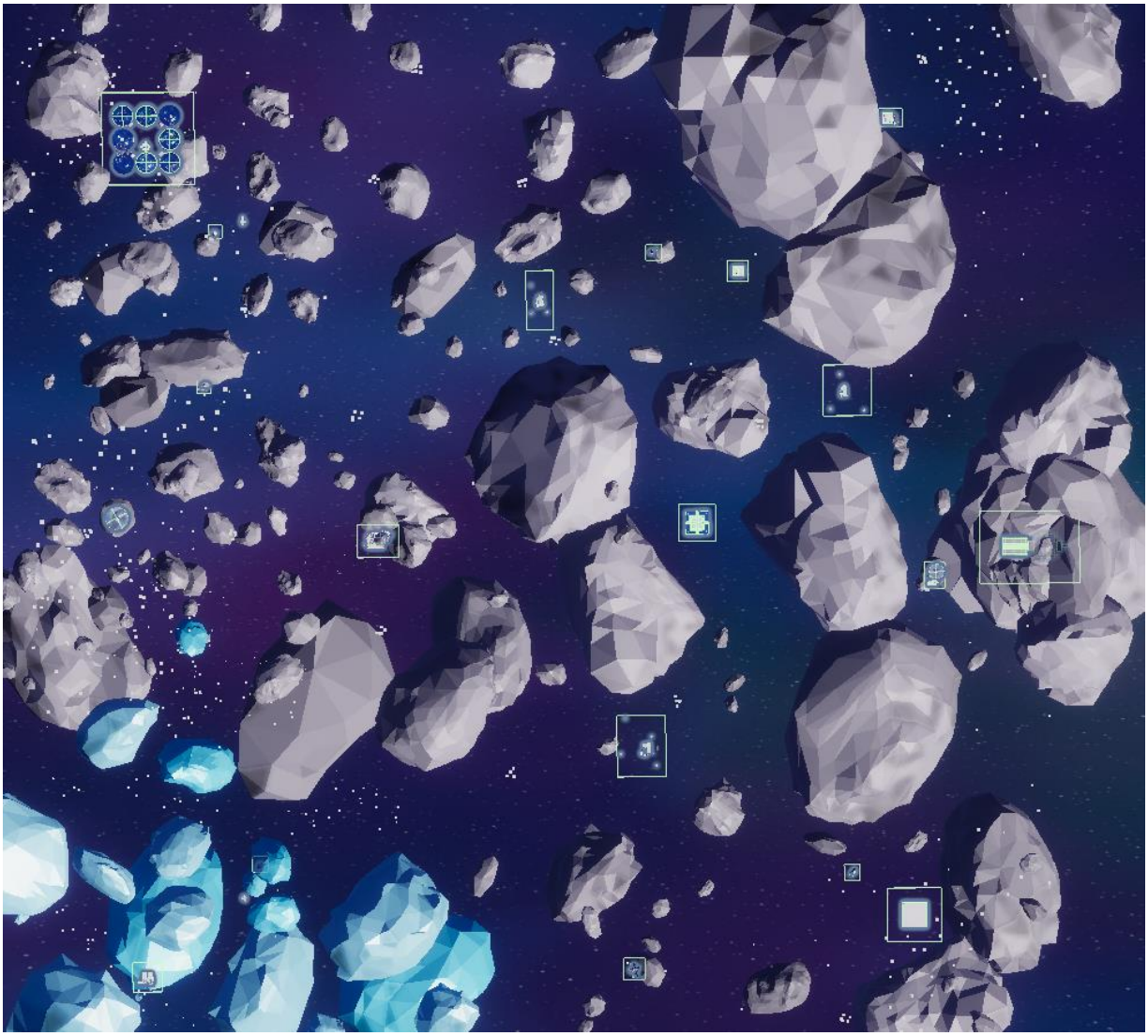

Figure 7.16 Orthographic view of world with points of interests highlighted

Harvestable marching cubes and voxel nodes are placed around the map to be generated. As they are not generated during scene editing, Unity's Gizmos [45] is used to draw a cube with the size of the node to indicate where and how large they will generate as shown in Figure 7.17.

Figure 7.17 Node indication of their generation

7.5.1 Story

The story is kept simple due to time constraints. Around the map are landmarks that the player can visit where the player is rewarded with items they can loot. In some landmarks, there are unique-looking items that they can collect and store in a separate inventory. These items later can be used upon the player discovering an end-game landmark where they can use these items for an objective for the story. Upon completion, the player can activate a machine that reveals a black hole, designed by Mert Kirimgeri (2020), within their peripheral prompting them to move towards it. The player can then interact will a trigger wall which will suck the player into the black hole ending the game, displaying a game-ending message and menu where they may continue playing or return to the menu.

A small mini-game is added to the game where upon entering a room shown in Figure 7.18, the player must fend off enemies that constantly spawn and at the same time destroy an object for a story item to progress.

Figure 7.18 Mini game room

7.5.2 Boundaries

A World Bounds class is used to keep the player within the play area by firing asteroid projectiles at the player if they reach and exceed the borders of the map. Variables can be adjusted to change how effective and harsh the border is shown in Figure 7.19.

Figure 7.19 World boundary manager instance

7.5.3 Occlusion Culling

After finalizing the map, occlusion culling is baked into the scene which prevents Unity from performing rendering calculations for game objects that are completely hidden from view by other game objects [46]. This is done by flagging objects to hide other objects as occluder static and objects to be hidden by occluder objects as occludee static in the static dropdown for each game object.

A small issue to consider especially when making an external map is the internal parts of the map that the player can enter. In Figure 7.20, as the very bottom room is completely sealed off from the outer world, Unity does not know that players can enter by using the hatch so when the player enters this room, Unity has not calculated any occlusion data for this room resulting in missing objects.

Figure 7.20 Teleportation facility interior with many levels

8 Testing

To deliver a fully functional and entertaining game, it must be tested to ensure all features work as intended as well as work in such ways as to provide a fun experience.

8.1 System Testing

8.1.1 Unit Testing

As the features were implemented separately, unit testing is possible for each feature along with using these features with others where different scenarios are taken into account and tested. Below is a test table with a 99% success rate with only one conditional pass.

Feature	Functionality	Inputs	Expected Outputs	Actual Outputs	Pass/ Fail
Player Movement	Movement with gravity	Horizontal movement, sprinting and jumping	Player traverses on surface, speeding up upon sprinting and jumping only when touching floor	Player walks on surface, runs on sprint and jumps only when touching floor	Pass
	Movement without gravity	Horizontal movement, vertical movement, Speeding up	Player can move in all directions freely and speed up	Player moves in all directions and speeds up on demand	Pass
	Movement transition from gravity to without	Horizontal movement, jumping and sprinting transitioning to vertical movement	Player should only walk and jump on surface in gravity field. Upon exiting field, the player can float in all directions	Player walks on surfaces and jumps and upon exiting gravity field, they can float in all directions	Pass
Inventory System	Picking up items	Pick up standard item	Player interacts with an item which disappears. Item's information is then display to a vacant slot	Item disappears when player interacts with it and the item's icon is shown in the slot	Pass
		Pick up a weapon	Player interacts with weapon which will disappear, and	Weapon disappears upon interaction and the	Pass

			its info will display on a slot	weapon icon is shown in the slot	
	Full inventory check	Pick up an item when inventory is full	Player interacts with the item which should not disappear, and inventory should remain unchanged	No effect on item and inventory's content remains unchanged	Pass
	Consume items	Pick up consumable item and consume it	Player picks up the consumable and presses use key to consume it, supplying their benefits	Consumable is picked up and upon use key, consumable provides set benefits	Pass
	Item swapping	Drag and drop an occupied slot with another	Player drags a slot, following the cursor and upon release, icon will swap with the hovered slot	Slot follows the cursor upon drag and on drop, slot is switched with the hovered slot	Pass
	Item display	Hover over empty slot	Item display should remain blank	Item display remains blank	Pass
		Hover over slot with item	Item display should display item's name, icon, and description	Item display shows the item's name, icon, and description	Pass
	Special equipable items	Pick up shields and equip to its slot	Player picks up shield item and equip to the shield slot, providing its bonuses	Shield item is picked up and equipped to its slot. Maximum shields are increased to set values	Pass
	Drop items	Drag and drop item slot outside of inventory	Item should respawn back into the world and disappear from the inventory	Item returns to world upon dropping item slot outside of the inventory and slot it emptied	Pass
Weapon System	Equipping and unequipping	Drag and drop weapon icon to hot bar	Weapon slot should display weapon and hot bar should update and weapon should display on player's hands	Weapon is displayed on the weapon slot, hot bar, and player's hands	Pass

	Ray Weapon	Attack and damage objects	Ray weapon shoots and instantly damaging object	Weapon shoots on attack and instantly damaged object	Pass
	Projectile Weapon		Weapon shoots a projectile which should explode shortly damaging surround objects	Projectile is spawned and objects are damaged upon explosion	Pass
	Melee Weapon		Melee weapon attacks and instantly damaging object within a range	Weapon damages objects within the set range	Pass
	Swapping weapons in hot bar	Number keys from 1-6	Hot bar displays the weapon at the corresponding key pressed	Correct weapon at hot bar is displayed upon number keys pressed	Pass
		Scroll wheel up and down	Hot bar displays switches with correct direction	Hot bar navigates in correct direction displaying the corresponding weapons	Pass
Crafting System	Opening crafting interface	Correct set of recipes displayed on opening corresponding crafting system	The set of recipes assigned to crafting manager should display upon opening a crafting table of that set	Crafting interface displays correct recipe content	Pass
	Crafting an item	Craft a standard item	Items required to craft should be cleared and product item is added to inventory	Required items are cleared and product item is added to inventory	Pass
		Craft a functioning weapon and attack	Weapon is crafted with required items and equipped and used successfully	Required items are cleared and weapon is added and equipped and successfully attacked with	Pass
		Craft a consumable and use	Consumable is crafted with required items and	Required items are cleared and consumable is	Pass

			used providing its benefits	added and used successfully providing benefits	
		Craft a shield and equip	Shield is crafted with required items and equipped providing shields	Required items are cleared and shields is added and equipped providing max shields	Pass
		Attempt to craft without ingredients	Item should not be crafted, and inventory remains unchanged	Item is not crafted, and inventory is unaffected	Pass
Building System	Opening interface	Open interface with tool gun	Interface opens displaying all buildable recipes	Interface opens and all buildables are shown with their required ingredients	Pass
	Building structures	Select a recipe to build with sufficient items	Interface closes and blueprint of buildable should display	Interface closes upon click and blueprint of buildable is displayed	Pass
		Select a recipe to build with insufficient items	Interface remains open and nothing should happen	Nothing happens upon click	Pass
		Build a floor	Another instance of the buildable should spawn while blueprint remain if enough items remain to build another after clearing the required	Floor is built and items are cleared. Sufficient items remain to keep building to blueprint correctly remained	Pass
		Build a wall		Wall is built and items are cleared. Sufficient items remain to keep building to blueprint correctly remained	Pass
		Build a floor on an existing floor	Nothing should happen	Nothing happened. Item remained	Pass

		Build a wall on an existing wall		Nothing happened. Items remained	Pass
		Build a decorative structure	Selected structure should be built consuming required items	Required items are cleared and structure is built	Pass
		Build a decorative structure overlapping another	Nothing should happen	Nothing happened. Items remained	Pass
	Seal Check	Build a 1x1x1 and check if sealed	Player should be granted oxygen	Player is granted oxygen	Pass
		Build a 2x2x2 and check if sealed		Player is granted oxygen	Pass
		Build a 5x5x5 and check if sealed		Player is granted oxygen	Pass
		Build a random enclosed structure		Player is granted oxygen in some areas. Specific case of edges needing to be places within the system and facing outwards rather than wall placed outside of system and facing inwards to block air way	Pass & fail
		Build a random unenclosed structure with one hole	Player should not be granted oxygen	Player is not granted oxygen	Pass
		Build an enclosed room and unenclosed room in the same system	Player should be granted oxygen in the enclosed room and not in the unenclosed room	Player is granted oxygen in enclosed room and is not in the unenclosed room	Pass
Resource Harvesting	Harvesting with tool types	Harvest with correct type of tool	Damage taken and object is hidden,	Damage taken and object is spawned upon harvest	Pass

			and its resource is spawned		
		Harvest with wrong type of tool	No damage should be taken	No damage taken by harvestable	Pass
		Harvest with sufficient tier level tool	Damage taken and object is hidden, and its resource is spawned	Damage taken and object is spawned upon harvest	Pass
		Harvest with insufficient tier level tool	No damage should be taken	No damage taken by harvestable	Pass
	Harvesting voxels	Harvest voxel with correct tool	Resource spawns at the position the voxel is broken at	Set resource spawns at the position the voxel is broken at	Pass
	Harvesting marching cubes	Harvest marching cube with correct tool	Resource spawns at the position the marching cube is broken at	Set resource spawns at the position the marching cube is broken at	Pass
Player Vitals	Changing vitals levels	Consume item that grants health	Vital level is increased by set value in the consumable	Vital level increased by the set value in the consumable	Pass
		Consume item that grants oxygen		Vital level increased by the set value in the consumable	Pass
		Consume item that grants water		Vital level increased by the set value in the consumable	Pass
		Consume item that grants food		Vital level increased by the set value in the consumable	Pass
		Equipping shield	Maximum shield should increase and shields gradually increase	Maximum shield level increased by set value and shields slowly recharged	Pass
	Managing vital levels	Player stays inside oxygen zone	Player's oxygen should increase and be capped at its maximum	Player's oxygen increased till maxed out	Pass

		Player exits oxygen zone	Player's oxygen should start ticking down in real time	Player's oxygen decreased till zero	Pass
		Player oxygen is at 0	Player should start losing health	Player starts losing health	Pass
		Player water level is at 0		Player starts losing health	Pass
		Player food level is at 0		Player starts losing health	Pass
		Player health is at 0	Console should log a message when health reaches 0	Console logs message when player health reached 0	Pass
	Shield and health	Player takes damage with shields active	Shields should negate the damage taken and start recharging shortly	Shields negated damage and recharged after	Pass
		Player takes more damage that current shield value	Damage should bleed onto health value where shield negates as much as it can, and health is decreased by remaining amount	Shield depleted and remaining damage is applied to health	Pass
Death System	Respawning	Player dies	Respawn menu shows and player can respawn by pressing respawn button	Player respawns upon pressing respawn	Pass
	Respawning at different set points	Player interact with cube to set spawn point and dies elsewhere	Player should respawn at the cube upon pressing respawn button	Player respawns at cube when respawn button pressed	Pass
	Item drops on death	Player dies with items in inventory. Set to drop all item	All items should be dropped upon death	All items dropped upon death	Pass
Enemies	Idle state	Player out of range of enemy	Enemy should randomly move and rotate to random places	Enemy moves randomly while player out of range	Pass
	Hostile state	Player in range of enemy's hostile range	Enemy should target and move towards the player	Enemy moves towards player when entered hostile range	Pass

	Attack state	Player in range of enemy's attack range	Enemy should attack the player	Enemy attacks and damages the player	Pass
	Dead state	Player kills the enemy	Enemy should disable and not do anything	Enemy is stopped immediately upon killed	Pass
	Enemy spawner	Game starts	Enemies should start spawning at the spawner	Enemies spawn at spawner	Pass
		Set maximum number of allowed enemies per spawner	Enemies start and stop spawning when the number of enemies currently active is at a max	Enemies spawn till maximum capacity. One is killed and spawner spawns another and stops at max again	Pass
Difficulties	Initializing with settings	Initialize with different maximum vital values	Player spawn with set maximum values per vital	Player spawns with set maximum values per vital	Pass
		Initialize with enemies enabled	Enemies should spawn at spawners upon game start	Enemies spawn at spawners	Pass
		Initialize with enemies disabled	Enemies should not spawn at spawners upon game start	Enemies do not spawn at all	Pass
		Initialize with crafting ingredients enabled	Player should be able to craft requiring ingredients	Player can craft as usual with ingredients	Pass
		Initialize with crafting ingredients disabled	Player should be able to craft anything regardless of inventory content. Inventory should remain unchanged	Item is crafted upon click regardless of inventory. No items removed upon crafting.	Pass
		Initialize with building ingredients enabled	Player should be able to build requiring ingredients	Player can build as usual with ingredients	Pass
		Initialize with building ingredients disabled	Player should be able to build anything regardless of inventory content. Inventory	Blueprint is spawned upon click regardless of inventory. No items removed upon building.	Pass

			should remain unchanged		
		Initialize with drop items on death enabled	Player should drop items upon death	Player drops all item on death	Pass
		Initialize with drop items on death disabled	Player should keep all items upon death	Player's inventory is unaffected on death	Pass
		Initialize with increased enemy health	Enemy health should be increased viewed through the inspector	Enemies' health is increased	Pass
		Initialize with increased enemy damage	Enemy should deal more damage than what is set	Enemy does more damage than what is set by default	Pass
		Initialize with increased enemy capacity per spawner	Each spawner should spawn increased number of enemies based on multiplier	Spawner exceed its default max capacity and stop at its new calculated capacity	Pass
Story	Storing story items	Interact with story items	Item should disappear and corresponding story item slot should show it	Item disappears and story slot icon displays the item	Pass
	Placing story items	Click on story item slot icons	Item slot should be cleared and appear at console	Correct item icons appear at console when story item slot is clicked	Pass
	Pressing button	Button pressed before placing all items	Nothing should happen and prompt is not shown	Prompt is not shown and no effect on interaction with button	Pass
		Button pressed after placing all items	Black hole appears and energy crystal should light up	Black hole appears and energy crystal lights up	Pass
	Ending scene	Player interacts with ending trigger	Player should be pushed towards black hole gradually and ending the game showing an end game screen	Player is pushed gradually to black hole and end game screen shows when reaching black hole	Pass

Saving System	Saving and loading	Player saves after starting new game with a selected difficulty	Player should load in with the same initialized values for the difficulty	Player and world properties are loaded with the same values for the difficulty	Pass
		Player saves at a random location	When loading, player should start off where they saved the game	Player starts where they saved when loading back into the game	Pass
		Player saves with different levels of vitals	Each vital level is restored to the values they were saved at	Player vital levels restored to the values saved at	Pass
		Player saves with a collection of items in their inventory	Player should start the game with those exact items in the exact slots they were saved at	All items are reassigned to their exact slots they were saved at	Pass
		Player saves after building some structures	Structures should be rebuilt at the same location after being rebuilt	Structures are restored at their exact locations	Pass
		Player builds and sets a new spawn point and saves	When loaded, after dying the player should respawn at that set respawn point	Player respawns at the new set spawn point after dying	Pass
		Players saves after dropping some items in front of them	After loading, the exact items saved should start where they were saved	All items are restored to where they were saved at	Pass
		Player uses a weapons consuming ammo and saves	The weapon should load back into the player's inventory with the exact ammo count it had	The weapon is loaded back in the player inventory with the saved ammo count	Pass
		Player uses a weapons consuming ammo then drops it and saves	The weapons should load back in the world where it was dropped with the exact ammo count it had	The weapon is loaded back in the world with the saved ammo count	Pass
		Player stores items in a storage and saves	The storage should be loaded with all its content including what the	All items are restored in that exact storage in	Pass

			player placed inside	the right slots where saved	
		Player picks up one story item and saves	That specific story item should appear in the inventory	Story item is present in the inventory	pass
		Player picks up one story item and places another story item in the console and saves	First item should be in the inventory and the second should be in the console	First item present in the inventory and second in the console	Pass
		Player picks all items up and places in the console but does not press the button and saves	All items should be in the console and the button available to press	Button is available to press, and all items present in the console	Pass
		Player picks all items up and place in console and presses button and saves	All items should be in console and button unavailable to press but black hole and energy crystal present	All items in console and black hole and crystal present	Pass
		Player ends the game completing all story objectives	Player should spawn at the initial spawn point. Console stays fixed, black hole and energy crystal remains present	Player spawns at initial spawn point. Console is fixed, black hole and energy crystal present	Pass
Hazards	Boundary hazard	Player exits the play area and enters asteroid zone	Asteroids spawn in intervals and launches towards player damaging them	Asteroid spawn and launched towards the player and damages them	Pass

8.1.2 Requirement Testing

The project requirements are tested to ensure a functional game with all its functioning proposed requirements. Below is a test table for both mandatory and optional functional requirements and non-functional requirements stated in the requirements analysis chapter.

The optional requirements 17, 20 and 24 are left blank as they were not implemented due to time constraints.

Req. No.	Requirement	Inputs	Expected Outputs	Actual Outputs	Pass /Fail
Mandatory Functional Requirements					
1	The game should have different difficulties to select from and play	Starting new games with all five difficulties	Each difficulty should provide varying rules and stats for the player and enemies	New game with each difficulty grants different player and enemy stats and rules apply such as building with or without ingredients	Pass
2	The game should have an inventory system	Player picks up and drops items	Picking up items stores them in the player inventory allowing the player to move them around or drop them	Items picked up are stores and displayed in inventory and can be moved around as well as drop them	Pass
3	The player should be able to die and lose their items or respawn with them	Player dies with items with rule set to lose items and without	Items should be dropped on death when rule is enabled and not dropped when rule is not set to drop items	Items drop on death with rule enabled and not without	Pass
4	The player should have vitals that need to be managed	Time passing, player consuming items, player in and outside of oxygen	Player should lose food and water levels over time, refill these with consumables and gain and lose oxygen when in and outside of it respectively	Player food and water levels decrement over time, refills correctly when consuming items and gains oxygen inside oxygen zones and loses oxygen outside of it	Pass
5	The game should have a save and load feature	Saving with new changes to the world	When loading, the world should be set to its new changes made when saving	New changes reapplied to the world upon load	Pass
6	The player should be able to craft items	Crafting items at a crafting station	Desired item should appear in the inventory removing the required items. The item must be usable as intended	Item is crafted, removing its required items and functions as intended, I.e., weapons shoot, consumables can be used	Pass
7	The player should be able to build structures in appropriate areas	Build structures, attempt to build at a landmark	Structures should be built anywhere besides near a landmark	Structures are built correctly, and nothing happens when trying to build near a landmark	Pass
8	The player should be able to harvest resources	Attack a harvestable object with a harvesting tool	Harvestable object should disappear, and another object should appear	Harvestable object disappears and another object replaces it.	Pass

9	The player should be able to attack enemies with several types of weapons	Player attack enemy with melee, ray, and projectile weapon	All weapons should work and damage the enemy	Each weapon deals their set damage to the enemy	Pass
10	The game should have at least one type of enemy that can harm the player	Enemy attacks player, Enemy attack player with another weapon	Enemy should be able to target the player when in range. Each weapon assigned to the enemy should deal damage to the player	Enemy targets player when in range and attack when close enough. Each weapon correctly damages the player upon hit	Pass
11	There should be at least one type of hazard that can harm the player	Player leaves the play area of the world	Asteroids should spawn and launch towards the player applying damage upon impact	Player takes damage by asteroids when leaving the play area. Asteroids stop spawning once back inside	Pass
12	The game should have puzzles that the player can complete for rewards or story line progression	Player completes puzzle, Player picks up story items	World should have a mini game where the player can complete to obtain story item.	Story item is rewarded upon completing puzzle. Items can be picked up.	Pass
13	The game should have a simple story that the player can follow and complete	Player picks up story items, Player uses story item at a specific location	Player can pick up and used story items at a location to progress in the story	Player picks up story items and stores them securely. Items can later be used at a location correctly progressing the story	Pass
Optional Functional Requirements					
14	The player should be able to experience various levels of gravity	Player moves with gravity and without gravity. Player transitions between the two	With gravity, the player moves parallel to the surface while jumping returning to the surface. Without gravity allows the player to move in any axis freely. Movement rules should correctly apply when in or outside of gravity	Player moves correctly with gravity while jumping successfully. Player moves well in any axis without gravity and transitions correctly when moving between gravity zones.	Pass
15	The game should have advanced weapons that	Player picks up and attack with melee, ray, and	Each weapon type should attack in different manner; Melee is instant hit with	Each weapon works as intended	Pass

	perform different effects	projectile weapon	a short range, ray is instant hit and projectile launches a projectile that explodes damaging object in its radius		
16	The game should have more than one type of enemy	Enemies with different stats and weapon	Enemies should be able to attack with any weapon assigned and stats should correctly apply such as movement speed	Each weapon works as intended and stats are correctly applied giving their appropriate effects	Pass
17	The game should have different means of navigation and transport	Not implemented	Not implemented	Not implemented	N.A
18	The player can upgrade their stats and tools	Player picks up and equips upgrades	Each upgrade should give their set effects	Upgrades provide their intended effects and effect is removed when unequipping upgrade	Pass
19	The game will have tamable or creatable pets that follow the player	Player interacts with pet	Pet should follow the player and player should be able to access its storage	Pet follows the player upon interaction and player can store items within the pet	Pass
20	The key binds should be able to be remapped	Not implemented	Not implemented	Not implemented	N.A
Mandatory Non-Functional Requirements					
21	The game should run at least above 30 frames per second	Play the game with high end pc. Play the game with work laptop	High end PC and laptop should reach above 30 frames per second.	High end PC exceeds 30 frames per second. Laptop runs at 20 frames per second with high fidelity graphics but achieves 30 when graphics are tones down	Pass
22	The game will run on Windows 10 operating systems	Play the game on a system running on Windows 10	Game should run normally on windows 10 platforms	Game runs normally without issues	Pass
Optional Non-Functional Requirements					

23	The game should reach 60 frames per second	Play the game with high end pc. Play the game with work laptop	High end PC and laptop should reach above 60 frames per second.	High end PC exceeds 60 frames per second, but laptop fails to reach this	Pass /Fail
24	The game's graphic settings can be adjusted	Not implemented	Not implemented	Not implemented	N.A

8.2 User Testing

The game is also tested with other people who have never seen the game before presenting new perspectives of the game which can be used to improve usability and features as well as identify any issues.

Individuals are requested to test the game which they may freely decline or accept. Those that accepted are sent a build of the game or sat down in person at a computer system with the game installed. Participants are informed that they may withdraw at any time and do not need to complete the game, the data that will be collected from them as they play and briefed on the stages of the test.

The participants are briefed and then directed to play the game. The first stage of testing focuses on the game's usability and participants are left alone to see how they navigate the menu, world, and features of the game as any player would when playing a new game. The second stage of testing focuses on the features of the game where the participants are then given simple tasks to see how easily they accomplish the task such as 'Craft an item.'

After testing the game, the participants are debriefed and offered the emails of both the researcher and their supervisor. Finally, they will then be asked to complete an online questionnaire where their responses are kept anonymous. Results from the questionnaire can be found in Appendix C. Both results from the questionnaire and observations made are used to make changes and improvements to the game shown below:

- Improved input feedback to interactions by adding outlines to interactable objects, icons to display the type of interactable they are and interaction sounds
- World space health UI display on damageable object that are displayed upon taking damage
- A short tutorial when starting a new game covering all basic features of the game
- Multiple quality of life shortcut keys:
    - A key to quickly drop the item in a slot the cursor is currently hovering over
    - A key to quickly move an item to another appropriate slot I.e., weapons to weapon slot or item to external storage
    - Added multiple ways to exit an interface
- Added helper text to show key bindings

- Improvement items pick up detection especially for smaller items
- Removes mining spawn chance probabilities for faster game play
- Reduced items required for recipes for faster game play
- Added a utility script for player world space canvas' to accommodate for different screen resolutions

9 Conclusion

A space-inspired open-world survival game was proposed and successfully created implementing, achieving all mandatory and most optional requirements and positive feedback from user testing seen in Appendix C with minor changes made to cater to the feedback. However, requirements 17, 20 and 24 were not implemented due to time constraints.

9.1 Implementation Results

The most challenging feature to implement was the building system as it was designed in a way to check if the structure was sealed, connected with others, or disconnected and accommodating to these changes. During the unit test, one test failed in some circumstances; The random structure successfully checks if the structure is sealed however, because of the way the building grid is designed, each grid unit stores its reference to a wall for each side (front, right, back, left) so when building walls, they can seem to be blocking or sealing in a building but since they're placed in a unit that isn't connected to the main system, they are not used to check if the system is sealed there resulting in some areas remaining unsealed. The building system can be redesigned to just store shared references of a wall at a point.

Although functional, the save system must be optimized when saving game data as it currently stores a minimum of 2 million characters. This makes encrypting the data an awfully expensive task even when using single clock cycle operations. Currently, the save system saves every pickable item, storage and respawner which can amount to a large number of things to save. A compromise to fix this issue is to just save the items and storage that the player creates and drops where this may cause items around the map to respawn upon reloading the world, but this yields less data to save. One other small detail to consider when designing the map or implementing is to convert each game object's position and rotation to integer values or minimize decimal places to shorten the overall data size.

A modular enemy system is implemented where their weapons can be switched to others, models can be changed and their stats such as health or speed can be modified for each instance. However, their actions are limited as they only roam and attack the player by charging straight at them. This is fine when around empty areas and the current design of the world and where they spawn however when objects are in the way, the enemies have no way of evading these objects and combat is simple as they stand still and shoot straight at the player. Additional actions during a state can be added such as moving aside before attacking or attacking at the position where the player is moving towards. Micael DaGraça talks about using visual along with audio awareness, using probability and possibility maps and more [47] which can be used to improve enemies in this game.

9.2 Gameplay Results

Based on the questionnaire in Appendix C, the game was deemed to be fun with some small issues in navigating the features and interface of the game. The game could be made easier to learn by introducing the primary features slowly as the player progresses since the player is just thrown right in without any indication of what to do or how the game works. Enemies should also be introduced earlier but at an easier level as most testers did not come across any.

The storyline and its objectives were implemented at the very end with little time left of the project, which resulted in a simple story. A much more comprehensive story can be rewritten and implemented to tie all the features together naturally along with giving the player tasks to play towards where sandbox aspects would still exist but are not the primary aspect of the game.

The map implemented came out as exactly planned from a sketch although the scaling of the map should be improved as the map is a bit large compared to the player which could lead to a lot of time just travelling around the map which could easily become boring if the player is not met with encounters.

Currently, the game is not very customizable where implementation focused more on the gameplay and features hence, optional requirement 20 was not achieved. This can be achieved by adding the ability to rebind keys to let the player feel more comfortable playing the game, enhancing immersion as they will not be spending time figuring out the keys. Additionally, adding a settings menu to change the volumes of the game and most importantly, mouse sensitivity as every player has varying hardware hence varying dots per inch (DPI) settings for their mouse which makes the optimal sensitivity ranging from 0.45-1.8°/mm [48], where the character rotates a given degree per millimetre moved by the mouse, harder to achieve by every player. Frequent video gamers generally also have a comfortable and optimal sensitivity they prefer to enjoy any game they play.

The game runs very well despite being an open-world game with various features and numerous objects and items existing in a single scene. Further implementing an option to switch graphic settings along with optimizing these graphic settings for the best balance between visuals and performance can achieve optional requirement 24.

10 Future Works

Although complete and functional, the game has a lot of room for improvement where many different features can be added, or existing ones be improved.

10.1 Improved Building System

Improvements and add-ons to the building system can be made; The sealed structure system can be used to detect and set parts of a system to function or not, controlling the operability of built machines within that room. For example, crafting table functions on power which is generated by other built structures such as solar panels, where power is cut when the room is not sealed. Or an oxygen volume system where each grid unit provides a maximum volume, and this volume needs to be filled with generated oxygen and the rate that the player regains oxygen depends on this oxygen per volume.

Another change that can be made is to make the structures entirely voxel-based to provide destructive abilities and optimization when drawing the structure's mesh. This can easily be done as a voxel generation system already exists where a custom voxel editor can be created to design voxel models where they can be placed in the world to be generated.

10.2 Better Movement

Another improvement suggested by a user tester is to make the movement acceleration based to simulate the zero drag in space where the player must control their acceleration and deceleration. Controls to rotate the player in all axes can also be added to give the player more freedom in their movement and contribute to making them feel like they are floating in space. These features should be varied with difficulty levels or a setting as it may not be desired but more to add a challenge to the game and to prevent the risk of motion sickness.

A method of faster or more secure transport should be added to allow players to move around the vast map more efficiently such as vehicles where these vehicles must be upkeeped in fuel and integrity. They may also contain storage to let the player carry additional items, making it a huge incentive to obtain. With the addition of the voxel system, a modular destructible vehicle system could also be implemented where the vehicle consists of different major components which can be damaged and destroyed rendering the vehicle inoperable or less efficient.

10.3 Better Enemies

As mentioned in the conclusion, enemies can be greatly improved by making them predict player behaviour and make smarter decisions such as avoiding obstacles.

A boss can also be added for the player to defeat and progress in the game, encouraging them to obtain stronger weapons and better upgrades to fight against it.

10.4 Game Design

The game art can be made a little more detailed by adding simple material textures to the models retaining its simple art style but giving models some distinction. Models created in Blender can be manually added using Blender and generated meshes such as voxels and marching cubes can be textured with methods described in section 6.8.5.

The map design can be revamped to fully optimize the performance benefits of occlusion culling by strategically hiding structures, objects, and parts of the map with other larger static objects.

Map design can also be improved to support a better story by creating a much more linear world where the player will progressively get deeper in along with the story rewarding the player with their progression with a richer environment however a much more challenging one. This draws a clearer line on where the player should go which can naturally give the player suggestions for tasks and where to go to progress. However, this method generally minimizes the free exploration aspects of the open-world game and the idea of space itself being a vast open area with points of interest the player can freely explore. A different method, keeping the open-world nature of the map, is to make landmarks more visible and alluring to the player or to provide enough content to make it interesting around the map so that there is no set, structured path but indications to help the player form their own [49].

References

Model References

- Lowpoly Robots (2021) by Satendra Saraswat. Available at: https://sketchfab.com/3d-models/lowpoly-robots-7181d9e5ba434aa59dc7b9042c9723df
- Cat Lamp Free 3D model (2022) by Pundus. Available at: https://www.cgtrader.com/free-3d-models/character/other/cat-lamp-44e19be5-09a1-4b98-988d-33e1bc109b2b

Audio Clip References

- OTIS MEOW (2017) by blimp66
  Available at: https://freesound.org/people/blimp66/sounds/397661/
- Robot (2017) by ScreamStudio
  Available at: https://freesound.org/people/ScreamStudio/sounds/397253/
- Space Swoosh - brighter (2014) by GameAudio
  Available at: https://freesound.org/people/GameAudio/sounds/220191/
- Large Anvil & Steel Hammer 4 (2010) by Benboncan
  Available at: https://freesound.org/people/Benboncan/sounds/103632/
- Bonfire Being Lit (2013) by samararaine
  Available at: https://freesound.org/people/samararaine/sounds/186374/
- Sci Fi Interface (2016) by Jofae
  Available at: https://freesound.org/people/Jofae/sounds/367997/
- Breathing (2017) by pointparkcinema
  Available at: https://freesound.org/people/pointparkcinema/sounds/407238/
- Water_Drink (2008) by Q.K.
  Available at: https://freesound.org/people/Q.K./sounds/56270/
- Eating Crisps (2016) by Sethroph
  Available at: https://freesound.org/people/Sethroph/sounds/334209/
- sword7 (2007) by Streety
  Available at: https://freesound.org/people/Streety/sounds/30248/
- 1911 Reload (2017) by nioczkus
  Available at: https://freesound.org/people/nioczkus/sounds/396331/
- tone beep (2015) by pan14
  Available at: https://freesound.org/people/pan14/sounds/263133/
- nail gun (2016) by natemarler
  Available at: https://freesound.org/people/natemarler/sounds/338911/
- Laser (2018) by harrietniamh
  Available at: https://freesound.org/people/harrietniamh/sounds/415082/

- Charged laser (2012) by LegoLunatic
  Available at: https://freesound.org/people/LegoLunatic/sounds/151243/
- Energy Whip 2 (2008) by ejfortin
  Available at: https://freesound.org/people/ejfortin/sounds/49695/
- JM_NOIZ_Laser 01 (2015) by Julien Matthey
  Available at: https://freesound.org/people/Julien%20Matthey/sounds/268344/
- Sword Swing B - Medium 07 (2019) by Glaneur de sons
  Available at: https://freesound.org/people/Glaneur%20de%20sons/sounds/420619/
- Laser Gun Recharge (2013) by Dpoggioli
  Available at: https://freesound.org/people/Dpoggioli/sounds/196907/
- laser (2007) by THE_bizniss
  Available at: https://freesound.org/people/THE_bizniss/sounds/39459/
- Whoosh (2008) by qubodup
  Available at: https://freesound.org/people/qubodup/sounds/60013/
- Swinging staff whoosh (strong) 10 (2018) by Nightflame
  Available at: https://freesound.org/people/Nightflame/sounds/422512/
- metal_rings_04 (2010) by Department64
  Available at: https://freesound.org/people/Department64/sounds/95275/
- wood hit (2020) by Ian_G
  Available at: https://freesound.org/people/Ian_G/sounds/547414/
- Rock Smash (2020) by NeoSpica
  Available at: https://freesound.org/people/NeoSpica/sounds/512243/
- Pickaxe Mining Clank OWI (2021) by WolfOWI
  Available at: https://freesound.org/people/WolfOWI/sounds/588306/
- UI Click (2020) by EminYILDIRIM
  Available at: https://freesound.org/people/EminYILDIRIM/sounds/536108/
- UI Completed Status Alert Notification SFX001 (2015) by Headphaze
  Available at: https://freesound.org/people/Headphaze/sounds/277033/
- Click (2014) by complex_waveform
  Available at: https://freesound.org/people/complex_waveform/sounds/213148/

Asset References

- Easy-Icon-Maker (2020) by Servalstar. Available at: https://github.com/Servalstar/Easy-Icon-Maker
- Black Hole Shader with Unity Shader Graph (2020) by Mert Kirimgeri. Available at: https://www.youtube.com/watch?v=SWJZcQvTmnw
- Marching cube triangles configuration table (1994) by Paul Bourke. Available at: http://paulbourke.net/geometry/polygonise/
- Saving and Loading system (2022) by Trever Mock. Available at: How to make a Save & Load System in Unity | 2022

- PerlinNoiseMaker (2017) *by blackears* Available at: https://github.com/blackears/PerlinNoiseMaker

Report References

[1] Unity Technologies. *Unity 3d*. https://unity.com/

[2] Halpern, J. (2019) *The what and why of Game Engines*, *Medium*. Medium. Available at: https://medium.com/@jaredehalpern/the-what-and-why-of-game-engines-f2b89a46d01f#:~:text=Game%20engines%20provide%20tremendous%20efficiency,entirely%20on%20writing%20gameplay%20code. (Accessed: November 1, 2022).

[3] Derk (2016) *How to write a good game story*, *Paladin Studios*. Available at: https://paladinstudios.com/2012/08/06/how-to-write-a-good-game-story-and-get-filthy-rich/ (Accessed: October 24, 2022).

[4] Wikipedia (2022) *Open World*. Available at: https://en.wikipedia.org/wiki/Open_world (Accessed: October 24, 2022).

[5] Philippa Warr (2018). *SUBNAUTICA REVIEW* [Review of the game *Subnautica*, by Unknown Worlds Entertainment]. PCGamer. https://www.pcgamer.com/subnautica-review/

[6] Leana Hafer (2018). *Subnautica Review* [Review of the game *Subnautica*, by Unknown Worlds Entertainment]. IGN. https://www.ign.com/articles/2018/01/26/subnautica-review

[7] Community. *Astroneer*, *Astroneer Wiki*. Available at: https://astroneer.fandom.com/wiki/Astroneer#:~:text=Astroneer%20is%20a%20space%2Dthemed,unearthing%20rare%20treasures%20and%20discoveries (Accessed: October 25, 2022).

[8] Nicola Ardron (2019). *Astroneer Review* [Review of the game *Astroneer*, by System Era Softworks]. God is a Geek. https://www.godisageek.com/reviews/astroneer-review/

[9] Leana Hafer (2018). *The Forest Review* [Review of the game *The Forest,* by Endnight Games]. IGN. https://www.ign.com/articles/2018/05/10/the-forest-review-2

[10] Pontypants (2019) *How to make low poly look good*, *Sunday Sundae*. Available at: https://sundaysundae.co/how-to-make-low-poly-look-good/ (Accessed: October 26, 2022).

[11] Bernardo Bittencourt. (2019) *Game design: Choosing a Theme*, *Ambal Tournament.* Available at: https://ambaltournament.com/game-design-choosing-a-theme/ (Accessed: October 26, 2022).

[12] Epic Games. *Unreal Engine*. https://www.unrealengine.com/

[13] Asaf Eldad (2022) *Unity vs Unreal - which Game dev are you?*, *Incredibuild*. Available at: https://www.incredibuild.com/blog/unity-vs-unreal-what-kind-of-game-dev-are-you (Accessed: October 26, 2022).

[14] Blender Foundation. *Blender*. https://www.blender.org/

[15] MAXON Computer. *Cinema 4D*. https://www.maxon.net/en/cinema-4d

[16] Image-Line. *FL Studio*. https://www.image-line.com/

[17] Ableton. *Ableton Live*. https://www.ableton.com/en/

[18] Josh Bycer (2019) *The importance of background music and sound effects in video games*, *Game Wisdom*. Available at: https://game-wisdom.com/guest/importance-background-music-sound-effects-video-games (Accessed: October 27, 2022).

[19] Jennifer Larson (2020) *How Many Frames Per Second Can the Human Eye See?* Available at: https://www.healthline.com/health/human-eye-fps (Accessed: October 29, 2022).

[20] Ivica Crnkovic and Magnus Larsson (2001) *Challenges of component-based development.* Available at: https://www.cin.ufpe.br/~in1045/papers/art09.pdf (Accessed: March 15, 2023).

[21] Microsoft (2022) *Delegates (C# Programming Guide)* Available at: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/ (Accessed: March 15, 2023).

[22] Microsoft (2021) *Distinguishing Delegates and Events.* Available at: https://learn.microsoft.com/en-us/dotnet/csharp/distinguish-delegates-events (Accessed: March 15, 2023).

[23] Unity Technology (2023) *Garbage collection best practices.* Available at: https://docs.unity3d.com/Manual/performance-garbage-collection-best-practices.html (Accessed: March 15, 2023).

[24] Unity Technology (2018) *ScriptableObject.* Available at: https://docs.unity3d.com/Manual/class-ScriptableObject.html (Accessed: March 15, 2023).

[25] Unity Technology (2020) *Custom Pass*. Available at: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@10.2/manual/Custom-Pass.html#:~:text=HDRP%20Custom%20Passes%20allow%20you,like%20depth%2C%20color%20or%20normal (Accessed: March 16, 2023).

[26] Unity Technology (2023) *ShaderLab command: ZTest*. Available at: https://docs.unity3d.com/Manual/SL-ZTest.html (Accessed: March 22, 2023).

[27] Unity Technology (2023) *Animation System Overview.* Available at: https://docs.unity3d.com/Manual/AnimationOverview.html (Accessed: March 22, 2023)

[28] Mega Voxels (2023) *What is a voxel?* Available at: https://www.megavoxels.com/learn/what-is-a-voxel/ (Accessed March 23, 2023)

[29] Atomontage Inc. (2018) *Atomontage Volumetric Technology Reel – April 2018* [Video]. Available at: https://www.youtube.com/watch?v=nr5JqYYye3w (Accessed March 23, 2023)

[30] Jennifer L. Whitwell (2009) *Voxel-Based Morphometry: An Automated Technique for Assessing Structural Changes in the Brain.* Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6666603/ (Accessed March 23, 2023)

[31] Carlpilot (2017) *[Unity] Easy 3D Perlin Noise* [Video]. Available at: https://www.youtube.com/watch?v=Aga0TBJkchM (Accessed March 23, 2023)

[32] Sebastian Lague (2016) *Procedural Landmass Generation (E01: Introduction)* [Video]. Available at: Procedural Landmass Generation (E01: Introduction) (Accessed March 23, 2023)

[33] Unity Technologies (2023) *Mesh.indexFormat.* Available at: https://docs.unity3d.com/ScriptReference/Mesh-indexFormat.html (Accessed March 23, 2023)

[34] A.H. YURTTAKAL and H. ERBAY and T. İKİZCELİ and S. KARAÇAVUŞ and G. ÇINARER (2018) *3D Visualization Thyroid CT Images Using Marching Cubes Algorithm.* Available at: http://indexive.com/uploads/papers/icatces2018-63.pdf (Accessed March 23, 2023)

[35] Brent Owens (2014) *Use Tri-Planar Texture Mapping for Better Terrain.* Available at: https://gamedevelopment.tutsplus.com/articles/use-tri-planar-texture-mapping-for-better-terrain--gamedev-13821 (Accessed March 23, 2023)

[36] Unity Technology (2021) *JSON Serialization.* Available at: https://docs.unity3d.com/2020.1/Documentation/Manual/JSONSerialization.html#:~:text=Supported%20types,with%20the%20%5BSerializable%5D%20attribute (Accessed March 24, 2023)

[37] Microsoft (2023) *Deserialization risks in use of BinaryFormatter and related types.* Available at: https://learn.microsoft.com/en-us/dotnet/standard/serialization/binaryformatter-security-guide (Accessed March 24, 2023)

[38] Alan Thorn (2015) *Unity Animation Essentials* [Book] Available at: https://subscription.packtpub.com/book/business-&-other/9781782174813/7/ch07lvl1sec41/inverse-kinematics (Accessed March 25, 2023)

[39] 25games (2020) *How to create Level of Detail (LOD) in Blender 2.8 for Unity – Tutorial – preperation & export* [Video] Available at: https://www.youtube.com/watch?v=KRRPreI6SNM (Accessed March 25, 2023)

[40] Code Monkey (2021) *I made an RPG only with Visual Scripting! (NO CODE)* [Video]. Available at: https://www.youtube.com/watch?v=BkzNnKBHMPg (Accessed March 25, 2023)

[41] Dorian Iten (2019) *Understanding the Fresnel Effect.* Available at: https://www.dorian-iten.com/fresnel/ (Accessed March 25, 2023)

[42] Alexander Ameye (2021) *5 ways to draw an outline*. Available at: https://alexanderameye.github.io/notes/rendering-outlines/ (Accessed March 26, 2023)

[43] Ben Golus (2020) *The Quest for Very Wide Outlines.* Available at: https://bgolus.medium.com/the-quest-for-very-wide-outlines-ba82ed442cd9 (Accessed March 26, 2023)

[44] The Audactity Team. *Audacity*. https://www.audacityteam.org/

[45] Unity Technology (2023) *Gizmos.* Available at: https://docs.unity3d.com/ScriptReference/Gizmos.html (Accessed March 26, 2023)

[46] Unity Technology (2023) *Occlusion culling*. Available at: https://docs.unity3d.com/Manual/OcclusionCulling.html (Accessed March 26, 2023)

[47] Micael DaGraça (2017) *Practical Game AI Programming* [Book]. Available at: https://subscription.packtpub.com/book/game-development/9781787122819/pref01 (Accessed March 26, 2023)

[48] BEN BOUDAOUD, JOSEF SPJUT, and JOOHWAN KIM (2022) *Mouse Sensitivity Effects in First-Person Targeting Tasks.* Available at: https://arxiv.org/pdf/2203.12050.pdf (Accessed March 26, 2023)

[49] Jennifer Mendez (2016) *Game Design: Open World Vs. Linear.* Available at: https://blackshellmedia.com/2016/05/15/game-design-open-world-vs-linear/ (Accessed March 26, 2023)

Appendices

Appendix A - Project Log

Term 1 – Week 1

- Briefed over the aspects of the game: goals, storyline, challenges
- Created draft Gantt chart for project tasks
- Looked at previous personal projects to revise over different feature implementations and world design

Term 1 – Week 2

- Completed first draft of project proposal
- Developed each feature of the game; what the feature is and how to implement and several types of each, for example, challenges could include enemies to defeat or puzzles to work out or different weapons and what they will do

Term 1 – Week 3

- Finalize project proposal and present
- Decided art style of game; art style will be low poly
- Look for models and art to use and list which to possibly self-create; art such as background for the world or textures for the game objects
- Study the unity shader graph and visual effect graph to create effects for certain items and the world design

Term 1 – Week 4

- Started interim report, included details from meeting
- Created first implementation of a simple character movement system where the player can move in the x, y, and z axis, simulating a manned maneuvering unit (MMU) in space
- Researched similar games and taken ideas as inspiration for gameplay and design
- Researched the fundamentals of space and took notes to apply it into game logic

Term 1 – Week 5

- Sent interim report to supervisor for review
- Created a new colour palette for the new game and remapped UVs for old models to it
- Refined models to suit the game's advanced low poly theme

Term 1 – Week 6

- Reviewed feedback for draft report and made changes and sent to supervisor

- Added advanced movement; switching between walking and floating

Term 1 – Week 7

- Reviewed feedback and finalized interim report for submission

Term 1 – Week 8

Term 1 – Week 9

Term 1 – Week 10

- Added inventory system to store and carry items
- Added types of items such as shield and upgrade to store in special slots
- Created dynamic object pooler to pool objects on picking up items and reenabling them on drop
- Created more item models for crafting and general resources and consumables

Term 1 – Week 11

- Added player vital system with health, water, hunger, oxygen, and shields
- Added damageable objects that drop items on death and different types of weapons to damage objects; Melee, ray, and projectile weapon
- Added oxygen detection and gravity detection with colliders

Christmas Break

- Added types of crafting tables and different sets of craftable items
- Added building system with a grid allowing sealed systems
- Update oxygen detection within sealed buildings
- Added more models for buildings and furniture

Term 2 – Week 1

- Added voxels with random generation
- Added marching cubes with random generation
- Made voxels and marching cubes harvestable

Term 2 – Week 2

- Added storage and buildable storage
- Changed most UI to world space and added to storage

Term 2 – Week 3

- Added enemies that attack player if in range
- Added spawners for enemies
- Added respawn system and able to set different spawn points

- Added ability to drop items upon death

Term 2 – Week 4

- Replaced enemy models and added animations
- Added ability to harvest enemies after death
- Started level development adding asteroids to set the environment
- Created space background

Term 2 – Week 5

- Added landmarks and structure around the map with items to loot
- Added harvestable objects, voxels, and marching cubes
- Added a pet that follows you and stores items
- Created world boundary with hazards that shoot asteroid at player if too far off

Term 2 – Week 6

- Finished off developing map
- Added short story with items around the map to collect
- Added a small survival styled mini game

Term 2 – Week 7

- Created VFX for tools, weapons, and others

Term 2 – Week 8

- Added sound effects for tools, weapons, and others
- Created main menu and switch between the two
- Added background menu music

Term 2 – Week 9

- Added saving and loading game
- Added post processing
- Added occlusion culling

Term 2 – Week 10

- Testing and review

Term 2 – Week 11

- Added quality of life shortcut keys to inventory
- Added short tutorial for basic and essential feature of the game
- Updated recipes for shorter gameplay and less farming
- Updated harvest increased chances for harvesting

- Updated UI to increase icon visibility
- Added world space UI to display health of damageable objects

Easter Holiday and beyond

- Worked on final report

Appendix B - Supervisor Log

Microsoft Teams Meeting – 05/10/2022 10:30-11:00

- Brief discussion about the project, what it is, what it will be about, how and plans for implementation
- Discussion about ensuring to being able to implement planned things, not to be over ambitious
- Plan to meet in person to show ongoing progress of the project in the future
- Start working on project proposal; what game engine to use for project and benefits over the other, other software for other multimedia such as models and music ETC, research similar projects and the issues behind them

Microsoft Teams Meeting – 18/10/2022 12:00-12:30

- Start working on prototyping the game as well as the interim report
- Discussed about what should be in interim report
    - intro - 3d environment, type of game, functionality, component related to game - age rating, Pegi rating, gameplay risks such as epilepsy, code of conduct (use other assets - need to reference)
    - professional ethical consideration - can withdraw from usability testing (sign form and send to supervisor)
    - background research - look at related games (2-3), what happens that is interesting (good? issues?), where do ideas come from, how to implement (unity? unreal? any other software for other uses)
    - requirements analysis - list of requirements, add detail to requirements, can change if anything goes wrong
    - project plan (Gantt chart for next 6 month) - key milestones, interim report, final report, prototype of usability testing, overview of general structure
- Send draft interim report before next meeting
- Complexity – can do own stuff, show one thing so it demonstrates skills (modelling, VFX, etc.)

Microsoft Teams Meeting – 1/11/2022 12:00-12:30

- Talked over changes and improvements for report
    - Intro should be more general rather than talk about the specifics of the game
    - Add reviews about the researched games, talk about any issues with them
    - Profession consideration: look at BSc code and talk about points that will be met, E.G. PEGI rating, epilepsy warning, respect for third party assets/code
    - Ethical considerations: plan useability study; show the game, let them play, take feedback. Or use questionnaire. Respect privacy, if they want to stop, they may.

- Plan to send another draft before the end of the week
- Set another meeting in 2 weeks

Microsoft Teams Meeting – 17/11/2022 12:00-12:30

- Discussed interim report feedback
- Planned next development plans

Microsoft Teams Meeting – 08/12/2022 11:00-13:30

- Showcase update on development and catchup
- Planned next development plans

Microsoft Teams Meeting – 27/01/2023 10:30-11:00

- Showcase update on development and catchup
- Planned next development plans

Microsoft Teams Meeting – 09/02/2023 11:00-11:30

- Showcase update on development and catchup
- Planned next development plans

Microsoft Teams Meeting – 09/03/2023 11:00-11:30

- Showcase update on development and catchup
- Planned next development plans

Chichester 1-122 – 23/03/2023 11:00-11:30

- Showcased current game progress in person and catchup

Microsoft Teams Meeting – 19/04/2023 14:30-15:00

- Showcased final build of game and catchup
- Talked about final report
- Talked about plans after final report deadline to talk about presentation

Appendix C – Questionnaire Results

Q1 Customise

How often do you play video games?

Answer Choices	Responses		
Never		16.67%	1
Rarely		16.67%	1
Sometimes		33.33%	2
Usually		33.33%	2
Always		0.00%	0
Answered: 6 Skipped: 0	Response Total:		6

Q2 Customise

Which difficulty did you play on?

Answer Choices	Responses		
Relaxed		16.67%	1
Peaceful		50.00%	3
Easy		0.00%	0
Intermediate		33.33%	2
Challenge		16.67%	1
Answered: 6 Skipped: 0	Response Total:		6

Q3

Customise

How long did you play for?

Answer Choices	Responses		
0-10mins		0.00%	0
11-20mins		33.33%	2
21-30mins		16.67%	1
31-60mins		16.67%	1
> 60mins		33.33%	2
Answered: 6 Skipped: 0	Response Total:		6

Q4

Customise

How easy or difficult was the game to learn?

Answer Choices	Responses		
Very easy		0.00%	0
Easy		16.67%	1
Neither easy nor difficult		50.00%	3
Difficult		33.33%	2
Very difficult		0.00%	0
Why: 5			▸
Answered: 6 Skipped: 0	Response Total:		6

5	I didn't know what to do at the start A tutorial would be nice and an explanation for the controls
4	Was hard to figure out what buttons did what but after i found out the game was easy to play
3	Not very clear on what to do
2	Knew basic game controls and inferred other possible interactions. although some controls where not easily identifiable e.g. was using the escape key to exit inventory UI
1	It was a bit harder on Intermediate mode, but it was well balanced.

Q5 Customise

How fun or boring was the game?

Answer Choices	Responses		
Very boring		0.00%	0
Boring		0.00%	0
Slightly boring		0.00%	0
Neither		16.67%	1
Slightly fun		33.33%	2
Fun		33.33%	2
Very Fun		16.67%	1
Why: 5			
Answered: 6 Skipped: 0	Response Total:		6

5	The world was fun to look at but the game wasn't clear
4	Map was fun to explore but it took some time to get there and too much mining and crafting
3	Game looked nice and it was fun to explore the map
2	It was fun, but the grind was too much. It wouldn't be too bad if picking up items was easier and movement was more fluid and no item RNG.
1	It was so satisfying to farm the Materials and fun to explore the World (the Added Cat too).

Q6 Customise

How immersive was your experience

Answer Choices	Responses		
Not immersive at all		0.00%	0
Not so immersive		16.67%	1
Somewhat immersive		66.67%	4
Very immersive		16.67%	1
Extremely immersive		0.00%	0
What could have made it better: 4			
Answered: 6 Skipped: 0	Response Total:		6

4	Again the map was cool but i felt like i was just floating in space!
3	Maybe some background sound or ambience so it feels more spacey
2	More things to explore within short distances. Some things feel to far to get to or too long to get there
1	VFX on mining could be better. Was confusing the vfx on rocks dissipating rather with the actual item that could be picked up, and was attempting to pick up/interact with the VFX rather than item.

Q7 Customise

How well did the game run?

Answer Choices	Responses		
Unplayable		0.00%	0
Bad		0.00%	0
Sufficient		0.00%	0
Well		0.00%	0
Very well		100.00%	6
Answered: 6 Skipped: 0	Response Total:		6

Q8 Customise

Did you come across any bugs?

Responses: Hide — Text Analysis

5	I dont think so	20/04/2023 15:21 PM ID: 215477037
4	No	20/04/2023 00:27 AM ID: 215429705
3	No	20/04/2023 00:12 AM ID: 215429519
2	No bugs	19/04/2023 22:18 PM ID: 215426881
1	No	19/04/2023 19:53 PM ID: 215420328

Answered: 5 Skipped: 1 Response Total: 5

Q9 Customise

How easy was navigating the UI? (such as inventory, crafting, main menu, etc)

Answer Choices	Responses	
Very easy	33.33%	2
Easy	16.67%	1
Neither easy nor difficult	16.67%	1
Difficult	33.33%	2
Very difficult	0.00%	0
Any improvements: 4		
Answered: 6 Skipped: 0	Response Total:	6

4	Maybe add some explanation on how each is opened and used
3	Eventually got used to it but would be nice to rebind some keys
2	It was abit confusing toggling some UI as different buttons lead to turning off the screen
1	Needs helper text, UI organisation, more contrasting colours, Items are hard to distinguish. Can't read what items are needed for recipes.

Q10 Customise

How balanced do you think the enemies were for your difficulty? (if any)

Answer Choices	Responses		
Too hard		0.00%	0
Hard		0.00%	0
Neutral		20.00%	1
Easy		20.00%	1
Too easy		0.00%	0
Other (please specify): Hide		60.00%	3
3 Didn't see any			20/04/2023 15:21 PM ID: 215477037
2 I didn't come across any			20/04/2023 00:12 AM ID: 215429519
1 Didn't progress far enough to encounter enemies			19/04/2023 22:18 PM ID: 215426881
Any improvements: 1			
Answered: 5 Skipped: 1	Response Total:		5

1 Maybe make them predict their shots as your move?

Q11 Customise

Would you want to continue playing this game?

Answer Choices	Responses		
Yes		66.67%	4
No		0.00%	0
Maybe		16.67%	1
Other (please specify): Show		16.67%	1
Answered: 6 Skipped: 0	Response Total:		6

1	Yes, after improvements

Q12

What would you want added to the game? (if any)You may suggest one or multiple

Responses: Hide —

6	Instructions!!
5	More buildings to build or customizable buildings like colours, a dog pet too
4	More things to explore and faster ways to travel
3	Robot cat to be customisable
2	Make items stackable, adding mouse sensitivity controls, add more realistic space movement i.e. acceleration and decceleration. Make picking up resources easier.
1	More Furniture to build more, More Map places to explore more!

Appendix D - Project Proposal

Candidate Number: 230956

Supervisor Name: Dr Paul Newbury

Working Title: 3D Unity Game – Lost in Space

Aims and Objectives

The project will be a 3D Unity game based in an open world environment where the player may freely explore and interact in. The player would start with almost nothing and may gather materials and use them to craft items to aid them or build structures such as a base. Some materials are unobtainable unless the player has a specific tool or grade of tool which will encourage the player to collect these materials. The main focus of the player is to find help or a way to get back home (earth or mothership) and there will be objectives and clues for the player to achieve and figure out how to get there. There will be puzzles around the map that the player can complete for unique items to use which may help the player or quest items required to complete the story. The game will have different difficulties where the player may pick which they prefer where the difficulty will change properties of the enemies and player.

The game will use assets which will mainly be self-made, and some royalty free/ The game will run on the Unity game engine and will be targeted towards Windows operating systems and will be delivered through a zipped file with a Unity executable file will open and run the game.

Primary Objectives:

Objective	Description
The game will be in 3D	The player will be able to navigation and look around in a 3-dimension space
The game will run on Windows 10	-
The map will be partially procedurally generated	The map will procedurally generate resources and non-primary landmarks such as puzzles or enemy spawns
Inventory system will store items and attire	The system will hold what the player collects throughout the game

Implement death system	The player should be able to die and respawn. The death system will return the players inventory upon death depending on the difficulty of the world otherwise leaving it in the position they died at
Implement saving system	The game should be able to be saved and loaded successfully
Crafting system	The player should be able to craft different items using materials
Building system	The player should be able to construct, place, move and destroy structures
Different gravity	The game will have various levels of gravity depending on where the player is
Basic tools and weapons	The game should have tools that the player can use to perform basic tasks such as harvesting or attacking
Enemies	The game should have at least one type of enemy that will perform basic actions such as attack and evade
Hazards	The world will have various hazards where the player should avoid otherwise the player dies or takes damage
Difficulties	The game will set the player's and enemies' properties depending on the difficulty selected
Puzzles	The game will include interactable object that will simulate a puzzle where the player may solve for rewards

Extension Objectives:

Objective	**Description**
Story elements	The game may have a complete story where the player makes it home and the game ends and the player can continue freely exploring their world.
Advanced items and weapons	Various tools and weapons the player can use with different effects
Advanced enemies	Types of attack, ranged and melee, Boss?
Types of navigation	Vehicles or upgrades for player movements
Player upgrades	Upgrade station to upgrade the players suit/character to perform actions or increase stats
Pets	The player may tame 'pets' or create their own. Pets will follow the player and may perform certain activities and act as an inventory
Remapping keys and inputs	Change the keyboard key binds and sensitivity of the mouse
Change graphic settings	Able to change the graphics settings of the game

Relevance

Unity uses the object-oriented programming language 'C#' which is like the language 'Java' which I have been taught throughout my course. I can also plan and manage this project using my knowledge from my software engineering module and ensure at least a complete working project is delivered. My programming for 3D also covers and develops my understanding of the Unity game engine and 3D environments for my game as I will need to create a world.

Resources required

- Personal Computer
- Unity Game Engine (standard or students edition)
- Blender Modelling Software
- Ableton Live 10 (free 90-day trial for students)
-

Timetable

08:00					
08:30					
09:00	*Human-Computer Interaction* (Lecture 1) Arts A A02	Project Work	Project Work	Project Work	Project Work
09:30					
10:00					
10:30					
11:00					
11:30					
12:00					
12:30					
13:00					
13:30					
14:00				*Visual Effects* (Laboratory 1) Chichester 1 CI204/5	
14:30					
15:00	*Programming for 3D* (Laboratory 1) Chichester 1 CHI 014				*Programming for 3D* (Laboratory 1) Chichester 1 CHI 014
15:30					
16:00					
16:30					
17:00					
17:30					
18:00					
18:30					
19:00				*Human-Computer Interaction* (Seminar 1) Arts A A02	
19:30					
20:00					
20:30					
21:00					
21:30					

Appendix E – Similar Projects Research

Subnautica

Developers: Unknown Worlds Entertainment, Panic Button Games, Grip Digital, Shiny Shoe LLC

Publishers: Unknown Worlds Entertainment, Gearbox Software

Website: https://unknownworlds.com/subnautica/

Date: Jan 23, 2018

Subnautica is an underwater-based, story-driven survival where a spaceship has crashed onto a planet and the player tries to survive the aftermath and is able to explore the depths of the sea and debris from the spaceship and collect materials to build a habitat as shown in Figure E.1. As time passes, the player starts to get clues by receiving radio messages and location of the broadcast where the player is led to explore towards uncovering the story and the mysterious planet.

Figure E.1: A screenshot of Subnautica from the official Subnautica Steam page

By using story-driven gameplay, Subnautica leads the player to certain areas of the map for them to progress through the story and to even introduce new items that they can craft and use or build. Directing the player to points of interest gives them something to do without entirely relying on their own freedom to explore but still allowing them to explore. The objectives are not time-based, allowing them to explore the points of interest when they feel like it or when they have prepared for a long expedition.

The game includes some obstacles for the player to work around or fight on such as different levels of enemies depending on the danger level of the area, and mini puzzles/timed challenges the player must complete for the story to progress. These obstacles can give the player a reason to be wary when exploring and purpose for weapons and vehicles which adds some challenge to the game rather than it being just a free-roaming exploration game.

Subnautica uses a lighter art style rather than a hyper realistic one which helps performance-wise as the game is open-world and helps smaller developers with development times in terms of asset creation.

The world is not procedurally generated but rather pre-made which different regions and randomly generated resource locations, as shown in Figure E.2, due to the fixed locations of the story objectives and points of interest. Creating a non-procedurally generated world can be time-consuming where planning is required to decide where to place everything but helps Subnautica to guide the player in every area of the game.

Figure E.2: A map of the Subnautica world from

Philippa Warr from PC Gamer reviews Subnautica with a score of 89/100 saying, “A smattering of technical issues keep Subnautica from true legendary status, but only just” [5]. To create a completely immersive experience, the game should not have any bugs or should be minimized as much as possible. Leana Hafer from IGN gave a 9.1/10 score and says that “Subnautica is a template for what open-world survival games should strive to be. It is fantastical, fresh, and frightening from surface to seabed, with a story that kept on surprising me and a cast of sea monsters that quite literally haunted my dreams” [6]. Following this review, a game’s environment should be well designed to truly immerse the player into the world including the world design, unforgettable enemies, and a good story.

Subnautica allows different difficulty modes; Survival, freedom, hardcore and creative mode where freedom mode excludes the requirement of sustenance however the story mode is still available, hardcore mode where the player only has one life and if they die, they lose all their progress and cannot respawn, and creative mode where the player may freely explore and build and cannot die.

Astroneer

Developers: System Era Softworks

Publishers: System Era Softworks, Gearbox Publishing

Website: https://astroneer.space/

Date: Dec 16, 2019

Astroneer focuses more on the exploration and sandbox side of games; The player starts by landing on an unknown planet where the player is tasked with exploring alien worlds, colonizing the worlds, and unearthing rare treasures and discoveries and must scavenge for resources and explore and may construct buildings to help reach new horizons where new planets and moons can be reached and explored.

Each of the planets and moons which the player can visit are procedurally generated where every planet, except one, shares the same universal resources but each have their unique primary and secondary rare resources. Figure E.3 shows an example of one of the procedurally generated planets.

Figure E.3: Screenshot of Astroneer from the official Astroneer Steam page

The storyline of the game is freer where the story depends more on the player on what they want to do and go where the only restrictions would be progression from new technology that the player can research.

The game has a low poly but appealing art style with saturated colour themes which gives the planets a more random, alien-like theme. The game also “started out as a purely cosmetic art” until later it was “transformed into a full game project” [7]. The developers also talk about how choosing such an art style “means [their] content overhead is generally much lower than that of projects that have more realistic and detailed art directions” [7] and due to them being a small team, it seems like it would allow them a lot of space to improve the visuals without it even needing it as a low poly art style is already a low detail style.

The player also has the ability to terraform the world using a tool which grants the ability to build and harvest resources through one tool, which can also have modifiers changing the ways it is used, overall giving the player large sandbox capabilities.

As the game focuses more on the exploration aspect in space, there is not much of a challenge to survive where the player only needs to manage their health by avoiding hazards in the game, unlike a typical survival game where the player requires sustenance.

A review on Astroneer written by Nicola Ardon from God is a Geek, giving a rating of 6.5/10, says that "Astroneer is a lovely serene survival crafting game that can feel aimless at times, but has masses of potential to be something special" [8]. As Astroneer has no story and is solely based on exploration, some players may have a problem finding something to do in the game if it is not apparent in the game such as a story to follow, objectives to complete or even small tutorials to introduce a feature to the game like crafting.

The Forest

Developers: Endnight Games

Publishers: Endnight Games, Alfred CAN

Website: https://endnightgames.com/games/the-forest

Date: Apr 30, 2018

The Forest is a horror-based survival game based on an inhabited island. In The Forest, the player starts in a plane that crashes onto an island where the player wakes up to see their son be taken away by a strange humanoid. This starts the player's motive to survive and to find their son. However, they are met with some inhabitants of the island and must fend off with what they can get their hands on or craft.

The game's art style takes a more hyper-realistic approach where the world looks very detailed as shown in Figure E.4 including shadows, foliage, screen space reflections in the water and highly quality textures.

Figure E.4: A screenshot of The Forest on the official game Steam page

The game uses cutscenes, such as at the start, to narrate the story of the game. The story in this game is a lot more hidden where locations and points of interest are not so obvious to the player and requires them to explore and find things as there are no waypoints or markers on their graphical interface telling them where to go. The player may obtain a map that may help navigate around the map, but nothing is directed to the player which amplifies the horror experience in this survival game. The Forest manages to tell the whole story solely from pictures and notes that the player finds and the environment in specific areas. Until the ending scene, the player gets to watch a short cutscene where minimal talking is present and shows what is happening through animations and pictures again.

The player's main obstacle is overcoming the constant horde of cannibals on the island that are curious and increasingly aggressive towards them where each day they become stronger which may endorse the player to build and fortify their home as well as find stronger weapons and means of fighting them off.

The island is also not procedurally generated, possibly due to storyline purposes as to get the player to explore different areas and unveil the storyline. However, the player does spawn on one half of a plane fuselage in a random area and resources such as trees are randomly generated.

The game uses a unique inventory system where they can see and use everything they have collected and the only inventory limiting factor is the number of the same items they may have as shown in Figure E.5. Otherwise, the player has no weight restrictions or inventory size. The player can also craft on this screen where they can place and combine various items in the middle to create a new item. Every item has a fixed location on where to show up on the inventory which means the game resources are fixed and it may be hard to add new items in the future.

Figure E.5: Screenshot of the inventory screen in The Forest

The theme of the game has a strong relation to the survival aspect as the player builds structures from their survival manual and may also read about some of the local flora and fauna as shown in Figure E.6. Blueprints from the manual can be built by selecting and placing a ghost blueprint of that item in the world and feeding it resources to slowly build it.

Figure E.6: Screenshot of the building menu in The Forest

When it comes to obstacles, the enemies are a huge factor in The Forest as the player is constantly met with cannibals that are naturally aggressive towards the player. There are ways to 'tame' some low-tier enemies by applying camouflage/body paint to deal with common enemies, but it is rather hard to obtain. But using enemies to slow down the player's progress emphasizes the importance of base building and weapon advancement. The level of enemies is increased each day which may be unfair to new players however they may choose an easier difficulty.

Leana Hafer also reviews The Forest giving it a rating of 8.4/10 and summarizing the game as a “harrowing survival ordeal that knows how to play with tension and create a sense of a real world with complex inner workings and mysteries” with a “meaningful story progression that doesn’t overstay its welcome” and that “if you want a unique and memorable survival horror experience, then you should absolutely do dare to do so” [9]. The game’s use of a story completely shapes the game as it gives the player reason to survive and explore the mysteries of the character’s story, immersing the player to want to uncover more of the story rather than thinking about what to do in the game.

The player can select different difficulties between peaceful, normal, hard, hard survival or creative where peaceful means no enemies, hard survival (hardest difficulty) means sustenance is less nourishing and more damage is taken from every source, and creative where the player has no vitals and may play freely.

Appendix F - Project Plan

Gantt Chart

The Gantt chart in Figure F.1 shows estimate times for starting key tasks of the project plan

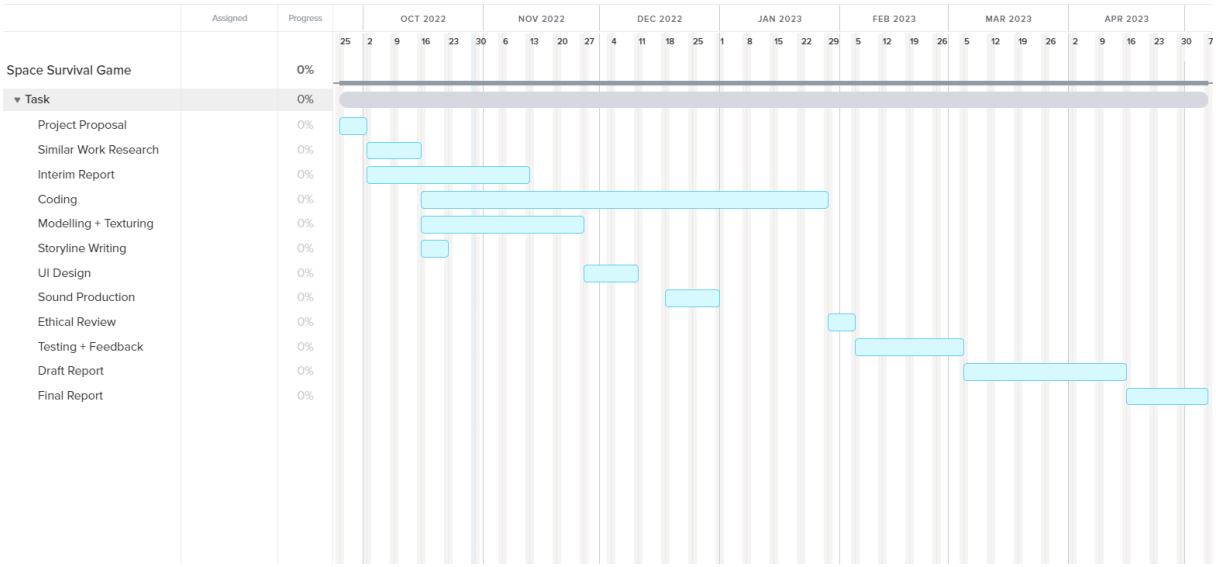

Figure F.1: Gantt chart of project plan

- Project Proposal – Work on project proposal and submit it to supervisor. This task must be done first to get an idea of what to create and include and the potential problems that may emerge.
- Similar Work Research - Research for similar games and observe for pros and cons along with issues with the feature.
- Interim Report – Draft an interim report and send it to supervisor for improvement.
- Coding – Start working on core functionalities of the games such as the player movement, game systems such as world generation, inventory system, crafting system, ETC.
- Modelling + Texturing – Start along with coding and create models and textures for models to implement in the game.
- Storyline Writing – Produce a short story line/goal for the game to create the world around and create scripts to work along with them.
- UI Design – Design and implement graphical interfaces for the inventory system, menu, ETC.
- Sound Production – Create/modify sounds and implement them into the game. Find/create background music.
- Testing + Feedback – Test game with volunteers and collect feedback to improve game.
- Draft Report – Start drafting the final report for the whole project and send it to supervisor to improve.
- Final Report – Finalize final report with feedback from supervisor.

Kanban board and GitHub

GitHub will be used to keep track of what is being developed and what will need to be tested. GitHub provides a Kanban board as shown in Figure F.2 to do this and easily swap the states of each task. Git will also be used as source control for development when implementing each feature.

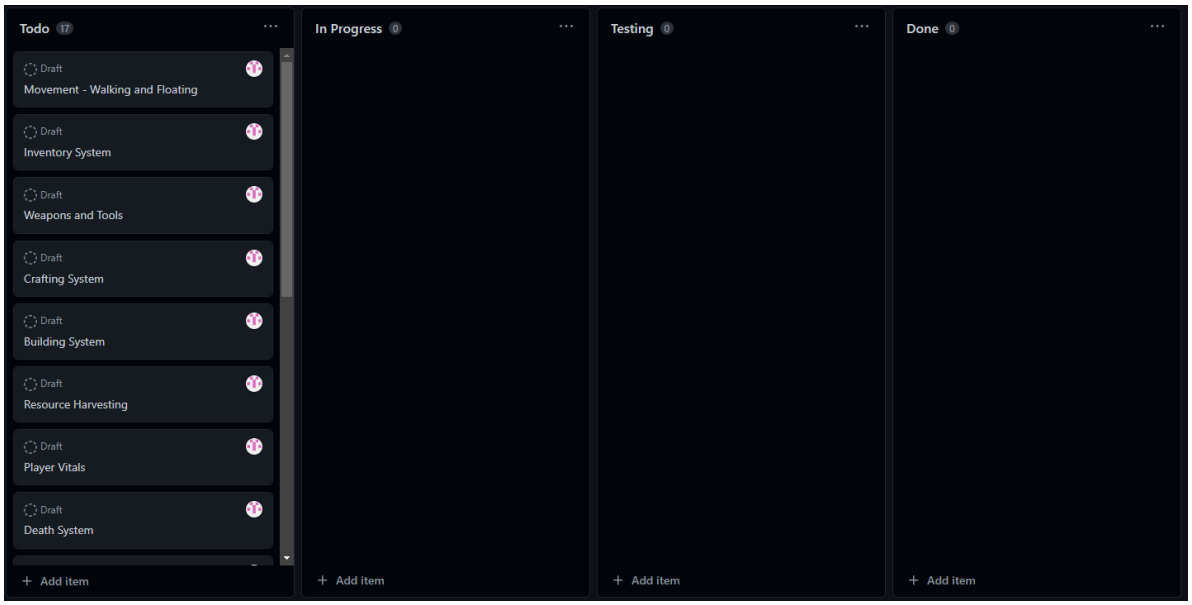

Figure F.2: Kanban board of development plan on GitHub

An alternative view of the Kanban board is the table view as shown in Figure F.3 showing more detail of each task set with ease of viewing.

	Title	Description	Notes	Status
1	Movement - Walking and Floating	Gravity changing from different areas	Walking, Floating	Todo
2	Inventory System	Inventory to hold many types of items	Items, Weapons, Tools, Consumables, Upgrade slots?	Todo
3	Weapons and Tools	Tools to gather resources and weapons to attack	See Description.	Todo
4	Crafting System	Crafting interface(s?) to craft different types of items	Recipes, Craft then take from slot, Timed?	Todo
5	Building System	Uses items and resources to build with blueprints	Snapping, Blueprint, Build progression	Todo
6	Resource Harvesting	Harvestable resources using tools	Levels of harvesting, Lucky/Secondary resource, Progressive	Todo
7	Player Vitals	Vitals that needs to be managed	Health, Thirst, Hunger, Oxygen, Radiation	Todo
8	Death System	A way to manage player's death	Drop items, Return (some) items	Todo
9	Hazards	Hazards around map that damage player	Traps, Radiation, Heat, Cold	Todo
10	Enemies	Enemies that attack player	Drops loot, Different types	Todo
11	Difficulties	Different difficulties to play the game on	Damage multi., Nourishments, Oxygen, Enemy level, Resources	Todo
12	Story	Map design for story and objectives	Objectives, Puzzles, Dialogue	Todo
13	Save System	Save and load worlds	Transforms, Vitals, Inventory	Todo
14	Vehicles	Manoeuvrable vehicles		Todo
15	Advanced Enemies	Enemies with tactics, evading and calculated moves		Todo
16	Upgradable Tools	Use resources to upgrade tool stats - Efficiency, unbreaking		Todo
17	Pets	Pets that follow and give buffs		Todo

Figure F.3: Table view of development plan on GitHub

Story And Map Planning

Figure F.4 shows a brief sketch of the map. The map is designed for optimization where occlusion culling can be utilized to hide areas of the map that would be unnecessarily drawn. It also promotes the idea of exploration by hiding or placing landmarks far from sight for the player to just about see them as they explore. The world is also surrounded by a hazardous border keeping the player within the play zone.

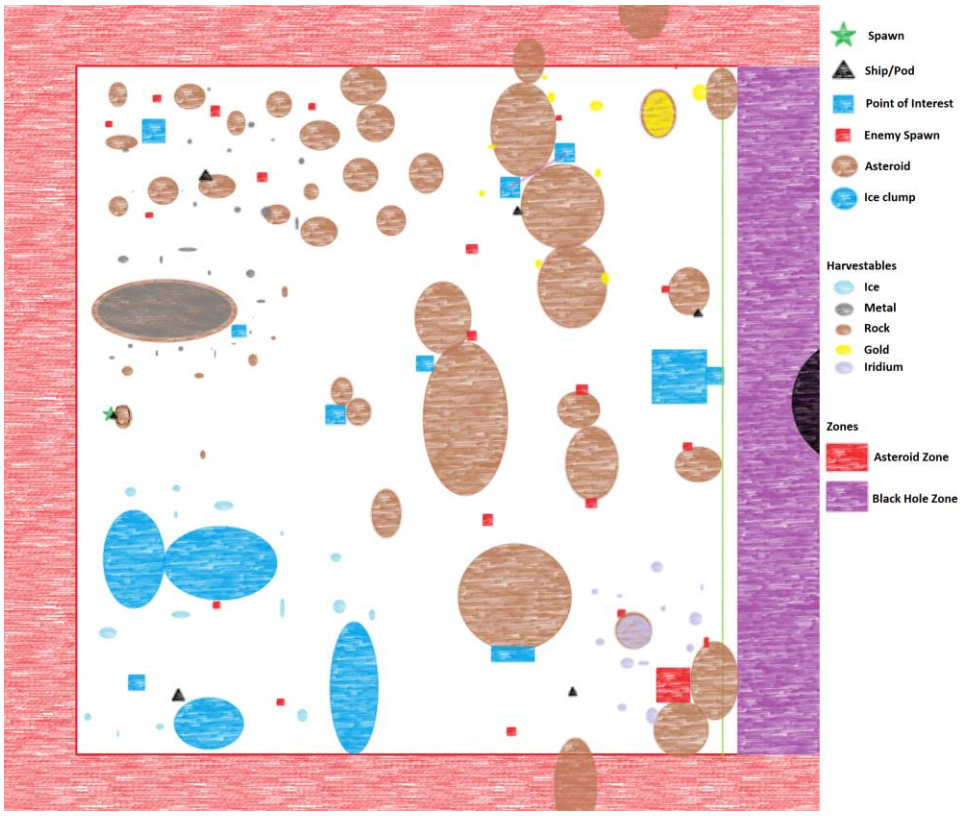

Figure F.4 Rough sketch for the map

Appendix G - User Testing Compliance Form

User Testing Compliance Form for UG and PGT Projects[1]

School of Engineering and Informatics

University of Sussex

This form should be used in conjunction with the document entitled "Research Ethics Guidance for UG and PGT Projects".

Prior to conducting your project, you and your supervisor will have discussed the ethical implications of your research. If it was determined that your proposed project would comply with **all** of the points in this form, then both you and your supervisor should complete and sign the form on page 3, and submit the signed copy with your final project report/dissertation.

If this is not the case, you should refer back to the "Research Ethics Guidance for UG and PGT Projects" document for further guidance.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.

   Investigators have a responsibility to protect participants from physical, mental and emotional harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, physical hazards or discomfort, emotional distress, use of sensory deprivation (e.g. ear plugs or blindfolds), sensitive topics (e.g. sexual activity, drug use, political behaviour, ethnicity) or those which might induce discomfort, stress or anxiety (e.g. violent video games), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.

2. The study materials were paper-based, or comprised software running on standard hardware.

   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and tablet computers is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.

   Participants cannot take part in the study without their knowledge or consent (i.e. no covert observation). Covert observation, deception or withholding information are deemed to be high risk and require ethical approval through the relevant C-REC.

   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, the data is to be published or there are future secondary uses of the data), then it will be necessary to obtain signed consent from each participant. Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script (see Appendix 1).

4. No incentives were offered to the participants.

   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle. People volunteering to participate in research may be compensated financially e.g. for reasonable travel expenses. Payments made to individuals must not be so large as to induce individuals to risk harm beyond that which they would usually undertake.

5. No information about the evaluation or materials was intentionally withheld from the participants.

   Withholding information from participants or misleading them is unacceptable without justifiable reasons for doing so. Any projects requiring deception (for example, only telling participants of the true purpose of the study afterwards so as not to influence their behaviour) are deemed high risk and require approval from the relevant C-REC.

6. No participant was under the age of 18.

   Any studies involving children or young people are deemed to be high risk and require ethical approval through the relevant C-REC.

7. No participant had a disability or impairment that may have limited their understanding or communication or capacity to consent.

   Projects involving participants with disabilities are deemed to be high risk and require ethical approval from the relevant C-REC.

8. Neither I nor my supervisor are in a position of authority or influence over any of the participants.

   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any study.

9. All participants were informed that they could withdraw at any time.

   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script (see Appendix 1).

10. All participants have been informed of my contact details, and the contact details of my supervisor.

    All participants must be able to contact the investigator and/or the supervisor after the investigation. They should be given contact details for both student and supervisor as part of the debriefing.

11. The evaluation was described in detail with all of the participants at the beginning of the session, and participants were fully debriefed at the end of the session. All participants were given the opportunity to ask questions at both the beginning and end of the session.

    Participants must be provided with sufficient information prior to starting the session, and in the debriefing, to enable them to understand the nature of the investigation.

12. All the data collected from the participants is stored securely, and in an anonymous form.

    All participant data (hard-copy and soft-copy) should be stored securely (i.e. locked filing cabinets for hard copy, password protected computer for electronic data), and in an anonymised form.

Project title: 3D Unity Game – Lost Horizons

Student's Name: Thanh Ngo Trung

Student's Registration Number: 22001340

Student's Signature: 

Date: 03/11/2022

Supervisor's Name: Paul Newbury

Supervisor's Signature: 

Date: 03.11.2022