CS3010 Bachelor's Report in Computer Science

Report on Research and Development of a Novel Video Game

School of Engineering & Applied Sciences, Aston University, Birmingham

Author

Matthew Ahearn

190232957

Supervisor

Ulysses Bernardet

April 2022

Table of Contents

1. Introduction

Video games that have developed independently from the largest corporations are an increasingly popular corner of the video gaming market, and cover a large number of game genres which are typically not touched by studios with huge budgets. This opens an area for the investigation and development of a novel video game which can be completed within the scope of this project and is capable of fulfilling its own niche within the existing market. With Steam as the only publisher providing consistent and up-to-date statistics, its data will be utilised for some analysis, and is shown in figures 1 and 2.

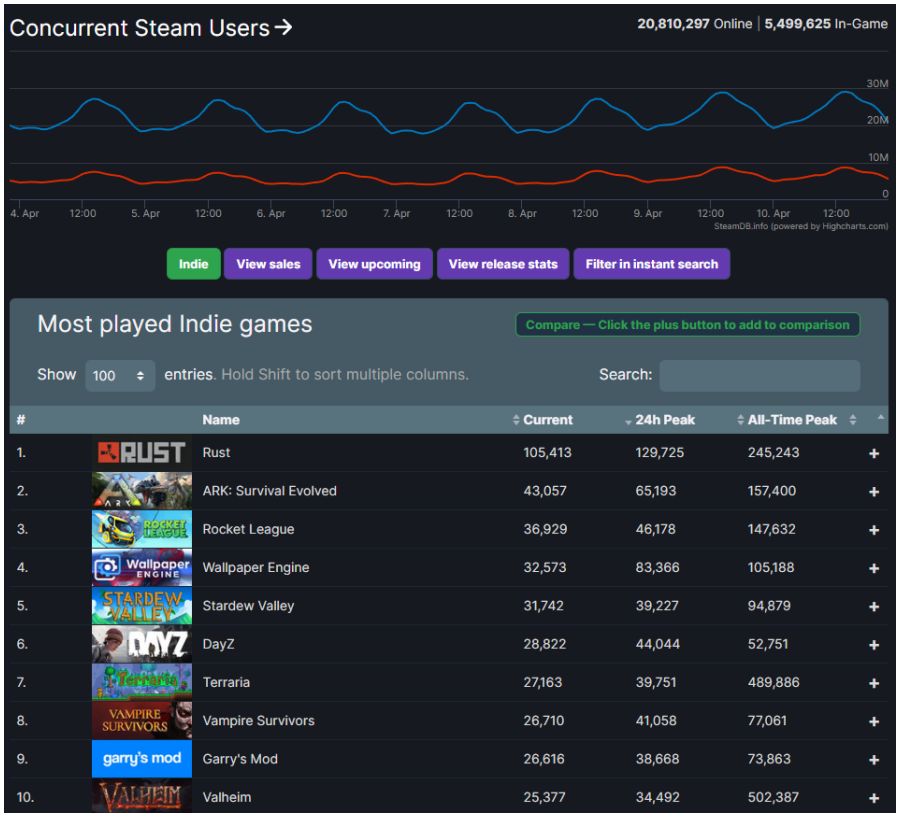


Figure 1: The Indie tag Steam chart for 14/03/2022 displaying the numbers of users playing the top 10 indie titles

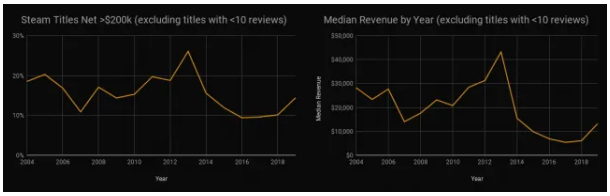
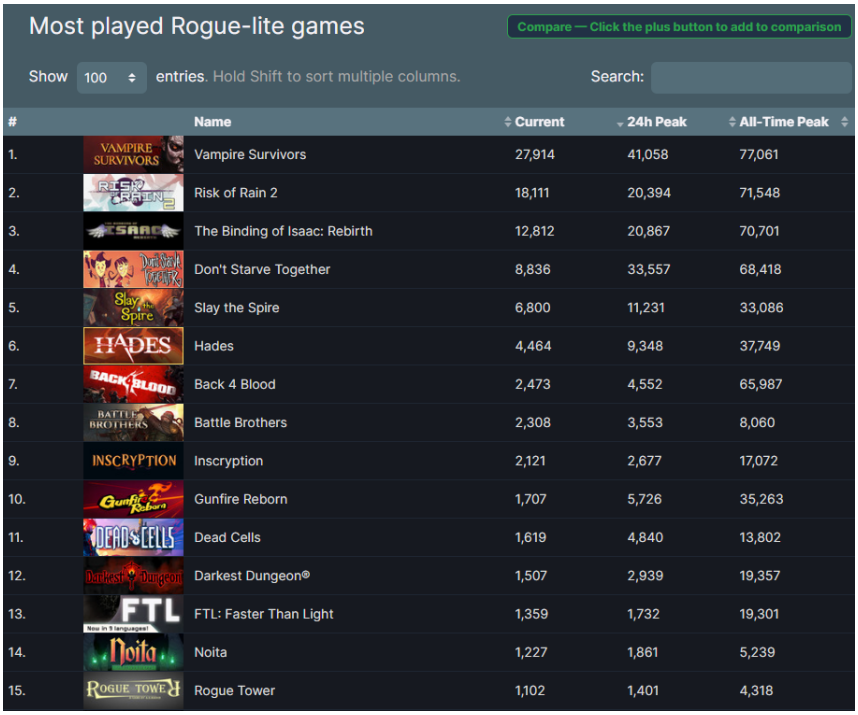


Figure 2: An analysis of games with revenue exceeding $200k, and median revenue of games with more than 10 reviews, from 2014 to 2019, compiled by Danny Weinbaum[30]

This project will follow the creation of a novel video game within an identified empty space of the market. Games played from the first-person perspective have historically been extremely popular, with the primary sub-genre being action games utilising ranged weapons - known as a First Person Shooter. A recent trend in gaming has seen the development of the so-called Roguelike, a system whereby the player enters into levels and tries to progress as far as possible, collecting upgrades which can be applied when the player fails in their objective and returns back to the start of the game. This loop results in a player gradually increasing in power and learning new game systems

over time, and lowers the workload of the developers as the same areas will be repeated. However, these two genres have never been combined to make a Roguelike First Person Shooter, which will make up the premise of the novel game concept. Data supporting the audience size of the rogue-like genre is found in figures 3 and 4, again from the freely available Steam data - though it must be stressed again that not all roguelikes may be on Steam, and many sales will have taken place on alternative platforms.



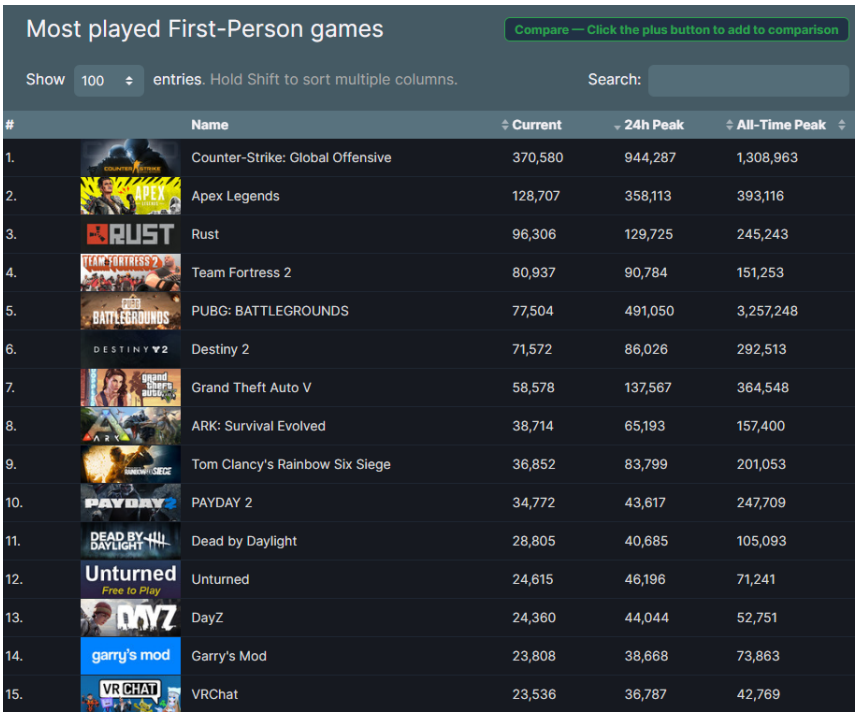
Most played Rogue-lite games

Compare — Click the plus button to add to comparison

Show 100 entries. Hold Shift to sort multiple columns. Search:

#	Name	Current	24h Peak	All-Time Peak
1.	Vampire Survivors	27,914	41,058	77,061
2.	Risk of Rain 2	18,111	20,394	71,548
3.	The Binding of Isaac: Rebirth	12,812	20,867	70,701
4.	Don't Starve Together	8,836	33,557	68,418
5.	Slay the Spire	6,800	11,231	33,086
6.	Hades	4,464	9,348	37,749
7.	Back 4 Blood	2,473	4,552	65,987
8.	Battle Brothers	2,308	3,553	8,060
9.	Inscryption	2,121	2,677	17,072
10.	Gunfire Reborn	1,707	5,726	35,263
11.	Dead Cells	1,619	4,840	13,802
12.	Darkest Dungeon®	1,507	2,939	19,357
13.	FTL: Faster Than Light	1,359	1,732	19,301
14.	Noita	1,227	1,861	5,239
15.	Rogue Tower	1,102	1,401	4,318

Figure 3: Current and peak player counts for the rogue-lite category, which encompasses all forms of rogue-like games - though limited only to data from the Steam platform.



Most played First-Person games

Compare — Click the plus button to add to comparison

Show 100 entries. Hold Shift to sort multiple columns. Search:

#	Name	Current	24h Peak	All-Time Peak
1.	Counter-Strike: Global Offensive	370,580	944,287	1,308,963
2.	Apex Legends	128,707	358,113	393,116
3.	Rust	96,306	129,725	245,243
4.	Team Fortress 2	80,937	90,784	151,253
5.	PUBG: BATTLEGROUNDS	77,504	491,050	3,257,248
6.	Destiny 2	71,572	86,026	292,513
7.	Grand Theft Auto V	58,578	137,567	364,548
8.	ARK: Survival Evolved	38,714	65,193	157,400
9.	Tom Clancy's Rainbow Six Siege	36,852	83,799	201,053
10.	PAYDAY 2	34,772	43,617	247,709
11.	Dead by Daylight	28,805	40,685	105,093
12.	Unturned	24,615	46,196	71,241
13.	DayZ	24,360	44,044	52,751
14.	Garry's Mod	23,808	38,668	73,863
15.	VRChat	23,536	36,787	42,769

Figure 4: Current and peak player counts for the First Person category.

As a declaration before going too far into this report - some 3rd party assets have been used in this project. They have been placed within the /Assets/3rd Party Assets/ folder in the project files,

there is a declaration within the /Assets/ folder, and the assets are also documented in various places in this report. Additionally, a link to the project source files - including the Unity files (using version 2021.1.21f1 - IMPORTANT), the blender source files and FBX files and all official documentation relating to this project - is found in the final Appendix, on the last page. Finally, the Project Definition Form is found in Appendix A, where it won't disturb every other figure in this report.

2. Design Document

2.1. Vision Statement

The creation of a sci-fi PC game where the player combines fast paced and engaging movement-mechanics with long and short ranged combat in a bright, neon-lit futuristic environment. By completing levels, the player will be rewarded with unique upgrades which can completely change the way the game plays, resulting in the ability to combine upgrades to tailor your character to your personal playstyle. By playing through repeatedly, the player will be able to reach a point where they can fully defeat the randomly generated space station and its enemies. The game will be easy to pick up, with intuitive controls and mechanics that react smoothly to input. By providing the player with randomly selected upgrades in random environments, each attempt to beat the game will be unique and challenging in new ways.

2.2. Target Audience

The intended user-base is very broad. Utilising the industry standard ESRB ratings, the Everyone rating category is the intended audience, though targeted specifically toward the Western sphere of gaming audiences, as there are wide divides between Western (Europe, American continent) and Eastern (Indian, Asian continents) in terms of taste. By the ESRB definition: 'Content is generally suitable for all ages. May contain minimal cartoon, fantasy or mild violence and/or use of mild language'[8]. The game as envisioned fits this descriptor neatly: the only violence will be against stylised robots, which will explode on death, there will be no language, and the game itself will support all ages by providing strong support for learning the systems and clearly marking the progression. Accessibility options for a variety of player styles or disabilities must be provided: including additional navigation or combat support, colour-blind modes, options for different control schemes and etc.

2.3. Genre

A roguelite first person action game set in a science fiction environment. The roguelike genre is a derivation of the roguelike genre - characterised by randomly generated environments and rewards, and permanent death for the player character whereby they must re-enter a newly generated dungeon. The roguelike adds on a layer of permanence – some upgrades will continue over after the death of the player and ease the difficulty of the next dungeon. In both cases, the player will continue to navigate new dungeons until they are bored, or the story is completed. The name itself is derived from the 1980 game Rogue, where the player attempts to navigate randomly

generated dungeons with permanent death for the character and no carried over resources. The first-person action genre consists of games set from the first-person perspective – from the player character's eyes – with a focus on action. In this case, the action will be gun combat, with additional movement mechanics designed to facilitate dodging enemy attacks and boosting the players' own damage and combat abilities.

2.4. Core Gameplay

The player will be running, jumping, dashing and shooting through arenas of enemies. The short-term goal of the player is to survive until the end of the current arena. This will consist of fighting waves of generated enemies, each spawned by a wave controller which will attempt to choose enemies to spawn and which will complement each other and the current arena. By dying, the player will lose this goal and be returned to their home base, losing any temporary upgrades. By completing the short-term goals, the player will be rewarded with permanent currency which can be used to purchase upgrades for the future, or with temporary, powerful upgrades which will make future combat easier. The medium-term goal is to progress further into an attempt than ever before. This will be done by completing more rooms than any previous attempt and will reward the player with higher value rewards – better upgrades, more currency or different types of currency. This goal will fail upon death. The long-term goal of the player is to survive until the end of a dungeon – the series of rooms which makes up the current attempt. The difficulty of the arenas ramps up the further into the dungeon the player travels, and it should not be possible on a first attempt. Reaching the end provides the best rewards – currency for huge permanent benefits, or interesting interactions with the ongoing storyline.

2.5. Core Components

There will be four enemies in the game. The first is a small enemy with low health which simply charges the player in an attempt to deal damage. They will spawn as a swarm, and the player will have to rely on either fast, accurate shots, or area-of-effect abilities - which target a wider zone, typically dealing less damage while being easier to aim and hitting multiple targets in one. This enemy will take the form of a small hovering drone with a superheated melee weapon. The second is a large enemy with a high amount of health and very high damage, which requires a charge up time before hitting the player within melee range. This provides a chance for the player to dodge the attack, with high punishment for failure. This enemy will be very large and utilise a charging up hammer attack. The third enemy is one which will charge up a short attack some distance from the player, before charging in and holding the player in place for a duration. The player will be able to "button-mash" (hitting a button repeatedly) to break free or shoot the enemy to stop the effect. The enemy will have medium health and low damage to make up for it trying to facilitate the attacks of others, and will be a hovering drone with a taser on the front and will be larger than the swarm enemy. It will have a large jet on the back which will charge up for attacks. The fourth enemy is fast, with low health but ranged attacks which either deal a high amount of damage or fire many projectiles, each dealing low to medium damage. This should fill the arena with projectiles which need dodging, while providing a different challenge to the melee enemies. This enemy will also run faster than the player and attempt to maintain distance. There will be plenty of cover available, in the form of shipping containers, consoles, etc. Explosive barrels, painted red as traditional, will be scattered randomly in arenas. There will be areas which damage the player, such as infinite holes or superheated liquids, which must be dodged. There will be boxes

the player can open dotted around the rooms which each provide a small amount of currency, so the player will still gain overall progress even if they focus entirely on temporary upgrades as their main rewards. These boxes will unseal and swing open at the top. Additionally, some enemies will drop currency. Each room will be sealed by doors. Once unlocked, by killing every enemy or some other mechanism depending on the door purpose, the player will be able to pull a nearby switch to open the door. The doors will shut behind the player and not be reopenable. The doors will be double wide, Star Trek style sliding doors, pulling to the left and right. A bright light will be placed prominently above the door, to guide the player to it and provide information on its state. There will be boss fights at the end of the dungeons before the player can claim the final reward and successfully return to their ship without dying. These boss fights will take the form of enhanced waves of enemies, and bigger variants of the enemies with upgraded health, damage and attack speed to provide more challenge.

2.6. Controls

The game will have built in options for mouse and keyboard, and gamepads. Jumping will be performed on the space or A button, fire on left mouse or right trigger, dashing on control or the right bumper, interaction with the environment on the E or X button, sprinting on the shift or pushing in the left joystick. Aiming will be performed with the mouse or the right joystick, and movement through the WASD keys or left joystick.

2.7. User Experience

The main menu screen will consist of the start/load game button (there will only be one active save file at a time), an options menu and a quit button. The options menu will contain all relevant graphical, controls, accessibility and gameplay options split into their own sub-menus. The game will itself have a pause menu with resume, options, restart game and quit to main menu/desktop buttons. There will be some UI elements in the game – the windows where upgrades are selected or purchased, statistics tracking the player throughout the game etc. These will be laid out as simply as possible and support controller input for all menus.

2.8. Visual Style

The game is set in two locations – the player's ship and the dungeons. The player's ship will be a spaceship with a central chamber allowing the player to select the next mission on a holographic screen, and there will be secondary rooms including an armoury, docking room, and respawn room. The dungeons will be set inside space stations. These rooms will be largely utilitarian and human in design – think Battlestar Galactica, Star Gate, the Alliance vessels from Mass Effect. The rooms will be themed around various functions but will ultimately be randomised. The art style will be quite stylised, making large use of bright neon lighting to accentuate the future setting, and flat, textured surfaces, as a method to improve visibility and navigation while still maintaining interesting visuals. There will be some attention to realism, but ultimately the "rule of cool" applies – anything which would look great could be used.

2.9. Characters and Storyline

There will be no characters or storyline present in the game, as development of a storyline would be far beyond the available time allocations.

3. Background Research

Not everything researched in this section will make it into the final project, and by no means should all of it be forced in - this is more exploratory research, which will both inform the project on the course forward and of alternative solutions to encountered challenges.

3.1. Analysis on Existing Games & General Gaming Space

3.1.1. Gaming Trends

Some of the most popular games of the last generations were 'movement shooters'. Doom (2016 - from now on referred to as 'Doom') reportedly sold 3.6 million copies by the end of 2016 - using data collected by SteamSpy, which counts the number of Steam users with the game tied to their account[25], and VGChartz, who track physical copies shipped to stores[29] - leaving plenty of room for error, but a ballpark area. Doom Eternal (2020), selling 3 million digital copies in the first month - reported by the now defunct Superdata Research, but reported in affiliated news sites[11]. Doom, as aggregated by MetaCritic, received critical scores averaging 85/100[2], while Doom Eternal scored 87/100[1].

Roguelikes, the main genre of this project, are also increasing in popularity. Darkest Dungeon (2016) sold over 2 million copies in its lifespan and maintains a strong average of close to 2,000 daily concurrent players even now, as displayed in fig 5.

Figure 5: Darkest Dungeon Steam statistics page

Hades (2020), a top-down roguelike from the well-regarded Supergiant Games, continues the trend by selling over 1 million copies[10], and a strong present-day player-base on Steam, displayed in the steam charts in fig 6.

Finally for this small look, Loop Hero, a 2021 indie breakout, sold 500,000 copies in its first

Figure 6: Hades Steam statistics page

week[32] with a million by the end of the year[9] - extremely successful for a game developed by four people styled after classic 80's and 90's pixel-based games.

3.1.2. Selection of Games to Review

The games to review must be relevant to the genre - movement shooters. Reviewing games which have been highly praised by the public and critics would allow the analysis to show what makes a good movement shooter and roguelike well received by players. Additionally, the ability to, or prior experience of playing the game will prove hugely useful, further narrowing the options available. The two largest movement shooter games released in the past decade are Doom and Doom Eternal, so they make perfect fits for this analysis. Alongside them, Hades will be analysed, as a recent game filled to the brim with unique mechanics and, as it turns out, more than a few similar mechanics.

3.1.3. Analysis of Selection

The most basic component of any game is movement - particularly movement shooters. Doom set a new standard for the genre - extremely fast but controllable movement, utilising mechanics previously only included in platforming games such as coyote time (named after the moments of anti-gravity the coyote in Disney's Looney Tunes has after falling off the edge of an object) and jump buffering (whereby pressing the jump key while still in the air queues up a second jump to perform upon hitting the floor), movement boosts while jumping and extremely tight controls. These combined allowed the games arenas to function as platforming puzzles as much as combat arenas. Doom Eternal, as a direct sequel, included the same movement system but augmented it with the ability to dash and more vertical mobility. These systems allowed the enemies to utilise more damaging attacks, forcing the player to rely on the brief invulnerability granted by dashing, as well as more advanced platforming mechanics - though these were often disliked by the community, and proved somewhat frustrating to perform in a first-person view. Finally, Hades also included the dash ability, with progression options available to upgrade its effectiveness and turn it into its own form of weapon. This ability made the dash an aggressive as well as defensive ability, allowing the player to dash in and out of combat and dodge more dangerous attacks in the process. As an aside, other games also feature a dash ability - it is a core component of the Dark Souls series and some characters in Overwatch and Apex Legends, and has proven popular in each case.

Combat is the second largest pillar. Both Dooms had largely similar combat - an array of weapons to switch between for any combat scenario, very high damage but low defences, and recharging health and ammunition through risky combat moves. Enemies are highly mobile, capable of navigating the complex maps and supporting each other with a wide array of abilities - basic enemies charge and melee, some throw projectiles, some have multiple projectile attacks to cover a wide area, and some provide boosts to their allies. Hades, on the other hand, provides a slowly increasing amount of damage and defences, with unlimited ammunition and very limited access to health regeneration due to the requirements of a roguelike - not only should the player be capable of dying, but they are also expected to die and lose their progress multiple times. Hades also restricts the player to a single weapon, forcing them to choose before the run what kind of gameplay experience they want, and having to work around the enemies the weapon is unsuited for dealing with. The AI is comparable to the Dooms, with several dozen enemy archetypes which all provide unique challenges, designed to function together to cover the weaknesses of their allies.

Level design is another similarity between the games. Doom and Hades follow very similar philosophies - the player enters a handmade room fitting the style of combat in the game, enemies spawn in waves, and once the player defeats them, they can move on. Doom Eternal expands upon this by making the levels seem open-world in format and providing much larger levels with different types of enemy wave system - some have hubs which must be destroyed, some simply spawn in groups, some contain large boss fights and scripted sequences. Hades expands upon the level system by providing the player with choices for upgrades at the end of each room, giving the player some limited choice over which upgrades they want. This system is particularly enjoyed by players who over time have come to desire more choice over the randomness inherent in many roguelike games and provides a more casual experience.

3.1.4. Conclusions

Several conclusions can be taken away. With noted exceptions, the mechanics described have to contribute to the popularity of each title, so it can be assumed they are beneficial. From the mechanics described, it is obvious that a movement-based game must have strong movement - fast, precise, and with mechanics to allow easy navigation and platforming. A dash ability with invincibility frames is also extremely popular, allowing talented players to dodge all damage while still providing casual players a way to out-play the AI in a satisfying manner.

Viewing gameplay of the Dooms, and from personal experience, it is shown that many players do not swap out weapons while in the heat of combat. This is not a problem, as the game is forgiving, but it does present an underutilised mechanic resulting from the inability of many to track so many fast-moving variables at once. The solution in Hades of making the weapon a choice before the game is ideal, removing this downfall and turning it into an extra form of engagement. The ability to add damage to the dash ability is also interesting and should be included, while the AI designs found in all three games provide a lot of inspiration for the design of AI in this project. A small selection covering the main bases of swarms, ranged, area-denial and buff enemy types would provide the kind of challenge presented across these games.

Finally, Hades has little to deliver in terms of level design aside from the ability to have some choice when it comes to random upgrades. This addition has proven extremely popular with fans and helped push it to the very large casual audience. Doom Eternal has levels far too expansive and complex to replicate with a single person team - but Doom provides a lot of inspiration. The tight arenas force the player to carefully track their surroundings and would be much more

manageable to create for a single developer.
The design of Hades and Doom both lend themselves to the kind of random generation being considered for this project - the handmade designs are very enjoyable to play in, but present low replayability. The modular look to both games would theoretically be capable of being randomly attached to other modules, expanding the level pool exponentially with the modules created.

3.2. Tools Research & Decisions

3.2.1. Game Engine

The first important decision which will influence all future decisions is the choice of game engine. There are multiple available, and a custom engine was not pursued for this project due to time constraints. Unity, Unreal and Godot are all popular solutions, and were considered.

Unity as a general platform provides excellent support for solo game developers. The tools are easy to understand and utilise, and the documentation is very extensive, with many third-party resources available online. The editor screen is easy to navigate and provides access to the Game Object design system, with the inspector panel giving instant access to the variables of any object and its assigned classes and components. On top of this, classes are compiled upon save, which allows the game to be run from within the editor - skipping the potentially lengthy full compilation time.

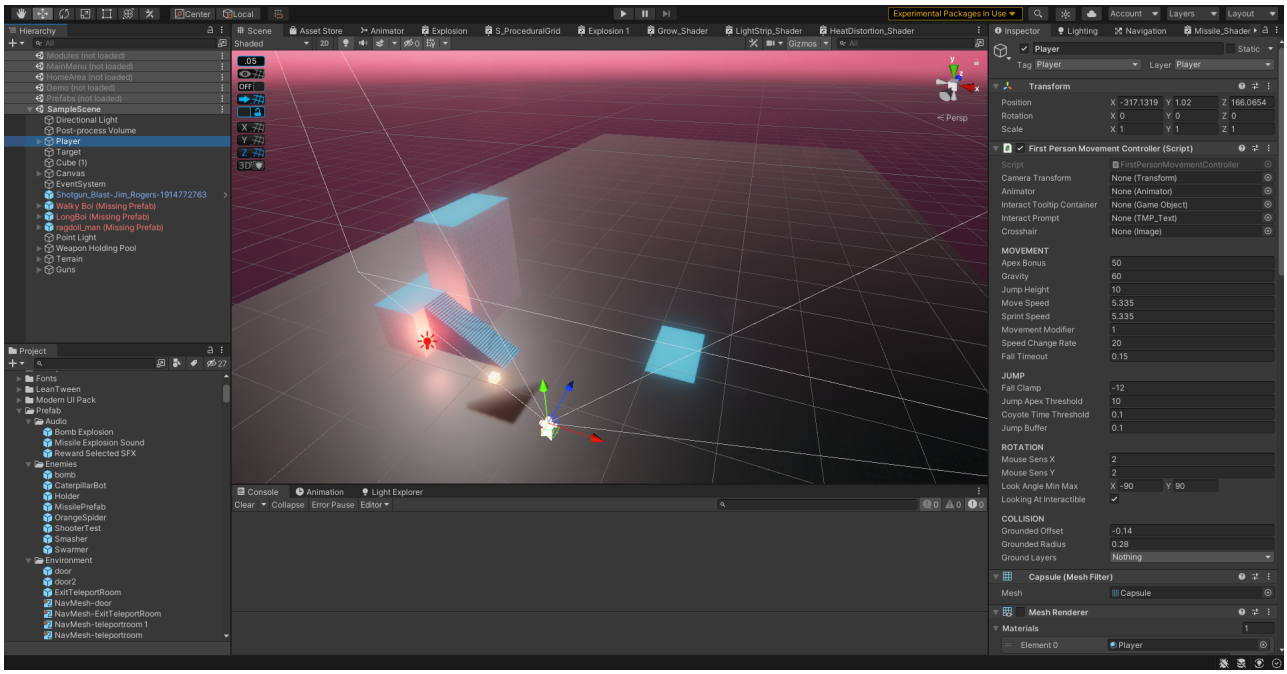


Figure 7: The editor screen for Unity

A built-in shader-graph system is also provided. Shaders are created in code - for Unity, using either HLSL or ShaderLab. With the rise of artist centric tools skipping the need for an available programming team, shader graphs have become increasingly popular, providing the ability to visually construct complex code and provide real-time previews of the results. By connecting nodes comprising of visual generation techniques (Voronoi, white noise, custom made assets etc.) to manipulation, conditional and mathematical nodes, practically any visual effect can be achieved.

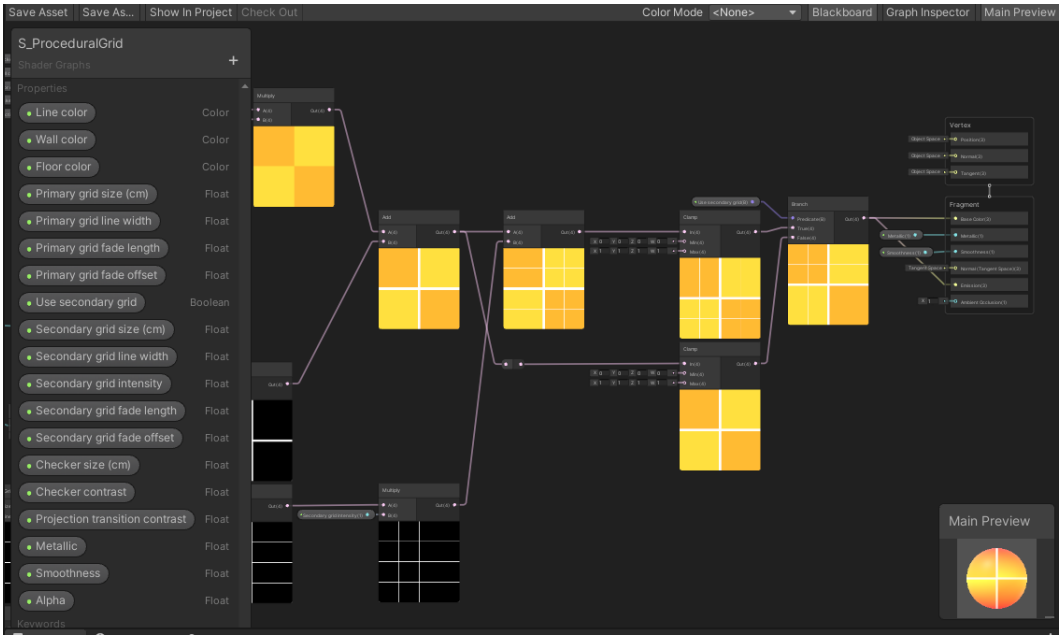


Figure 8: The Unity shadergraph, displaying the custom procedural grid generation shader

To accompany the shader graph is the relatively new visual effects graph - providing similar node-based tools to create particle effects, capable of producing tens of thousands of particles in real-time with little to no performance impact.

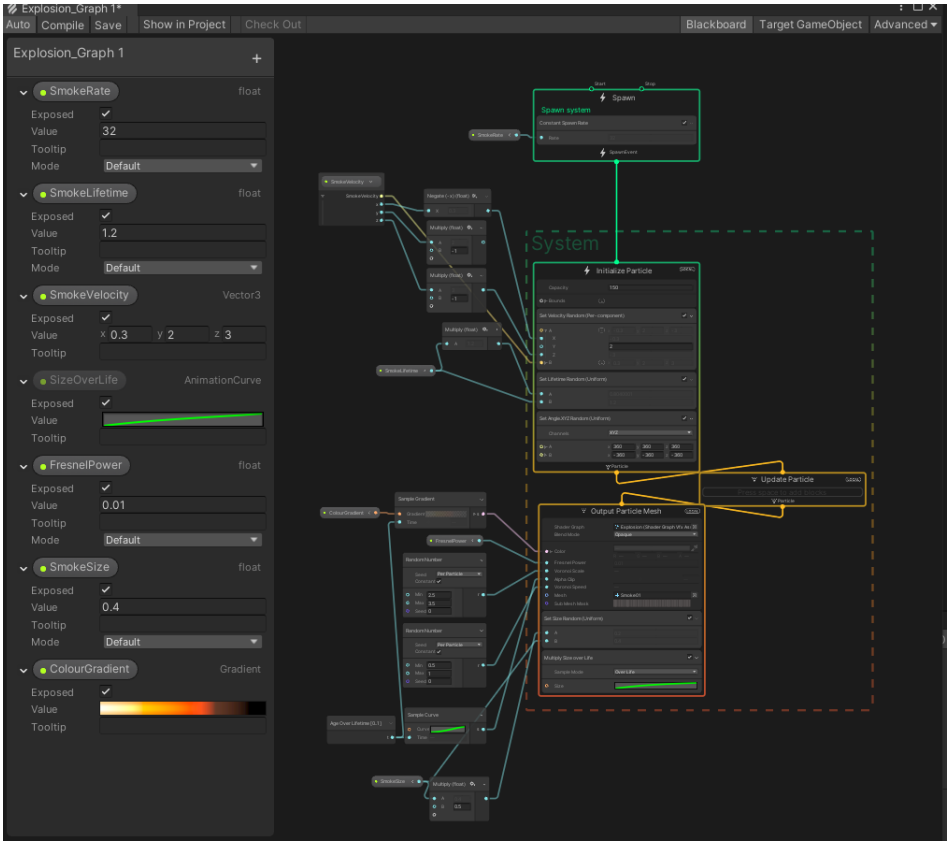


Figure 9: The Unity VFX graph, displaying the custom explosion effect used for a variety of events

Finally, Unity provides a built in, comprehensive animation state machine. This visually based state machine allows animations to be placed on a blackboard, connecting them with conditions for traversal and unlocking a suite of variables - such as allowing the animation to finish playing

before starting the next one, whether to loop, whether to blend seamlessly from one to the other, and more.

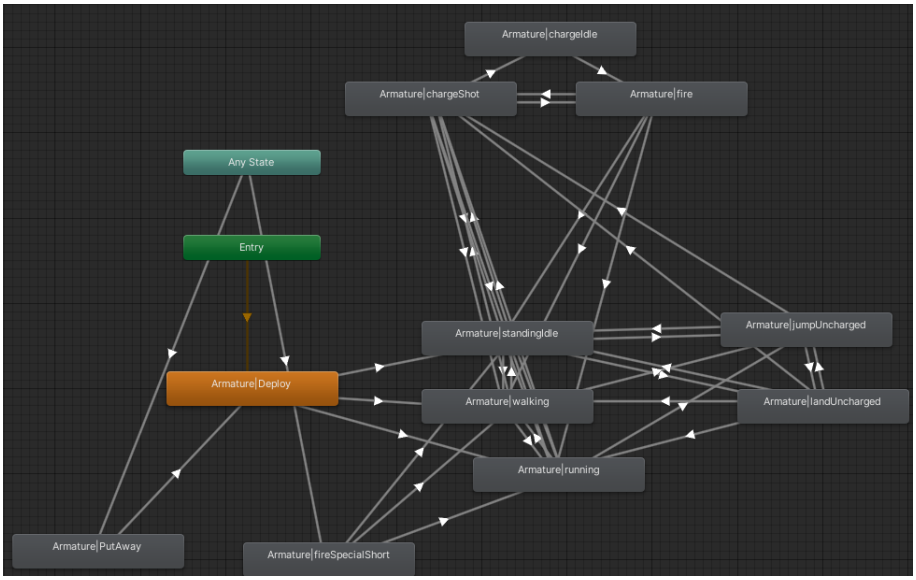


Figure 10: The animation state machine, displaying the animations and links for the player

Unreal was considered second and provides an interesting alternative to Unity. It has many similar features, with expanded visual scripting capability, and is renowned for its extremely high-quality graphical capabilities. This, however, comes at a cost for independent developers - the pipeline within Unreal is built for large companies, with dedicated visual artists, modellers and programmers. It is notoriously difficult to pick up solo and has far fewer resources available online. This has been changing - the last years has seen an uptake in the engine, and especially with the reveal of Unreal 5, there has been a focus on the usability of the software. At the time of choosing the software, however, Unreal 5 was not released, and so had to be discounted.

Additionally, Unreal functions on C++, a language which would be difficult to learn while creating a game and carries more of a reputation for being difficult to work with than C#. Regardless, it is an engine worth considering.

Godot is a much smaller development than the two before - an indie engine for indie developers. While this makes it quite easy to use as it is designed ground up for smaller teams, it does also reduce its audience - and by extension, the available resources and communities online. While Godot provides the ability to use multiple languages, including C# and C++, the ones with the best level of integration and official support are GDScript and VisualScript - GDScript being unique to Godot but with similar syntax to Python, and VisualScript being a custom visual scripting solution, allowing the developer to place nodes and connections visually, being designed for non-programmers. On top of this, the shader graphs and visual effect graphs found in Unity and Unreal are not present, making shader and effects creation far trickier.

The decision was made to use the 2021.1 branch of Unity. This was upgraded to product support prior to the start of this project - making it the most recent stable branch, although it lacks some of the experimental features introduced in 2021.2. Of note from features introduced in 2021.1 is point light shadows - the game makes heavy use of a very small number of point lights to light the dynamically generated areas, and without shadows being cast from these lights the game would look much lower quality. The engine offers the best trade-off between familiarity, power and ease of use for solo developers.

3.2.2. 3D Modeling & Animation

When it came to choosing an option for developing and animating 3D models, Blender was really the only choice. It is free, open-source software, thoroughly proven through professional and indie use, and contains a powerful suite of capabilities from designing simple 3D models, to articulating and assigning skeletons to models, to animating and decorating entire scenes for use in movie projects. Additionally, and most importantly, Unity supports importing direct models from Blender along with the animations, meaning the choice is available to animate in Blender's more comprehensive animation suite, or make use of Unity's animation suite which can be tied directly into the code for more gameplay-driven animations, or even procedural animations[3].

The following figures display various aspects of the Blender software in action. Of particular interest to this project, alongside the prime ability to design 3D models, is the animation suite - the ability rig, with inverse kinematics, the limbs of the player, as well as frame-by-frame control over the movement of attached items and automation capabilities. The suite is comprehensive, relatively simple to control with a combination of frame recording and manual transformations, and free to use.

Alternatives include Maya, an incredibly powerful solution which has the slight downside of costing £2,000 a year, Cinema 4D, which costs £661 a year, or Houdini - a film-quality VFX solution which provides tools for 3D modelling. Unfortunately, those tools are somewhat oblique to use, and many artists prefer to create models in another package before importing them to Houdini for the final VFX additions. It strictly utilises procedural generation technology, designed to allow artists to quickly work through prototypes and provide many procedural objects, but without a lot of extra effort over that required by traditional workflows, this can provide unneeded designs when a very specific design is in mind.

3.2.3. Programming IDE & Language

Unity provides built in support for Visual Studio. The support offered by Visual Studio includes expansive support for working in C# alongside IntelliSense features, which is a powerful suite of code editing assistance. Additionally, the built-in debugging features are expanded to support interaction with the Unity editor, boosting its basic functionality. Finally, official support for Unity-specific features (such as MonoBehaviour inheritance and related methods and classes) is built-in. Alternative IDEs exist but come down largely to personal preference and do not include the official support of Visual Studio. Some time was spent using Eclipse and IntelliJ IDEA to gauge their suitability, but neither offered any edge over its competitors.

Down to personal preference of the IntelliSense feature over alternative features in similar IDEs, and the official integration with Unity, Visual Studio was chosen as the IDE to use for this project.

There is more discussion to be held over the programming language used. C# is the primary language of Unity, but the backend is written in C++, and it does include hooks to access this backend. C++ is theoretically a faster, more efficient language for video games, with the trade-off of being more difficult to use, and specifically in this case, Visual Studio offering no C++ integration with Unity, and many helpful built-in Unity classes being unavailable to use. Additionally, with the C++ plugins requiring a call via C#, many repeated calls would begin to cause lag as the delay between calling C++ through C# is substantial.

Therefore, the language chosen to develop the game in is C#.

3.3. Code Storage and Versioning

There are multiple options for software to store code online. The most widely known of these is GitHub - robust, popular, and well documented. Alternatives include GitLab, which is much the same thing but with a different UI and open source, and AWS CodeCommit, a private solution built with businesses in mind, among others.

GitHub was chosen due to its relative ease of use, industry wide acceptance and being free. It provides the ability to host online backups of the project, which can be reverted to at any time in the event that it is necessary, and has an open-source but officially supported extension to add support for GitHub to the Unity editor, streamlining the backup and versioning process.

3.4. Graphical Editor

Krita is the chosen tool for 2D graphical design. There is very limited graphical design necessitated with the project - some UI assets and assets to be used in shader and VFX creation. Krita is free, and provides an extremely smooth and easy to use interface over alternatives such as GIMP and Photoshop, along with some useful features such as a wrap-around mode for displaying duplicate tiles on all sides of the edited tile, allowing the creation of seamless graphics which need to be repeated.

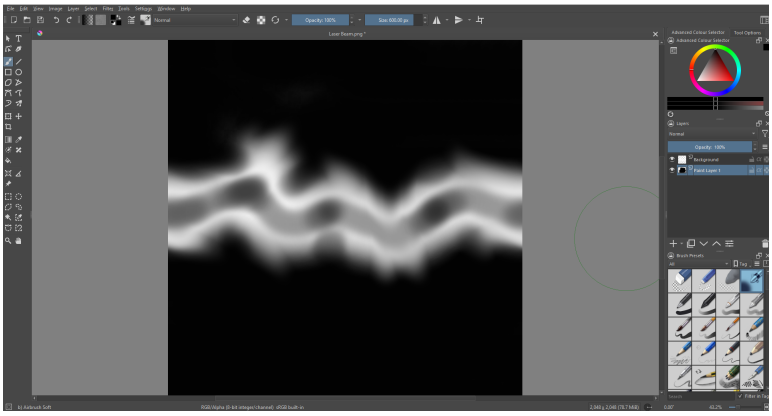


Figure 11: The Krita interface, with the tile used as the basis for the railgun shot shader

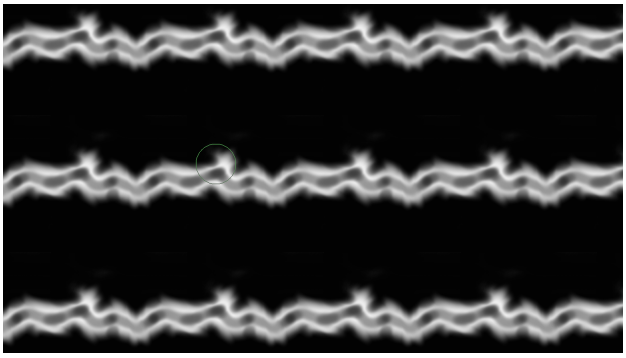


Figure 12: The railgun shot texture utilising the wrap-around mode to ensure a seamless texture

3.5. Problem-Solving Research

3.5.1. Procedural Generation

The largest technological issue is that of procedural generation. It is an intensely difficult subject to achieve, especially when you need to procedurally generate something with a particular use-case. Generating terrain for an open-world style of game is one approach, utilising voxel-based generation with Perlin noise[5] to create realistic flat terrain, before applying white-black Perlin noise heightmaps to generate interesting features. However, for this game type, voxel generation would not be particularly applicable as we have no need to generate infinite 3 dimensional maps that can be endlessly explored, as they introduce major gameplay issues – how does the player know where to go, how do we create interesting map features to facilitate combat, and what is the end goal?

The next method of linear procedural generation looked at was cellular automata with marching cubes[19]. This method is utilised to generate natural looking caves, allowing us to place limits on the sizes of caves and to what extent each cave must be connected, creating a series of larger cave rooms with connecting hallways. This method is useful for creating truly endless procedural terrain in a linear fashion, resolving the issue of the player having no concept of direction and no end point, as well as allowing us to generate premade structures, cover and other gameplay elements in the handily stored arrays of square data. However, this method has a new problem – the generation is directionless, there is no guarantee that any generated space will be enjoyable or fit for purpose, and the only form of generation achievable is of natural caverns, which does not fit with my chosen science-fiction aesthetic. A strong and useful method, perhaps, for generating terrain between levels of the dungeon, or being mixed with the voxel-Perlin based terrain generation from before to create a seamless planet-dungeon scenario, though both methods would be very difficult to implement as a bonus on top of the actual dungeon generation and the rest of the game features.

The next method is one which has been utilised since the beginning of 3D video game creation – handmade levels. This misses the key point of procedural generation, but it does have very strong pros for its case and can be used for the next method. Handcrafting every level and asset allows you to inject a high level of character into the level, craft environmental storytelling, and ensure that every level is perfectly fit for the intended purpose. This method is used to high effect in arena shooter games such as Call of Duty, which has a history of finely crafted maps for multiplayer fighting, or Doom Eternal, where rooms are created to fulfill the purpose of puzzle solving, wave defence or aggressive arena combat.

The final explored method utilises concepts from procedural generation and handcrafted design. The high level overview is that components of a level are handcrafted, but kept in isolation – a selection of a dozen rooms and room components (a main hallway may be supplemented by various wall designs, some including balconies or windows or various greeble), connectors (I, T, L hallways), and set-dressing (barricades, environmental storytelling, room features, enemy spawn points) are created which are then procedurally combined according to an algorithm which tries to remove too much accidental repetition or impossible geometry via a marching cube algorithm. This allows a series of fit-for-purpose areas to be strung together infinitely, utilising the marching cubes to ensure that geometry is possible and non-conflicting and to allow the loading and unloading of unnecessary parts of the map, maintaining performance. This algorithm can be run all at once or on the fly – analytics will need to be carried out to see which fits the intended use best. This methodology carries the additional benefit of navigation meshes being pre-made, so they do not need to be generated at runtime, resulting in lower performance or longer loading times and the

potential for bugs.

3.5.2. Hostile Artificial Intelligence

The A* algorithm is commonly used across most video games as a straightforward way to implement an efficient means of pathfinding for AI, and there is no reason to avoid using it in this project – especially as an A* solution from a previous project can be imported with minimal changes required. This can utilise the Unity Navmesh generation to create a grid compatible with the A* algorithm on any solid horizontal surfaces selected. This removes many of the issues with allowing AI to navigate the play space and makes sense as the optimal solution.

A secondary use for the A* algorithm is to utilise it in state machine decision making, as shown off in the game F.E.A.R.[21] By generating a matrix of nodes with various weightings pulled from the environment, the A* navigation algorithm can be used to identify not just the next best action for the AI, but the next series of animations which will best suit the current situation - with this series capable of changing on the fly in a performant manner. This has the additional benefit of the same system being transferable to any AI, regardless of motive - any decisions resulting in an attack will not be used by an AI with no method of attack, while AI with very offensive capabilities may find attack or aggressive actions being taken more often. As a final point, adding new nodes with new actions to the action matrix would be relatively simple - compared to a Finite State Machine (FSM), which may require large amounts of refactoring to include late on in the project.

The FSM is a very common method of AI decision navigation which has been used since the early 1990s. It is a machine capable of being in one specific state from a finite set of states, which can be navigated based on specific inputs, as per the following example:

State	Description
S0	Switched Off
S1	On and Moving Forward
S2	On and Moving Backward
Input	**Description**
Button push	On/Off switch
Sensor	Collision sensor

Table 1: State machine table[28]

One benefit of this system is the relative ease of implementation. Each action has a set number of interactions and state traversals, allowing each interaction to be individually curated. The downside of this is the difficulty in adding new actions later in development, the loss of ability to hand off much of the decision making to the algorithm, and the incredibly complex State Machine implementations which can result from more complex AI - taking Halo 2 as an example, the developers had an extremely hard time implementing the ability for AI to react not only to its own surroundings, but to its allies and the wider battlefield, resulting in hundreds of states and traversals.

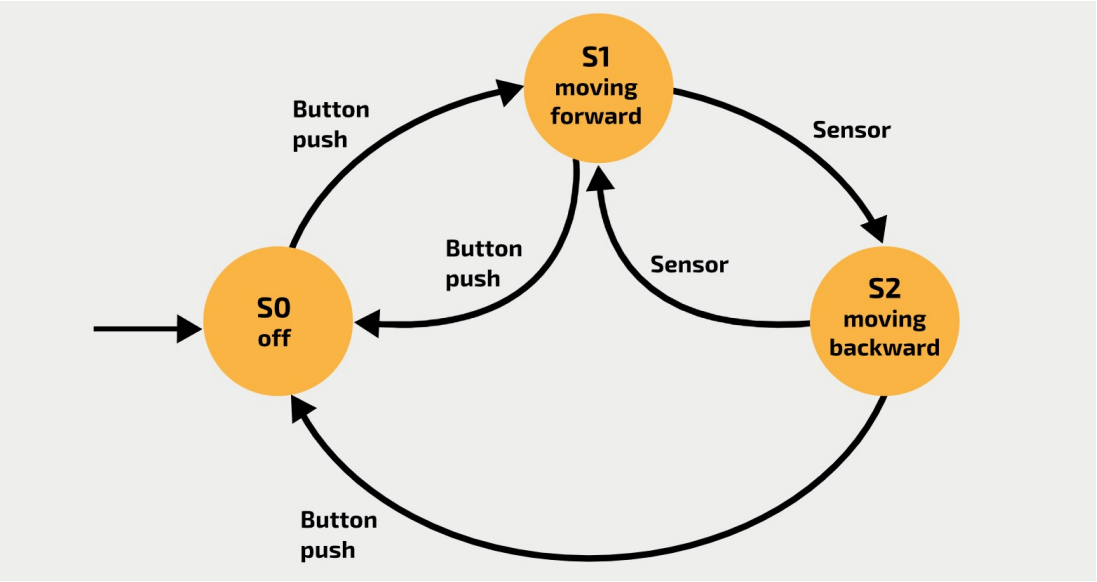


Figure 13: State Transition Diagram[28]

3.5.3. Unity NavMesh System

A built-in solution to AI pathfinding is found in the Unity NavMesh module. This allows a navigation mesh to be generated using all navigable terrain in a scene, which an 'agent' can then path-find across given a destination. The upsides to this method are clear - no need to implement a pathfinding system, as one exists which is already highly optimised, and a simple button click to generate a navigation mesh over all terrain. As a bonus, Unity uses the A* algorithm discussed above, meaning that basic knowledge needed for internal reworking of the system where necessary is already possessed. However, this method carries downsides - without building the pathfinding algorithms personally, it is harder to understand the internals of any system. Additionally, the Unity API for interacting with this system is limited - relying on quite basic commands which, without optimisation, could prove to be extremely hardware intensive.

3.5.4. Graphical Algorithms

Due to a lack of artistic ability available for integration in this project, the decision was taken to attempt to rely on algorithmic based graphical processing to achieve a distinctive aesthetic. To this end, three processing effects have been identified and researched to some depth: volumetric effects based in fog, clouds etc.[12], atmospheric scattering[18][16] with raymarching[31] to create unique colour combinations which change dependent on the camera alignment, and post-processing shaders[22][15] with particle effects.

Volumetric effects are an easy way to create an engaging world which looks a lot more immersive compared to flat camera effects – however, they can be extremely complex to implement in a performant manner. One way in which this can be taken care of is simply limiting the use of volumetric effects to small areas, unlike many recent games which use volumetric effects to render planetary cloud patterns, e.g., Microsoft Flight Simulator, Red Dead Redemption 2 and Star Citizen. Utilising Worley noise patterns to create the classic, fluffy cloud shapes we are used to, in combination with light scattering theory implemented to the render system, simple and easily defined random cloud patterns can be created in any shape desired, allowing flexibility for use in many scenarios, with an example shown in fig 14.

Figure 14: Computer generated procedural clouds.

The next relevant effect is atmospheric scattering – an interesting set of principles by which the natural light from the sun is scattered as it passes through the atmosphere, giving us a gradient from the oranges of sunset and sunrise on the horizon, passing up through a multitude of colours until reaching either bright blue or black directly above us – or, using the power of post-processing to modify the generated colours, we can create any interesting combination we desire. This general effect can be modified to work on a flat plane instead of a sphere, or to come from many sources of light – used to create, for instance, an interesting synthwave effect generated from lightbulbs in a room. This effect has no direct use case like volumetric effects but could be implemented post level design to create nuanced lighting effects for the level unlike other games which rely on simple baked lighting.

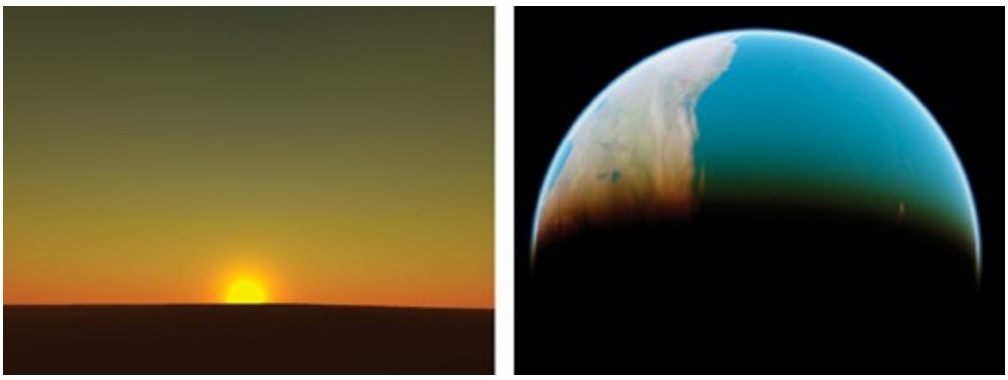
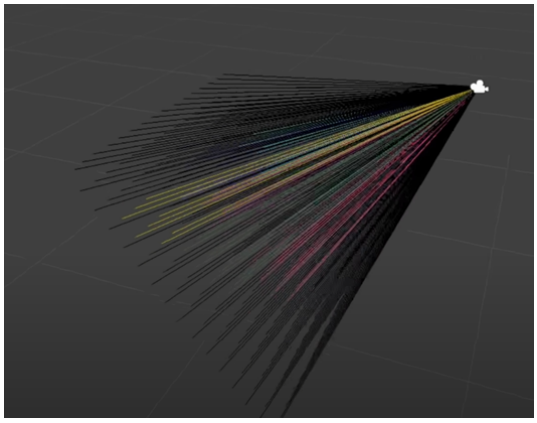


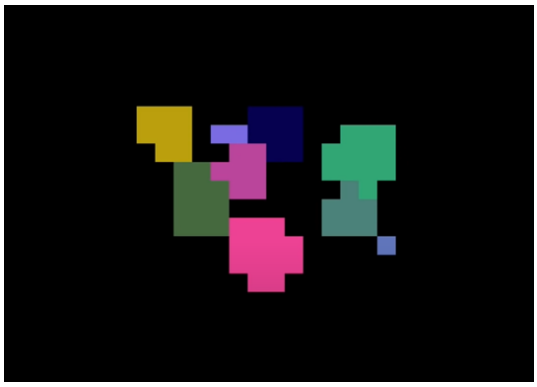
Figure 15: Depiction of atmospheric scattering from the ground and from space.

The last relevant effects shaders and ray-tracing. Unity provides built in systems for shaders and visual effects, in the form of the Shader and VFX graphs, but custom shaders must be created for effects such as ray-tracing and 3D portals, or even for computational reasons when necessary to remove CPU strain. For instance, terrain generation benefits massively from being run off a compute shader on the GPU where possible, although from there, interaction with CPU based scripts will be highly nonperforming. It is also important to note here that ray-tracing does not necessarily equate to the current popular use of ray-tracing inspired by NVidia[17], meaning realistic, real time, AI enhanced lighting – instead it simply means the graphical representation of traced rays drawn from the camera, which can result in many interesting results – reflections, custom rendering shaders, erosion techniques and more. Additionally, the core systems this tech-

nology is built in is utilised in many gameplay centric ways - casting a ray from a point in a given direction, and being able to return information on objects the ray has intersected with, is a core component in the creation of 'hit-scan' (meaning a weapon with no physical projectile, the hit point is instantly hit) weapons, detecting what the player is looking at, or determining whether there are objects between two points, useful for AI detection systems.



(a) Raycasts.



(b) On-screen representation.

Figure 16: (a) shows an exterior view of a small number of rays detecting coloured spheres, while (b) shows the graphical representation of these casts.

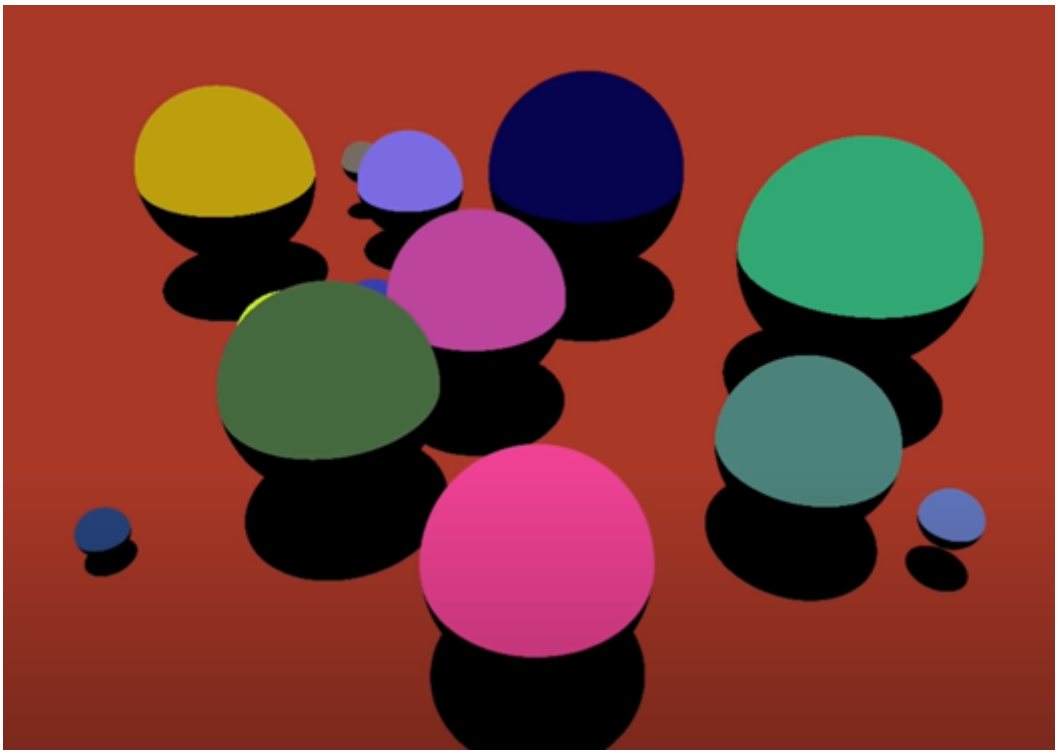


Figure 17: The same image with many more rays applied for higher fidelity, being used to create a very simple graphical representation of 3D space.

3.6. Review

The technology discussed will not have a direct impact on the second-to-second gameplay of the product but could have been utilised to bring greater immersion and a more enjoyable experience.

Additionally, this research is more of an exploration into current day techniques used to generate realistic and interactive video imagery and may not be possible or applicable for implementation into this project - regardless, an understanding of how these results are achieved can benefit any project, and some components may be utilised in other areas. Ray-tracing, for example, will not be implemented in the graphical form, as it takes a huge amount of time and effort to implement in any functional manner - instead, the basic form of ray-casting will be used to provide functionality to the game, in the form of the player weapons, AI detection systems, and more.

4. Component Breakdown

The components have been broken down into the tables seen in figures 18 and 19. Alongside this table is a UML Diagram for the enemy subset of classes, shown in fig 20, as they all inherit from one base class. The state of completion as of the end of this project is listed next to each component.

Component					
In-Game	Finished	Management	Finished	UI	Finished
Player Controller		Game Controller		Main Menu	
- Horizontal Movement		- Initialise game on start		- Button to start game	
- Aiming		- Pause game		- Display options screen	
- Jumping		- Unpause game		- Quit game	
- Step/Slope Navigation		- End game (death)		Pause Menu	
- Buffered Jumping		- End game (victory)		- Resume game option	
- Coyote Time		- Change to next room		- Display options screen	
- Apex Boost		Statistics Controller		- Give up on current attempt	
- Clamped Gravity		- Hold all player statistics		- Quit game	
- Jump Interrupt		- Hold all unlocks		In-Game UI	
Dash Controller		- Store currencies		- Display player health	
- Move to target location		- Save current data		- Display active enemy health	
- Default last direction if null		- Load saved data		- Display weapon state	
- Dash provides buff state		- Reset statistics to base		- Reticule (context based colou	
- Limit consecutive dashes		UI Controller		- Interaction prompt (updates)	
- Refill dashes		- Manage player input to UI		- Owned currencies	
- Dash provides invulnerability		- Provide global access to HUD		- Tutorial prompts (one time)	
Player Interaction		Room Generator		Options Panel	
- World-Space buttons		- Generate new room		- Display controls	
- Ability for player to interact		Room Controller		- Allow changes to controls	
- Doors		- Choose and spawn enemies		- Sensitivity slider	
- Teleporter		- Track enemies for room end		- Master volume slider	
- Shop kiosk		- Spawn the rewards		- SFX volume slider	
- Shop UI Display integration		- Setup the room over state		- Music volume slider	
- Map Selection		Enemy Controller		- Change resolution	
- Upgrade Drop		- Store all enemy prefabs		- Change window mode	
Enemies		- Store enemy spawn details		- Toggle screen shake	
		Menu Controller(s)		- Toggle accessibility option	
		- Handle swapping panels		- Save and load preferences	
		Save Management			
		- Save and load game state			
		- Handle exceptions (first time			

Figure 18: Breakdown of components. Green indicates finished completely, yellow indicates some issue and red means uncompleted.

Elaborating on the marked issues on the breakdown tables: the enemies category is finalised aside from the originally intended Holder enemy type, which was not included, as explained later. Similarly, the Holder component under 3D Models & Visuals is marked as uncompleted. The Hub Ship component was implemented through the use of a 3rd party asset, while the entry and exit rooms were completed but to a different specification than indicated in the design documents. Both are explained in the Implementation Section (6.4 and 6.7) and the Development Issues Section (7.1 and 7.2).

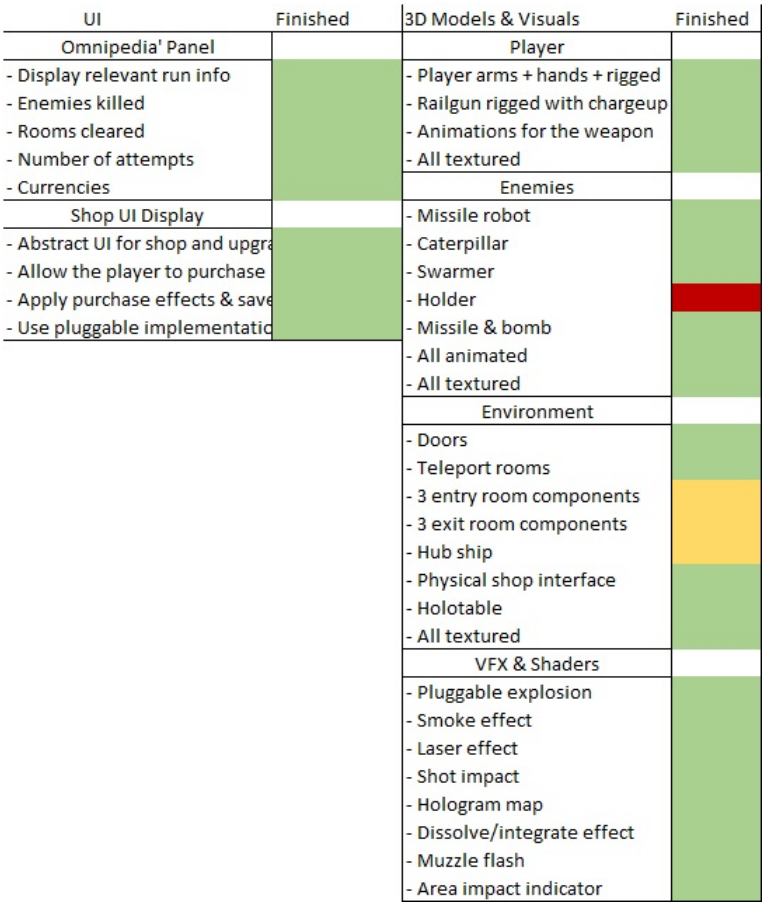


Figure 19: Continued breakdown of components. Green indicates finished completely, yellow indicates some issue and red means uncompleted.

5. Software Lifecycle

5.1. Lifecycle Model

The lifecycle model utilised for this project is Iterative development, alongside MVP development. Basic architecture for some components must be in place before development can continue, so the Iterative process allows the project to develop the necessary architecture early on. By establishing a Minimum Viable Product (MVP) early on, this basic architecture can be iteratively built upon to quickly develop the various aspects of the game with periods for retrospection on development. Additionally, further mechanics that are not core to the product MVP can themselves be created in MVP fashion, with quick and dirty development used to get them functional before time is allocated later to polish up the feature to a release-ready standard.

Iterative development has been known to exceed its scope heavily, using unplanned resources and time. By planning thoroughly and adhering to the design document, this can be avoided. MVP style development can result in a product where the final vision is hard to see - even close to the deadline, some feature can still be in an MVP state alongside the release ready features, and this disparity may make cohesive design difficult. As before, extra time spent in the planning phase to really hammer out designs and feature descriptions negate this downside.

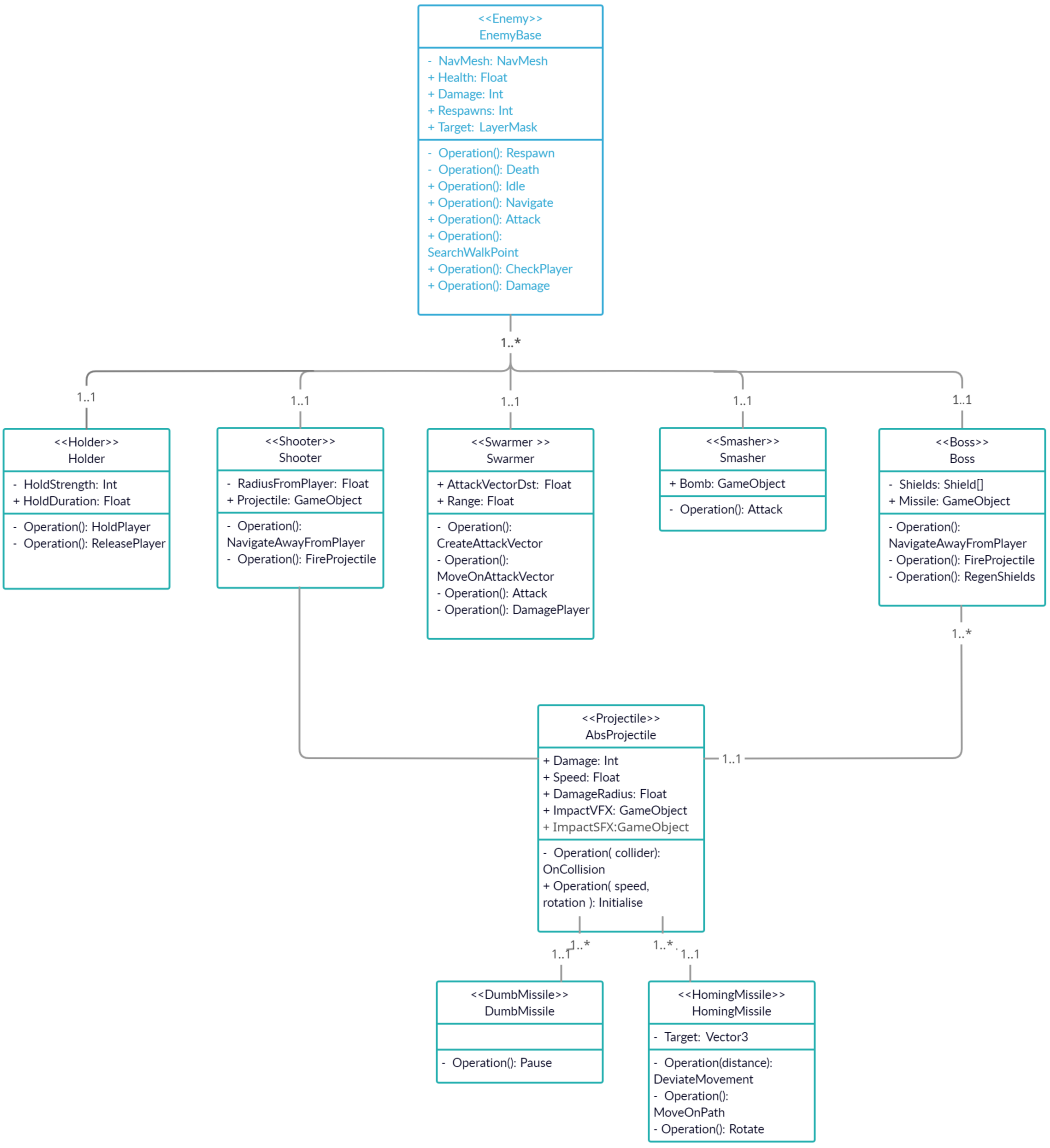


Figure 20: UML Diagram for the Enemy base class and all inheritors, plus the spawned projectiles.

5.2. Analysis & Planning

Analysis of existing material in the form of published video games and research into various technologies has been compiled in the Background Research section. Additionally, definition of the scope, requirements, purpose and boundaries of the project are defined in the Design Document. Following is the System Requirements Specification, based on analysis of the Components Breakdown - itself available in the previous Component Breakdown section.

5.3. Software Requirements

5.3.1. Functional Requirements

- Character capable of 3D space navigation.
- Rooms which can be generated and destroyed on command.

- AI which can perform independently in generated environments.
- Shops where upgrades can be exchanged for money gained in rooms.
- Game structure whereby a string of rooms is generated, and death or success returns the player to the homebase.

5.3.2. External Interface Requirements

- Main menu with access to the game start, options and quit functions.
- Options menu allowing changes to sensitivity, volumes, resolution, windowed mode, screen shake and God mode.
- In-game UI which displays weapon information and aim reticle, player and active enemy health bars and currency information.
- Pause menu with access to resume, options, 'omnipedia' and quit functions.
- 'Omnipedia' which displays game statistics and states, such as kills, attempt number and room number.

5.3.3. Non-Functional Requirements

- Game must run at 30fps on a PC with minimum hardware: i7-4702HQ, 8GB RAM, NVIDIA Quadro K1100M 2GB.
- Basic Game Accessibility Guidelines (GAG) must be implemented, and Intermediate GAG implementation should be aimed for - as per the Game Accessibility Guidelines group, formed of independent studios, specialists and academics involved in the disability and general software accessibility sphere[7].
- A high degree of visual and functional quality must be achieved - difficult to categorise.
- No internet connection so there are no security vulnerabilities related to internet usage.

5.3.4. Timekeeping & Task Management

Some target goals are structured by the University year and module outline. For example, completing the MVP by Christmas would provide a solid foundation for development up until the final deadline of 25th April. In this case, the MVP would constitute basic gameplay fulfilling the defined gameplay loop, with unsophisticated graphics and minimal bug-fixing - a minimum product which can be shown off. Additionally, Gannt charts have been created for the periods before and after Christmas: with the 2021 period being more focused on programming development it has need for a more rigorous structure, while the 2022 period being left to largely graphical and animation work alongside bug fixing has far fewer structured jobs requiring other work to be completed prior. Due to this, more jobs are performed in parallel. The charts are displayed in figures 21 and 22.

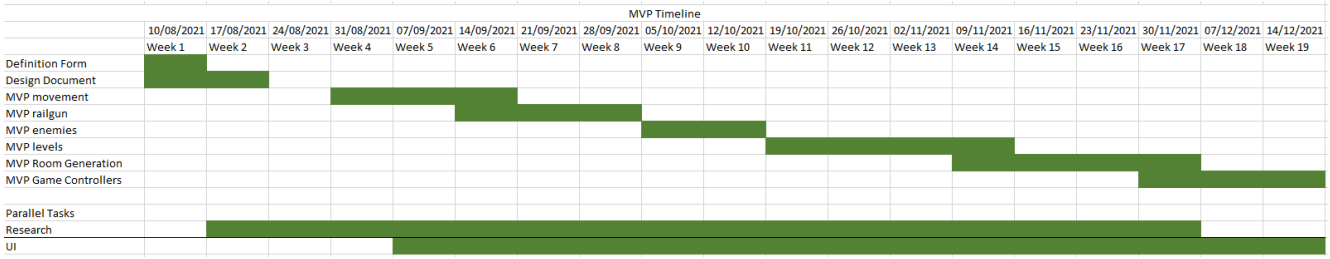


Figure 21: Time plan / Gannt Chart for the MVP section of the project - before Christmas.

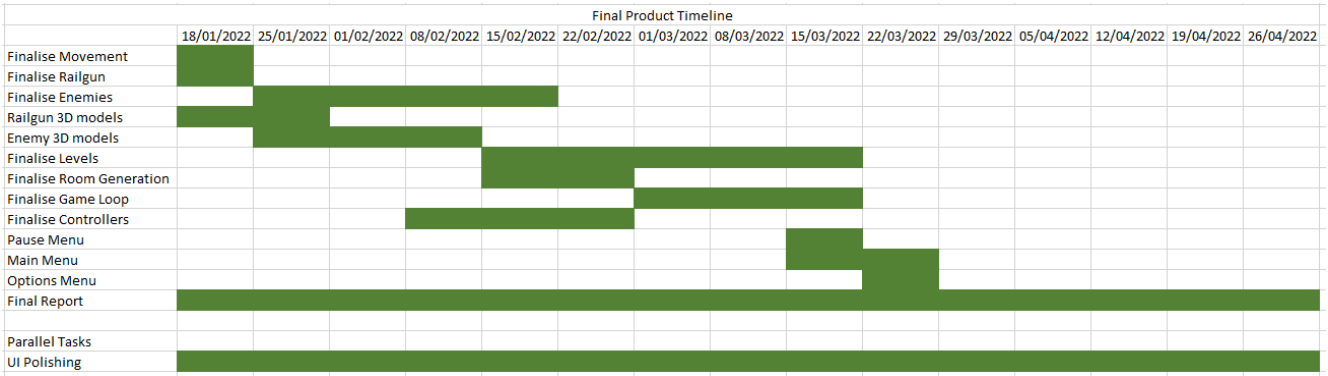


Figure 22: Time plan / Gannt Chart for the final half of the project.

Trello was used as a task management system, to split the tasks into defined components and keep track of what has been completed and when. This system allowed the project to proceed smoothly, without needing to check over previously completed tasks, and allowed work to remain within the defined constraints of the project. A screenshot of the Trello board is seen in figure 23.

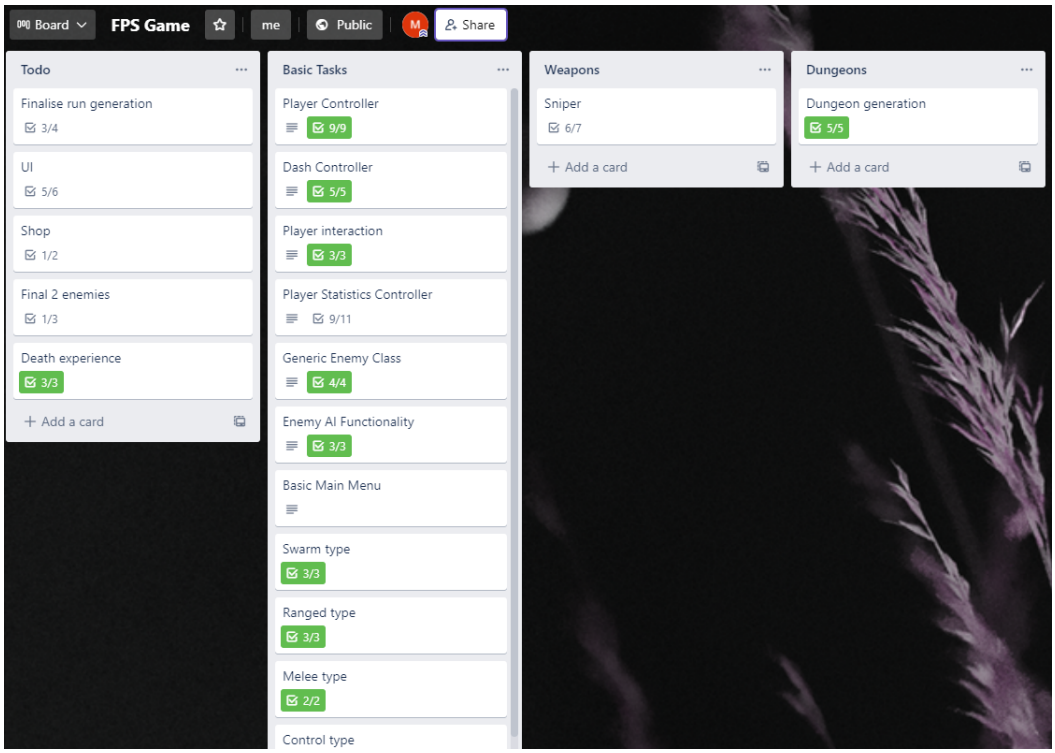


Figure 23: Trello board in a partial state of completion, as of 20th Feb 2022

5.3.5. Feedback

With the lack of specific client or business, client feedback sessions and a business plan are not required. Feedback will instead be attained through usability testing later on into development, where feedback should be more relevant.

5.4. Design

5.4.1. Architecture

There are very few concrete industry practices in the game development sphere. Many companies utilise their own technologies and working practices, meaning that there is very little to inform practices on this project. However, programming generic practices still apply; code is well commented, variables are informative instead of short, and camelCase is utilised. Additionally, versioning standards are adhered to; 1.0.0 for major changes and releases, such as entering the MVP and the final state. 0.1.0 will be used for large changes such as the implementation of new mechanics or overhauls, and 0.0.1 will be for any changes which do not fulfil these criteria - any change substantial enough to need backing up.

Event driven code and controller-based designs are used to power the project. Event driven principles dictate that code will be triggered by events instead of constantly listening to events, which helps to reduce the performance costs of the many interactive systems found in a game. A prime example of this is the UI, which, by utilising the in-built Unity UI Event System, allows the easy production of highly performant UI systems. An implemented example is found in the Game Controller classes - the many events controlling the game, such as the pausing, end of level or end of game events, are triggered from the source instead of passively listening for conditions to be met. In the Game Object focused architecture of Unity this can massively reduce performance costs, without needing to probe a potentially huge list of Objects.

The controller-based design hierarchy, utilising the Singleton structure, with the version used in this project shown in figure 24 is commonly used in the game development industry. This design forces methods to be kept strictly to related controller classes - the Game Controller, for example, controls the pausing, state of game, tracks enemies and progress, etc., while the Player Movement Controller specifically controls the movement, jumping and aiming of the player. By making these classes static, and ensuring there are no duplicates, events can be triggered from any scripts easily and with high performance, as well as ensuring that there is no duplicate code.

As shown in figure 6, the master class for this project's hierarchy is the Statistics Controller. This controller contains methods for saving and loading game data, as well as storing values which must be commonly changed at runtime, allowing the controllers below it to retrieve data as needed from a common, structured source. Below this, the Player Controller heads the related player classes, which each require access to the statistics controller and each other. The Game Controller is responsible for the current state of the game, and facilitates interaction between unrelated classes - for e.g., the player class responsible for detecting the pause key will trigger the pause method in the Game Controller, which will then trigger the display of the pause UI in the UI Controller. The Enemy Manager is responsible for the generation and tracking of enemies in the active game space and provides an interaction layer between the abstract enemy classes and the Game Controller. The Room Controller tracks the current room, state of progression, and triggers generation of a

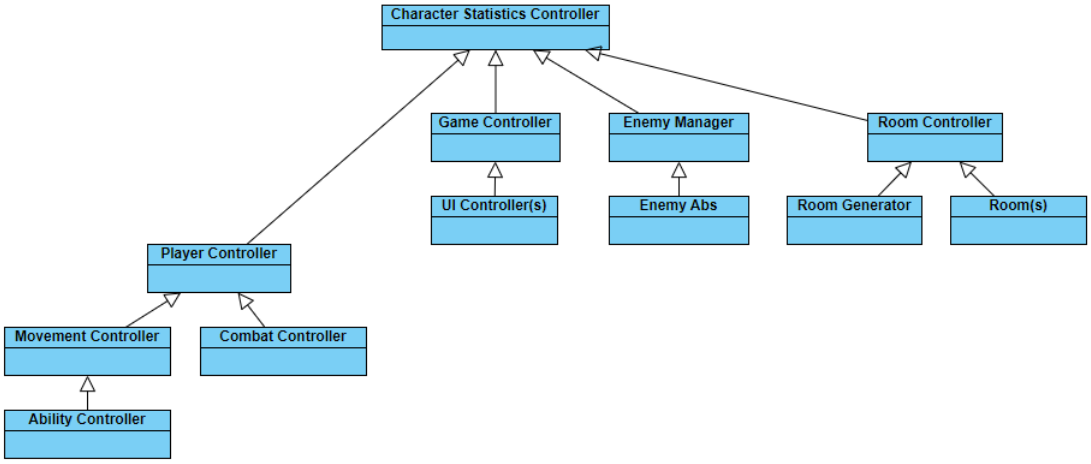


Figure 24: Controller Hierarchy

new room and its related components. It also facilitates interaction with individual room Objects through a series of arrays.

5.4.2. User Interface Design

The Unity UI Event System manages much of the interaction with UI layers without the need for additional code. This system monitors the defined interaction keys - mouse click or entry, or controller input, and triggers assigned methods, giving the UI practically no performance impact.

The design of the User Interface required multiple sketches to determine the optimal layout. These are shown in the following figures, alongside the chosen designs. Considerations taken into account during the design process were the need for relevant information to be displayed in prominent positions without obscuring the game space, natural navigation between elements in order of importance, and aesthetically pleasing layouts and elements. Hand-drawn concepts and computer-generated concepts are available in figures 25, 26 and 27.

This concepting phase provided a variety of options for the placement and designs of various UI elements. By placing the pause / main menu buttons on a black bar against the wider background, the eye is naturally drawn to them - and with western cultures, the main audience for this game, reading from the left to right and downwards[20][24], the buttons are placed in hierarchy of importance either top-down or left-right. The start / resume options take the highest priority, as the most important option in a game is the ability to start it. The options menu takes second priority as it is the second most likely option a player will want to explore, while any extra buttons can fill the space to the quit button - which comes last both as a form of convenience (most video games have the quit button at the bottom of any list), and to de-prioritise it in the eyes of the player.

5.4.3. Programming

The first global issue when designing any game in an engine reliant on Object architecture is interaction between different objects. Scripts are placed on the Game Object in the editor and need to be either manually linked (time-intensive) or discovered at runtime (exponentially computationally expensive with scene size). An alternate option is that of static classes or properties

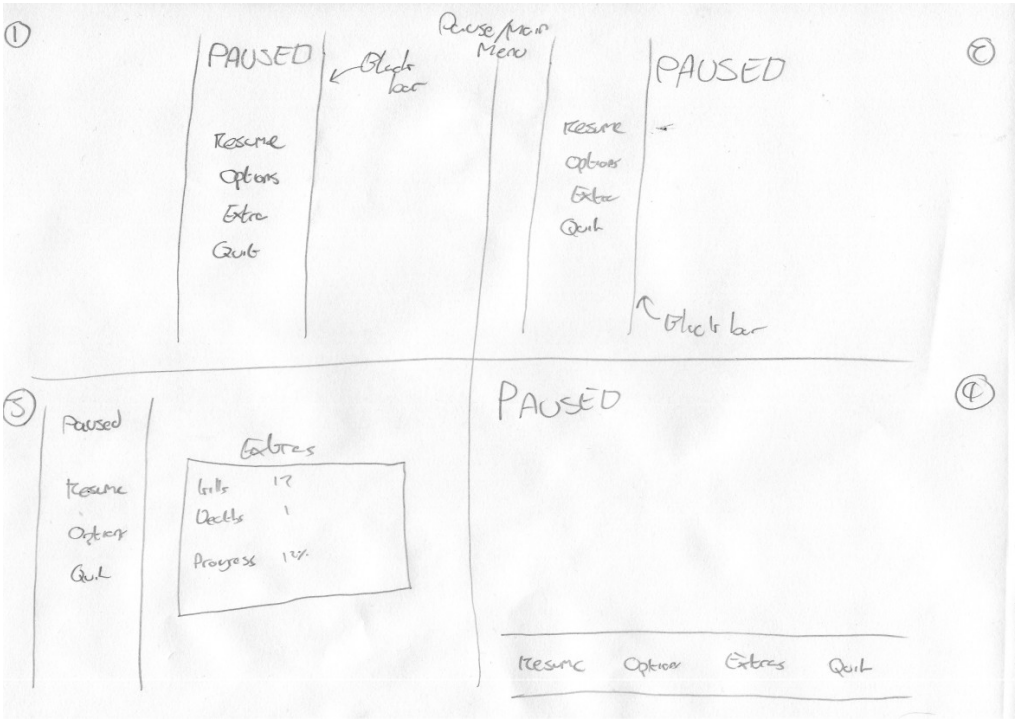


Figure 25: Concept pause menu designs sketched on paper

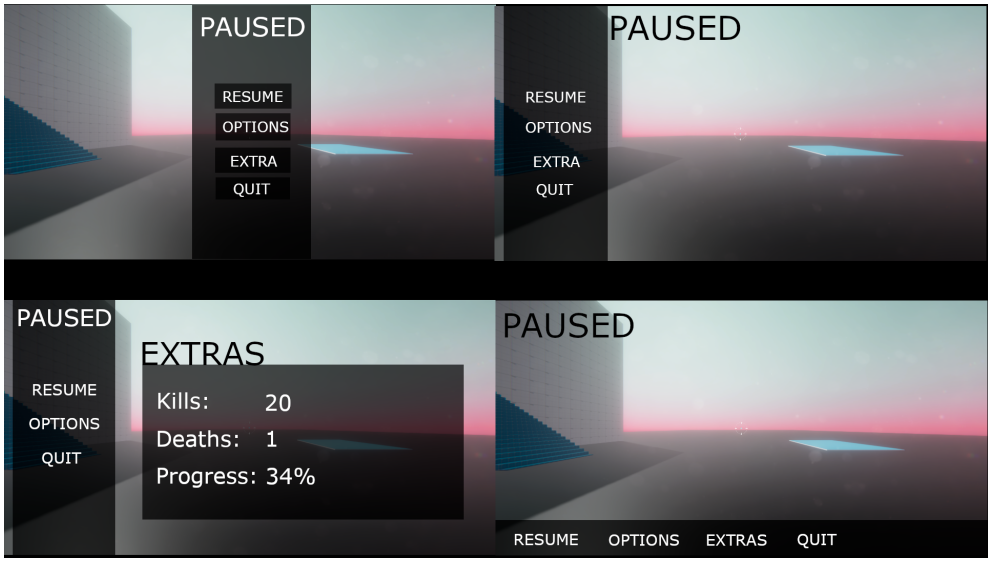


Figure 26: Concept pause menu designs roughly created in GIMP

- while only a single instance of a static class or property can exist, making it useless for tasks such as AI control, where there will be multiple instances of the AI with differing variables, it is extremely useful for controller classes. Assigning the Game Controller to its own static variable, for example, means that the Game Object the class is attached to can be accessed simply through 'GameController.instance.Method();'. This method is also unique to each scene - objects do not persist, meaning there is no possibility of conflicting methods or variables carrying over. One example where this is not appropriate is the Statistics Controller - the class itself must persist between scenes, and so does not need to be assigned to a static variable or a Game Object; just being a static class accessible anywhere, anytime, is perfect for its job.

The second global issue is that of repetitive code. Many components in a game will carry out similar roles in a similar manner - for example, AI behaviour. In the case of this project, an

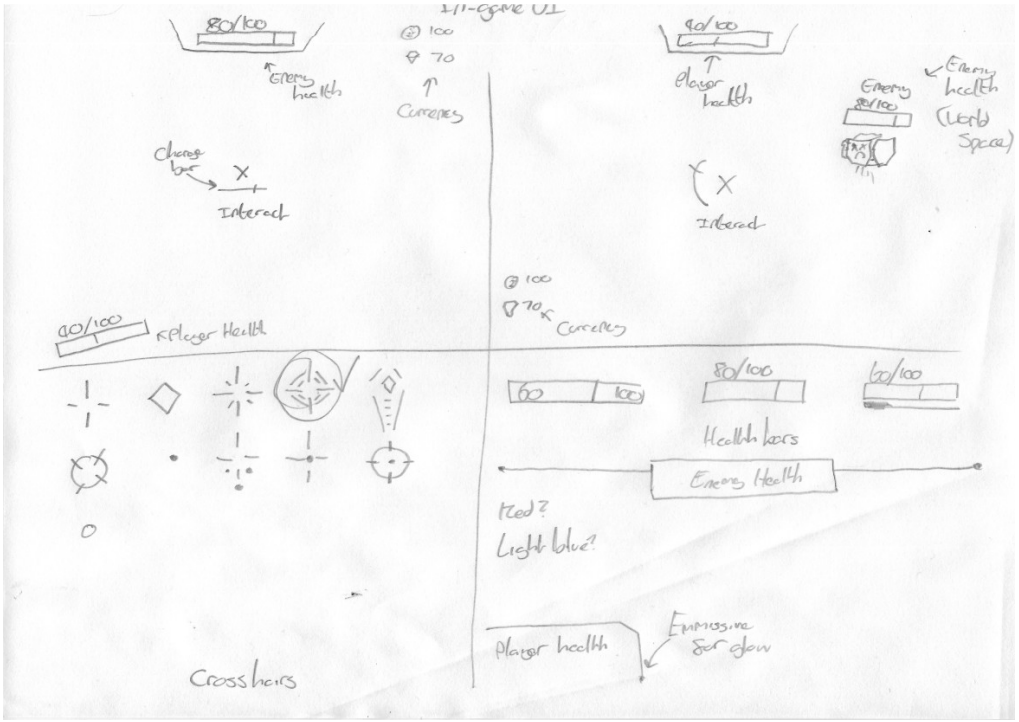


Figure 27: In-game UI layouts, crosshairs and healthbar / UI peripheral concept sketches

abstract class for the enemy archetype will provide basic shared functionality, such as idle and patrol behaviours alongside damage and death management. More specific classes will expand upon this functionality to, in the case of the Rocket Robot, keep the AI at a specified range from the player while launching projectiles. This philosophy carries over to the player weapons - though no additional weapons are planned, it makes sense to keep shared functionality, such as raycast projection and damage calculation, to an abstract archetype class.

Continuing AI discussion, the behavioural design for the AI constitutes a complex issue. Using a state-machine to operate decision-making provides the basic architecture to build behaviour from - with the added benefit that some of the state-machine will be replicated between AI due to the similar initial and idle states.

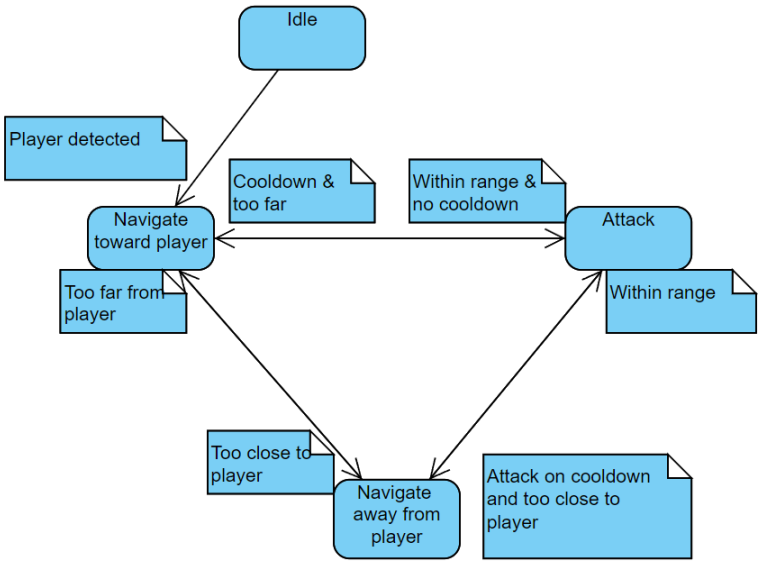


Figure 28: State-Machine for the 'Rocket Bot'.

```
0 references
void StateMachine()
{
    if (!hasSeenPlayer && !attacking)
    {
        // perform idle action
        return;
    }
    if (!playerInAttackRange && !attacking)
    {
        // navigate to attack range
        return;
    }
    if (attackOnCooldown)
    {
        // navigate to safe distance away from player
        return;
    }
    if (playerInAttackRange && playerInDangerRange)
    {
        // navigate to safe distance away from player
        return;
    }
    if (!attackOnCooldown && playerInAttackRange)
    {
        // perform attack action
        return;
    }
    // catch all in the event that an unexpected event or
    // combination of conditions arises:
    // navigate to safe distance away from player
}
```

Figure 29: State-Machine pseudo-code for the 'Rocket Bot'.

Resolving conditions for swapping states presents that most heinous of challenges - maths. Calculating relative positions in independent 3D space is a requirement for intelligent positioning, and so requires some basic understanding and implementation of vector-based mathematics. First up is determining the direction from v1 to v2: $v3 = v2 - v1$. This can be further combined with the position of the player and clamped to a range to return a random point on a circumference around the player: $v3 = v2 + clamp(Random.angle * (v1 - v2), clampRange)$. This equation will provide much of the player-relative positioning logic for the AI.

A third issue is the implementation of events. A commonly used method is to simply check for the trigger conditions of events every period of time. This can potentially reduce performance of the game dependent on the complexity of conditions, and results in difficult to read code. Instead, trigger driven events were chosen - instead of checking how many enemies are alive each frame, the death function on AI will send a trigger to the Game Controller which will reduce the initial enemy count by one, and perform end of room actions when no enemies remain.

One particularly difficult issue identified relates specifically to navigation meshes and baking light-maps. By default, Unity requires these to be generated in the editor and does not support runtime generation. An experimental resource[27] is provided allowing these to be generated, which has been adopted into the project, giving access to components defining NavMesh surfaces which can be saved into prefabs, a requirement for this project, with minimal identified issues.
The more difficult side of this issue is the generation of baked light-maps. By design, this project relies heavily on prefab objects - objects which have been saved, ready to be instantiated (cloning an object and returning that copy, while placing it in the game world) or created into a scene at runtime. Unity provides no capacity for saving light-maps onto prefabs, and very limited access and documentation into the lighting engine to facilitate independent development - instead, one of

the most popular assets on the Unity Asset Store is a third-party light baking solution[23]. This asset was already owned, and provides support for saving baked light-maps to prefabs, so it has also been adopted.

5.4.4. MVP

The MVP is a critical component of game development - establishing the basic code base and required components for functionality, while not dedicating enough time and polish that components cannot be changed if required. This project is no different - the MVP deadline was previously set before the Christmas break - 15th December. The MVP created fulfilled the planned requirements: the movement and aiming systems were established, the base functionality for the main weapon in the form of its charge up laser and spread attack were created, multiple exploratory methods of random generation were created in a modular fashion so each could be trialed, basic progression from room to room was made and a basic AI with idle and attack states was made. Additionally, some exploration into the visual style of the rooms was made with the placeholder assets, to represent a bit of the feeling that the final product would move on to have.

An additional variation on the MVP was used in this project. Individual features would be pushed to MVP level before further iteration upon the foundations - for example, the railgun was pushed to a point where the basic charge to shoot and shotgun functionalities existed, but there was no model, visual effects or animations.

The project-wide MVP status was achieved on 17th December 2021. Various key components (the railgun, basic enemy functionality, character movement and random generation) achieved MVP status before that date, while components not considered necessary for the MVP (such as visual effects, models, the continuation of the game loop to include currency and a repetitive pattern) were pushed to the MVP stage in 2022. For some components, there was very little difference between MVP and final state - the door model, for example, simply had lighting and materials applied after its MVP or 'greybox' stage. It is difficult to show evidence for the project-wide MVP state of the game - the image in figure 30 shows how the game looked at that stage, but going into details would take much more space.

5.5. Implementation

The implementation topic is too large to fit in this breakdown of the software lifecycle. Therefore, though it is mentioned here for clarity of its position in the lifecycle, the main body is found in section 6, titled Implementation.

5.6. Testing

5.7. Internal Testing

Much of the testing in pursuit of bug-fixing, suitability testing and playability testing was performed internally. There are many unavailable solutions here - for example, a team of playtesters who are able to look into features and attempt to break them in any way possible. Alternatively, machine learning is starting to be utilised by major gaming publishers such as EA[4], while Google

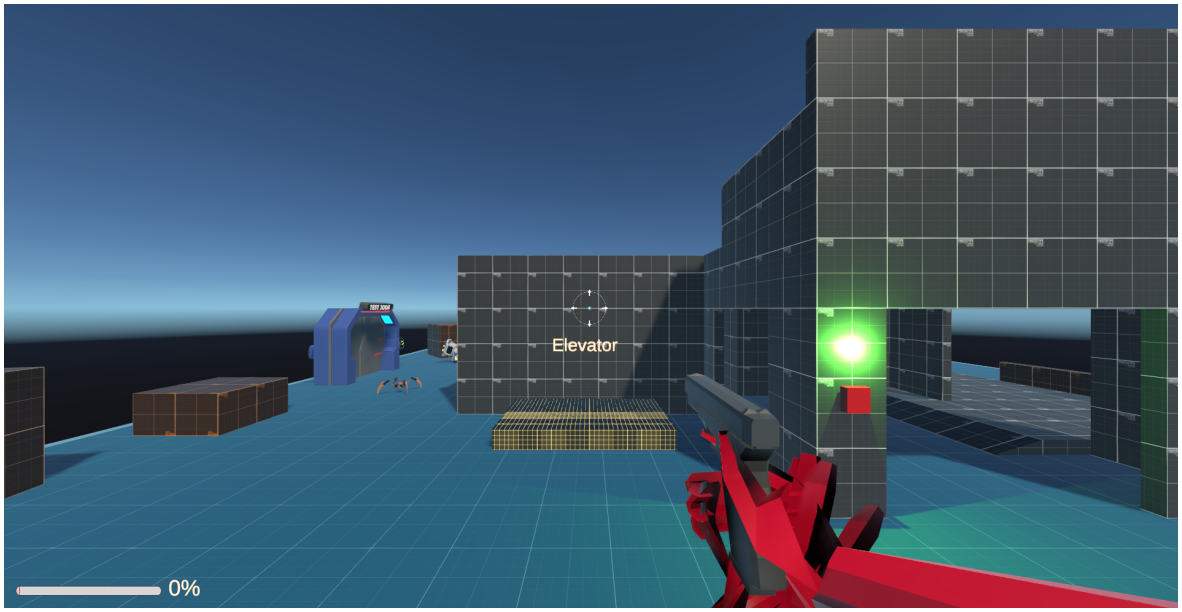


Figure 30: Image of part of the MVP - basic implementations of the railgun functionality with a stand-in model, and the Rocket Bot and Swarmer enemy types, alongside player navigation and controls.

has published frameworks and documentation on their own neutral solution to test a variety of games[13]. While this would be an interesting and highly valuable method of testing, it simply could not be utilised here due to the complexities of setting up the frameworks not falling within the given timeframe of the project. Utilising a team of testers would similarly be impossible, due to lack of funding to support this.

Instead, very basic testing needed to be implemented. Visual Studio provides testing and debugging integration with the Unity editor, allowing common debugging processes to be extended to the play mode editor. For example, breakpoints can be set which will then be activated while playing the game and stepped forward or backward as usual. For this project, the inbuilt debugging tools alongside exploratory was largely utilised - further into development, when the inter-connectivity between classes was far more complex, some unit testing was implemented, but it was mostly unnecessary. The nature of video games making many issues graphical, audio or physically based (holes in collider objects, NavMesh errors, obscure ways to circumnavigate game restrictions) makes unit testing totally irrelevant for many processes, while exploratory testing, with its focus on more abstract thought testing, learning, and investigation allows far more useful ground to be covered.

A form of maintenance and in-project documentation over testing, while still providing similar results in terms of reducing defects, is the creation of an 'interaction zoo'. Once a feature has been completed to a functional state, it should be added to a scene, correctly set up and labeled, so it functions as intended. This way, anyone else who takes over later on, or the original designer if it has been a while, can check back to this scene to discover what features are available which may fulfil some newly required criteria, or explore how specific functions should be correctly set up - or, in some cases where the function is capable of independent operation, directly copy that function. An example is doors: a game may have many different doors, opening horizontally, vertically, spinning or in the form of an elevator / teleporter. By placing each example in a scene, any developer can then walk around that scene as a player, explore which setup they like best for their door requirement, and then copy and paste that asset over to their own work, cutting down on time spent and providing working examples to reduce errors. Additionally, this scene can hold areas for testing new functions, such as a range for testing new guns, or the ability to spawn in

enemies within the game instead of having to place them into an existing level and hunt them down.

The interaction zoo combined with traditional debugging and testing methods provided a perfect release-environment simulation for testing new features and obscure interactions. Typically, bugs were very simple to resolve - unassigned variables or objects initialising in the wrong order. Relatively few were major bugs - incorrect mathematical equations for distance calculations provided some pain early into the AI implementation, and a reoccurring issue whereby the player would refuse to be moved to the starting position after loading in from the home-base, but would correctly move if starting directly from the dungeon scene proved especially annoying, but both were ultimately simple to resolve. Gameplay testing, to determine the 'balancing' of statistics between enemies and players to provide an enjoyable but challenging experience, identifying hang-ups or frame rate drops, or finding odd ways to break the game outside the boundary of incorrectly performing code all requires simple repetitive testing. While some was done for this project, it would take multiple testers a lengthy period of time to test every single combination of rooms, enemies and actions.

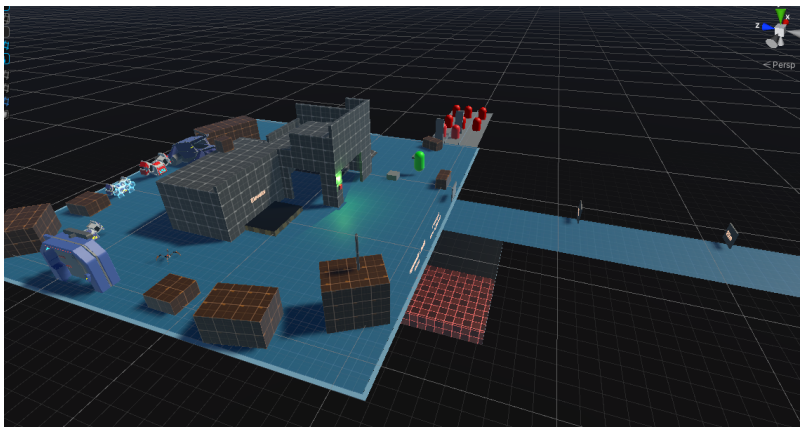


Figure 31: The Interaction Zoo - multiple elements combined into one area the player can navigate, and a developer can pull resources from.

In order to support rapid testing, the game was made to reset saved variables when a different version to its previously saved one was loaded up. This way, testing could be performed on the built game instead of in editor without the need for time consuming manual deletion of game saves or implementing a method in the game to do it on demand. Some issues found during this rapid testing phase, toward the end of the project, include the player not teleporting to new positions on specific hardware combinations, the shop allowing the player to incur crippling debt for no understandable reason, and the crab bots deciding to spend their AI lives hovering 2 meters off the floor, also for no understandable reason.

In addition to testing for issues and gameplay flow, performance metrics were taken on a variety of hardware combinations. The game was limited to run at 144 frames per second maximum, to reduce graphical jittering of the holographic grid on the levels, and achieved this 100% of the time on the developer computer consisting of an RTX 2070super, Ryzen 7 3700X and 32GB RAM. As would be expected, this benchmark was also achieved by a PC running an RTX 3070 with a Ryzen 7 2700X with the same RAM, though this hardware setup exhibited consistent issues with the player not moving correctly between points - it is theorised that this is due to the CPU being slower, the room generation was not finalised before attempting to teleport the player, as this work was performed asynchronously on virtual threads. A fix to this issue was made to ensure that the room generation and then player movement happened in chronological order. A third hardware setup was used in the form of an RTX 3090 with a Ryzen 7 3800X and the same RAM, displaying

none of these issues. Two lower end pieces of hardware was tested - a Dell M3800 laptop, with a Quadro K1100M, i7-4712HQ CPU and 16GB RAM. This laptop resulted in an average FPS of 87 and a 1% low FPS of 85 - meaning that the lowest 1% of the frames averaged at 85 FPS. The second was a laptop with a GTX 1070, i7-6820HK, and 16GB RAM, achieving 80FPS average with a 1% low of 72.

5.8. Usability Testing

Usability testing is an important component of any software project, especially one in which the usability metric is not simple, measurable functionality, but individual enjoyment and ease of use. To this end, a usability test was established and submitted for ethical approval, before being carried out. Due to the aforementioned implausibility of measuring specific metrics to judge a video game, judgement of the testing was not performed based on the completion of tasks, but rather a post-testing interview. Due to time restraints, participants would watch the game being played and provide real time instructions instead of playing themselves. While a downside in some regards, specifically in relation to the feeling of playing, it does remove the skill barrier provided by FPS games, providing a flat base by which participants could judge design, ease of navigation between the game systems and general gameplay mechanics. The sessions and interviews were recorded, stored on a local hard drive and an anonymous transcript created as soon as possible, before destruction of the recording. Four participants were chosen - two with a high level of knowledge of video games and the FPS genre, one suffering from Deuteranopia, who is additionally less proficient with video games, and a fourth who is familiar with games in general, but not FPS mechanics. All related documents to the Ethics clearance, and the transcripts of the recordings, are found in Appendix B. The recordings were lengthy - between 10 and 15 minutes each, and as such, the transcripts do not contain material which has proven irrelevant to this study - for example, the participant successfully completing tasks has simply been noted below, and the second-to-second conversation and directions would fill pages with no real benefit.

The first task participants were given was to start the game. All participants gave directions to start the game successfully, with two choosing to explore the options and controls menu before starting the game. There were no issues here of any kind.
The second task was to direct the use of all controls in the game. One participant opted to direct the controlling player using a controller, while the other three opted to use keyboard and mouse. In one case, the controls were not easily discovered, as the participant did not look at the controls menu at any time. Two other participants looked at the controls in the main menu, and the last participant looked at the controls after getting into the game proper and realising they could not direct many of the abilities of the player character.
The third task was to direct the completion of two levels. Each participant directed the gameplay in unique ways which resulted in different issues - one participant had no issues directing to the end of the two levels, and understood how to use the reward menu when it was opened. Another discovered that by quickly walking back into the entrance room after leaving, the door would lock and the player would be stuck. The last two directed the controlling player to stand underneath the exit elevator platform several times, and in one of these cases, the character was pushed through the floor. No other issues were encountered, and in each case, the participants understood how to progress the levels, how to deal with the AI, and one discovered and chose to use the god mode option, designed to make the game easier.

The specific questions asked in the interview section follows:

- How challenging did you find the tasks?
- What issues, if any, arose while starting the game?
- Does the default control scheme fit your preferences, and what, if any, alternative controls would you suggest?
- What thoughts do you have on the UI design and flow?
- Having seen the game played, would you be able to describe how the main weapon functions fully, and what each enemy type encountered does?
- Do you have any further comments relating to the usability of the game?

Asked how challenging the participants found the tasks to complete, the responses were not at all for the first, none found discovering the controls difficult, and all four stated that in general, completing two levels was not challenging - although without playing the game personally, it is of course difficult to tell exactly. Additionally, when asked about the controls, one participant missed one - indicating a possible need for more clearly broadcast on-screen tutorials.
The second question related to discovered issues, and the previously observed issues were mentioned. Additionally, it was noticed during the task period that the participant with Deuteranopia found the purple reward orb was difficult to make out against the blue background of the levels.
Asked whether the default control scheme fit the participants preferences, and whether there should be any changes, all responded that the scheme seemed comparable to other FPS games, with one participant saying that "It's very standardised with other games that I'm aware of" - which is the goal when making a game, as controls should be familiar to pick up. One suggested that the control scheme should be configurable by the player.
Asked for general thoughts on the UI design, and the flow of the UI, all participants liked the look of the main menu, pause menu and options menu, with one pointing out some design discrepancies with the upgrades menu. One participant said "IT (the UI design) suited the style of the game and was very intuitive".
Asked to describe the full feature-set of the railgun and what each enemy encountered does, three participants were able to do this completely. The one who did not check the controls menu could not describe the dash ability available to the player, and the participant with Deuteranopia did not realise that there was a fourth enemy with red armour which fired rockets more frequently. There is, however, some lack of consensus here, as not every participant had every enemy type generated for them to see.
Finally, participants were asked for general comments. Reiterating previous comments, two participants stated that an on-screen tutorial would be useful when entering a new room. One participant stated that an on-screen indicator for progress throughout the room, with a hint on what next step to take, would be useful. Finally, a participant requested that the red enemy be given a different colour or physical distinction, and that the loot drop be given a different colour and some kind of special effect to indicate more clearly that it has been dropped into the level. One participant said that "the motion was fluid, having had issues on proper games, fully built ones, lately, this all seemed to flow together." which is a strong indicator of the flow of the game. Another pointed out that, for balance, "maybe (you should) balance the enemies a bit more) - as they were quite difficult to manage.

As a result of this usability testing phase, an in-game list of controls, shown in figure 33, for operating the weapon was added - after performing an action once, the action would fade away, and upon completing all actions, the list will disappear. Additionally, the loot drop was turned into a very bright green, shown in figure 32, and had a new sound effect added for its appearance alongside

a UI indicator, and the previously red robot was changed out to a yellow one. Additionally, during testing it was noticed that it was difficult to aim with the gamepad, so a modifier was added to the aiming code which lowered the sensitivity of movement if using a gamepad and aiming at an enemy, to reduce the chances of the player overshooting the enemy.

Figure 32: The new loot drop colours, chosen to stand out against blue.

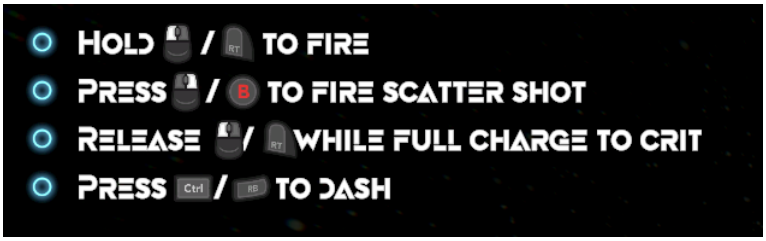


Figure 33: The new controls tutorial. Each bar fades out when the action is completed, and uses images to remove unnecessary, clunky text, as an accessibility feature.

Despite the required additions, in general usability testing showed that the game was in a good state - though, of course, this testing would have been benefited by an additional round where new participants could partake in the gameplay themselves. No major issues which could not be resolved were found, and the participants were able to direct the player to complete multiple levels, navigate the UI and identify the controls easily, as well as discover what the point of the game was and how to progress in it with no prior preparation.

6. Implementation

This section will discuss the methods of implementation for various components in the game, as well as providing evidence of implementation to support the evaluation of the project in the final section. As there is a lot of visual evidence to provide, it has all been collated into this section to improve legibility.

6.1. General State

The project largely meets the objectives laid out in the design document. The resulting game is limited in scope, but offers an example of how it would all work: randomly generating dungeons the player can choose between, a variety of AI per the design document, and some beyond, a

responsive and enjoyable character with one weapon option, the sniper/railgun, and ability to upgrade performance both temporarily and permanently, within the dungeon and outside. Many of the assets in the project were custom made - but there are exceptions, and these are clearly listed within the Assets/3rd Party Assets.txt file inside the project files, and as many of them as could be relocated were moved into the Assets/3rd Party Assets folder path.

There are some issues however. The visual fidelity is not quite up to the intended standard, with levels in particular. Due to issues listed in a later chapter, lighting the prefab-based levels proved unsustainable - so a solution requiring minimal to no lighting was needed. A holo-deck style shader and material were made to give the levels a simulation look, which also generated enough ambient light and bloom to keep things bright.

6.2. Character Controllers

The first implemented components were the character controllers, responsible for movement, abilities, weaponry and UI. The movement controller implemented many mechanics commonly found in platforming game: coyote time, jump buffering and apex boosts, giving the player additional control and a movement speed boost when at the apex of their jump. The implementations are found in the figures below. Below this additional functionality, the basic movement controller provides full support for 3D movement and jumping, as well as a method hooked onto the camera rotation which provides functionality for interacting with world-space objects. Support for controllers is offered through Unity's inbuilt input management system.

```
2 references
private bool canUseCoyote => _coyoteUsable && !collisionDown && timeLeftGrounded + coyoteTimeThreshold > Time.time;
1 reference
private bool hasBufferedJump => grounded && _lastJumpPressed + _jumpBuffer > Time.time;
```

```
if (grounded || canUseCoyote)
{
    // reset the fall timeout timer
    fallTimeoutDelta = fallTimeout;

    // stop our velocity dropping infinitely when grounded
    if (verticalVelocity < 0.0f && grounded)
    {
        verticalVelocity = -2f;
    }

    // Jump
    if (input.jump && canUseCoyote || hasBufferedJump)
    {
        _coyoteUsable = false;
        timeLeftGrounded = float.MinValue;
        jumpingThisFrame = true;
        // the square root of H * -2 * G = how much velocity needed to reach desired height
        verticalVelocity = Mathf.Sqrt(jumpHeight * -2f * gravity);
```

Figure 34: The code allowing coyote time and jump buffering.

The dash controller functions as an addendum to the movement controller, applying visual effects and invulnerability while moving the player a set distance in their movement direction. By interpolating between the original and end point, the player will never be able to dash through walls.

The combat controller and associated sniper class work together to handle the player's outward facing combat capabilities. The combat controller handles incoming damage, while limiting this damage so that multiple sources cannot affect the player within a short period to remove the possibility of the player being killed in 'one shot'. The sniper functions as laid out in the design document, allowing the player to charge up the weapon to deal increasing damage, providing a

```
1 reference
private void CalculateJumpApex()
{
    if (!collisionDown)
    {
        // Gets stronger the closer to the top of the jump
        apexPoint = Mathf.InverseLerp(_jumpApexThreshold, 0, Mathf.Abs(controller.velocity.y));
    }
    else
    {
        apexPoint = 0;
    }
}
```

```
if (targetSpeed != 0.0f)
{
    var _apexBonus = Mathf.Sign(currentHorizontalSpeed) * apexBonus * apexPoint;
    currentHorizontalSpeed += _apexBonus * Time.deltaTime;
}
```

Figure 35: The code for calculating and executing the apex boost.

```
1 reference
private void CastForward()
{
    Ray ray = new Ray(cameraTransform.position, cameraRot);
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit, 2, LayerMask.GetMask("Interactible")))
    {
        lookingAtInteractible = true;
        PlayerInteract(hit);
```

```
1 reference
private void PlayerInteract(RaycastHit hit)
{
    if (input.interact)
        hit.collider.gameObject.SendMessage("Interact", SendMessageOptions.DontRequireReceiver);

    input.interact = false;
}
```

Figure 36: Code for handling interactions in the world-space - world-space UI is finicky and provides no support for 3D intractable objects.

```
float startTime = Time.time;
float speed = distance / dashTime;
controller.gravity = 0;
dashBuffed = true;
CharacterCombatController.instance.dashInvulnerable = true;
chromaticAberration.intensity.Override(1.0f);
dashEffectImage.gameObject.SetActive(true);
dashEffectImage.alpha = 1;
LeanTween.alphaCanvas(dashEffectImage, 0, dashTime);

while (Time.time < startTime + dashTime)
{
    // Unsure why this needs to be multiplied by .5 to achieve the correct distance, but it does
    controller.controller.Move(controller.targetDirection.normalized * speed * Time.deltaTime * .5f);

    yield return new WaitForEndOfFrame();
}
```

Figure 37: Part of the implementation of the dash ability - the other half essentially does the reverse.

short window for a critical shot, and a secondary shot firing a shotgun spread.

```
IEnumerator ChargeUp()
{
    animator.SetBool("HoldingCharge", true);
    while(timeHeld < initialCharge)
    {
        timeHeld += Time.deltaTime;
        percentHeld = timeHeld / initialCharge;
        double percentCharged = Math.Round(percentHeld, 2);
        chargeIndicator.currentPercent = percentHeld * 100;
        yield return new WaitForEndOfFrame();
    }
    percentHeld = 100f;
    chargeIndicator.currentPercent = 100;
    critical = true;
    lastCritRoutine = StartCoroutine(CriticalPeriod());
}
```

(a) Charge Routine.

```
IEnumerator CriticalPeriod()
{
    critIndicator.gameObject.SetActive(true);
    float timeDelta = 0f;
    while(timeDelta < CharacterStatisticsController.critDuration)
    {
        timeDelta += Time.deltaTime;
        yield return new WaitForEndOfFrame();
    }
    critical = false;
    critIndicator.gameObject.SetActive(false);
}
```

(b) Crit Handling.

Figure 38: (a) shows the charge routine for the sniper, while (b) shows the critical period handling.

6.3. Random Generation

The random room generation went through multiple phases, each with functioning prototypes before settling on the ultimate solution. One component of the generation which stayed true throughout is the method of generating the room itself - instead of procedurally generating the superstructure, the generator would piece together two halves of a room to seamlessly fit together. By this method rooms could still be designed to facilitate interesting gameplay, while providing increasingly more variations the more room halves are made.

The first solution trialled was one contiguous generation. By utilising hallway assets, rooms could be pieced together physically - a room would have two hallways leading away, leading to new rooms, leading to new hallways. Upon review, it was found to be poorly performing - instantiating so many objects at runtime failed to correctly work in the editor, and a homemade solution needed to be made to load and unload rooms as the player navigated. Additionally, no satisfying solution was found for the ability of the player to traverse backwards - with hallways added, it made sense to allow the player to move back into previously cleared rooms, though this would provide no benefit and potentially confuse the player. With the ability to traverse backward removed no reason for the area to be physically connected presented itself. Therefore, alternative solutions were sought.

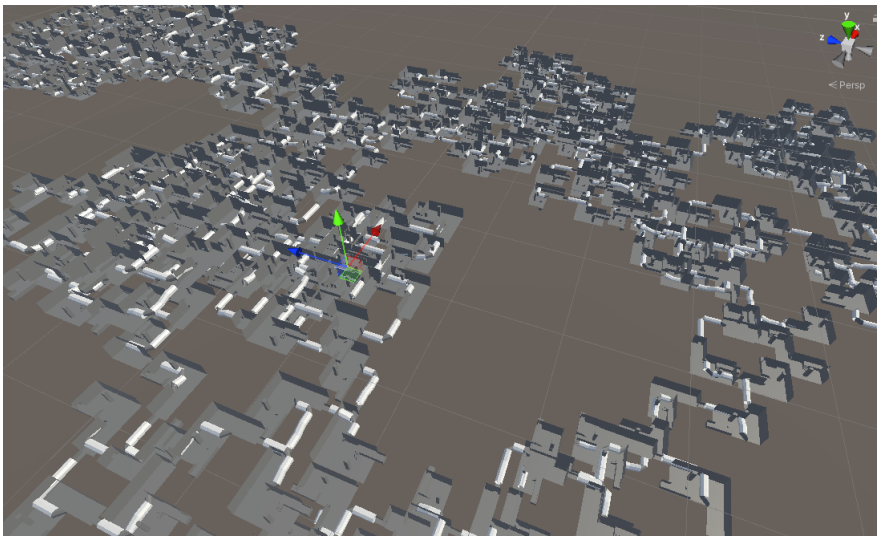


Figure 39: An example of a randomly generated dungeon with rooms connected by hallways.

The second solution was an attempt to remove the performance issues and unnecessary physical connections but retain hallways by creating portals which the player could walk through. By making these portals totally seamless, the player would never know that each hallway simply teleports the player back to the start of a new room which had just been generated, and the hallways could be closed off later. By moving the camera to a location such that its position and rotation is the same as the player's through the portal they are looking through, relative to its own portal, the resulting image will appear to be continuous with what the player would expect to see - visualised in figure 41. Retrieving the screen-space coordinates of the four corner vertices of the portal and mapping them to the output camera texture allows the correct portion of the texture to be 'cut out' and applied to the screen of the portal, as shown in figure 42. By then teleporting the player to the relative position and rotation of the twinned portal upon contact, a seamless transition experience is formed. However, upon testing, the performance hit of rendering a total of three cameras at once (one per portal plus the player) on low-end hardware proved to be damaging to the experience, as well as issues with instantiating the portals at runtime.

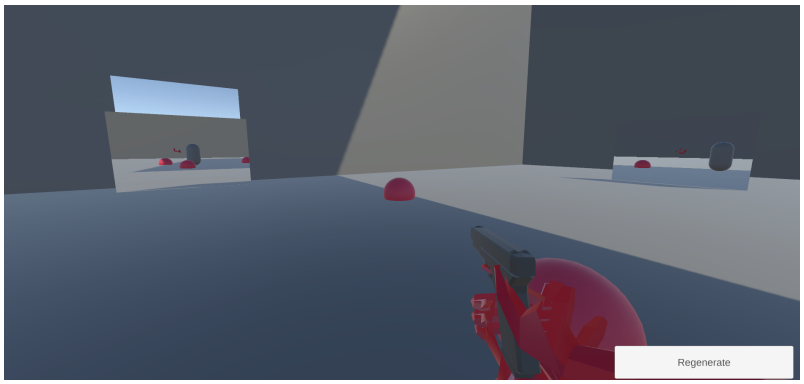


Figure 40: A display of the portals in action from the player's point of view.

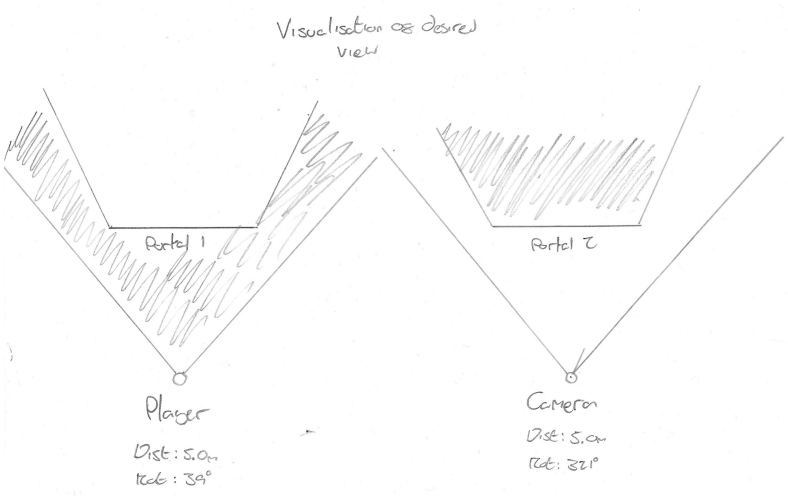


Figure 41: A representation of how the portals work, with the camera and player in the same relative positions to their own portals. The shaded volume of the Field of View cone represents what the player sees.

The third, and chosen solution, is also the simplest. By replacing corridors with teleportation rooms, the player can simply interact with a button to be instantly moved to the next room. This brings the additional benefits of not needing to design and model multiple corridor assets, a built-in solution to loading and unloading rooms as previous rooms can simply be thrown away as new rooms are generated, and it fits in with the setting.

As well as the method of room navigation, it is required to discern how to build the components together. All three trialled methods rely on the same underlying principle - stitching together

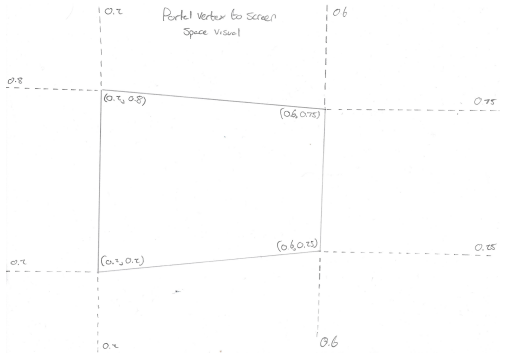


Figure 42: How screen vector coordinates are used to transpose the output texture of the portal's camera onto the portal itself, accounting for perspective.

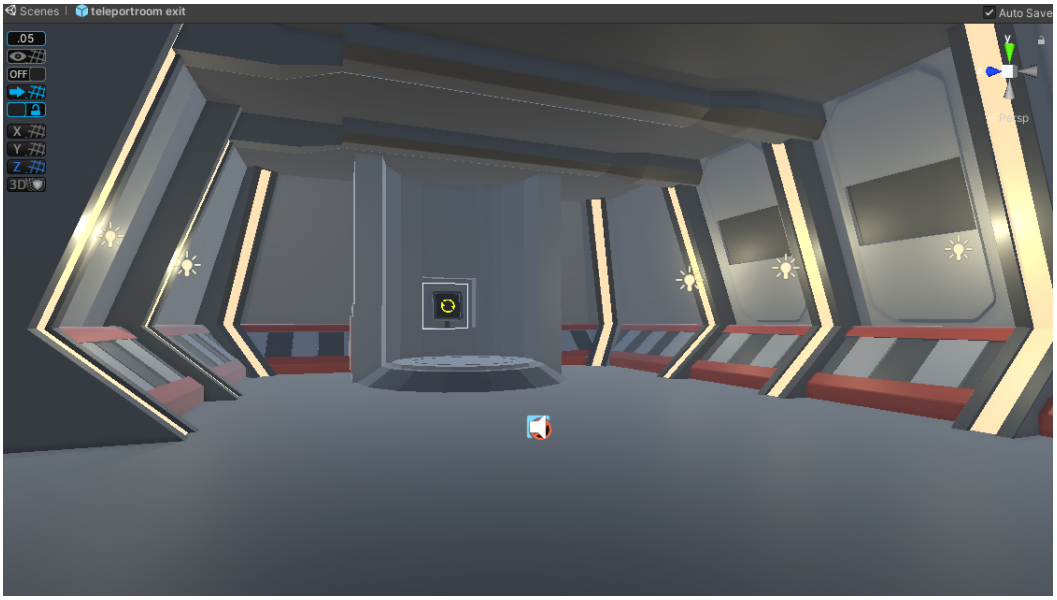


Figure 43: A view of the inside of the teleport room.

pre-built components seamlessly. The solution to this was to build in connection points to the prefabs - points the room generator could access and know that this is where the connection must be performed. To facilitate this, a method for recalculating the positions of prefabs relative to these connection points was made in case something went wrong during instantiation - as proved to be the case very often. With this, the generation was a simple case of randomly choosing the correct component to instantiate in order, as shown in the figure below. In case of changes further into development, the generation was made as abstract as possible - with any number of exit options or room 'halves' able to be slotted together.

By storing the instantiated segments in a Room Controller class, the state of the room could be modified as the player progressed, locking and unlocking specific doors and assigning upgrades to each exit which could be maintained into the next room.

6.4. AI

The AI all inherits from a single base class, containing shared methods and variables - damage management, health, respawning and death, etc. Built on top of that are AI specific methods and variables allowing them to perform unique actions, as shown in the related figures. Idle behaviour,

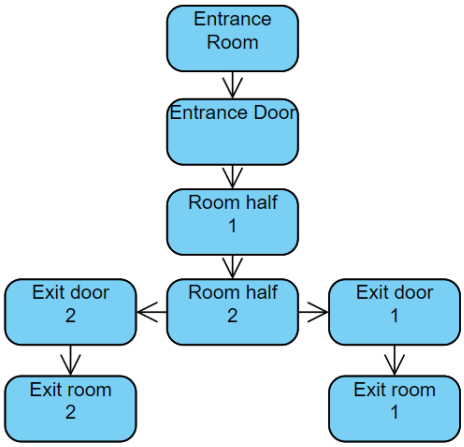


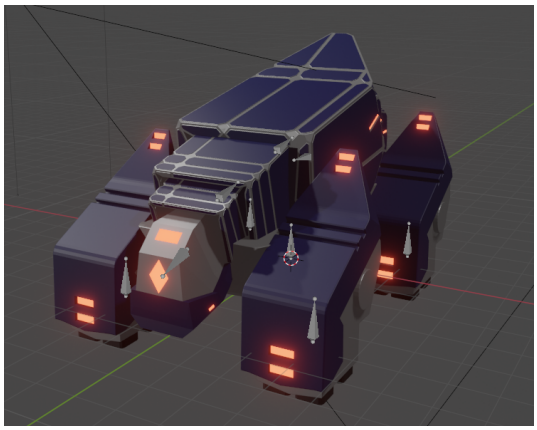
Figure 44: Sequence of generation for the rooms.

```
9 references
void RepositionTransforms(Vector3 origin, Vector3 connection, GameObject objectToMove)
{
    Vector3 delta = origin - connection;
    objectToMove.transform.position += delta;
}
```

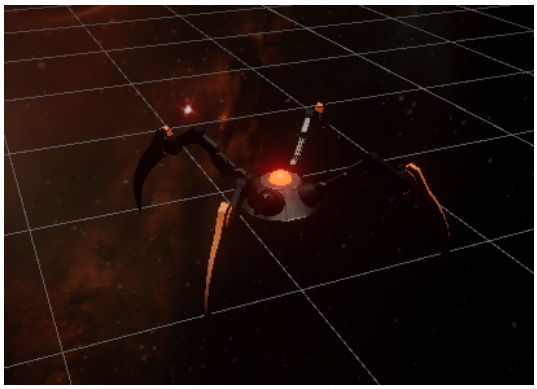
Figure 45: Simple method of re-positioning rooms via the vector deltas.

operating when the AI is unaware of the player, consists of randomly roaming around its starting position, and is shared between all AI.

(a) Shooter / Missile Bot.



(b) Smasher / Caterpillar Bot.



(c) Swarmer / Crabberbot.

Figure 46: Images of the three main models used for the AI - variants used very similar riffs off these models. The Crabberbot (c) model is not an original creation, while the others are.

```
9 references
protected abstract void Idle();

9 references
protected abstract void NavigateToPlayer();

8 references
protected abstract void Attack();

1 reference
public void initialise(int _numberOfRespawns, RoomController _room, Transform _spawnPoint, Transform _respawnBin)
{
    numberOfRespawns = _numberOfRespawns;
    room = _room;
    spawnPoint = _spawnPoint;
    respawnBin = _respawnBin;
    maxHealth = health;
}

1 reference
public void Respawn()
{
    Instantiate(explosionPrefab, transform.position, transform.rotation);
    //gameObject.SetActive(false);
    hasSeenPlayer = false;
    numberOfRespawns -= 1;
    navAgent.Warp(respawnBin.position);
    navAgent.isStopped = true;
    if(animator)
    {
        animator.SetBool("Dead", false);
    }
    StartCoroutine(RespawnDelay());
}
```

Figure 47: Example of inherited methods found in the base enemy class.

```
2 references
protected override void Idle()
{
    if (Time.time < timeUntilNewIdlePoint) return;

    if (!idleSet)
    {
        SearchIdleWalkPoint();
        if (idleSet) navAgent.SetDestination(idleNavPoint);
    }

    Vector3 distanceToWalkPoint = transform.position - idleNavPoint;

    if (distanceToWalkPoint.magnitude < 1)
    {
        idleSet = false;
        timeUntilNewIdlePoint = Time.time + Random.Range(1.0f, 3.0f);
    }
}

1 reference
private void SearchIdleWalkPoint()
{
    float randomZ = Random.Range(-idleRange, idleRange);
    float randomX = Random.Range(-idleRange, idleRange);

    idleNavPoint = new Vector3(originPoint.x + randomX, originPoint.y, originPoint.z + randomZ);

    if (Physics.Raycast(idleNavPoint, -transform.up, 2f, groundMask))
    {
        idleSet = true;
    }
}
```

Figure 48: The idle behaviour shared by all AI.

The Swarmer enemy will generate an attack vector upon proximity to the player, and then charge along it at a high speed, damaging the player on contact. All other methods utilised by this enemy are shared by other enemy types, as they are modular.

The missile robot will attempt to maintain distance from the player while dodging obstacles, and fire off missiles at the player's position. By aiming at the current position of the player instead

```
2 references
protected override void Attack()
{
    attacking = true;
    attackPoint = CreateAttackPath();
    attackWindupTime = Time.time + attackWindup;
    remainStationary = true;

    navAgent.isStopped = true;
    navAgent.updatePosition = false;

    StartCoroutine(AttackWindup());
}

1 reference
void MoveOnAttackVector()
{
    if (!attackPathSet) attacking = false;

    float step = attackSpeed * Time.deltaTime;
    transform.position = Vector3.MoveTowards(transform.position, attackPoint, step);

    if (Vector3.Distance(transform.position, attackPoint) < 1.0f)
    {
        attackPathSet = false;
        attackOnCooldown = true;
        remainStationary = true;
        attackWindupTime = Time.time + attackWindup;
        timeUntilNextAttack = Time.time + attackCooldown;

        StartCoroutine(AttackWinddown());

    }
}
```

Figure 49: Swarmer attack behaviour.

of predicting movement, the player will be encouraged to remain mobile instead of simply moving back and forth to throw off the aim.

```
void NavigateAwayFromPlayer()
{
    if (Vector3.Distance(playerLastPos, player.position) < 0.5f) return;


    Vector3 travelDir = player.position - transform.position;
    Vector3 finalDir = (travelDir.normalized * attackRange);
    Vector3 targetPos = player.position - finalDir;

    NavMeshHit hit;
    if(NavMesh.SamplePosition(targetPos, out hit, 4.0f, NavMesh.AllAreas))
    {
        navAgent.SetDestination(hit.position);
    }
}
```

Figure 50: Code to maintain a set distance from the player.

The caterpillar bot attempts to move in front of the player, blocking their view while throwing large area-of-denial bombs which detonate on a timer. These bombs do extremely high damage, and the AI has very high health, but they are largely not aimed and easy to dodge if the player remains mobile.

Additionally, the individual behaviours of the AI are modular and pluggable - meaning that

```
void ShootProjectile()
{
    animator.ResetTrigger("Damaged");
    animator.SetTrigger("Fire");
    missileLauncherFireSFX.Play();
    Vector3 dir = player.position - transform.position;
    projectileSource.transform.rotation = Quaternion.LookRotation(dir);

    GameObject _projectile = Instantiate(projectile, projectileSource.transform.position, projectileSource.transform.rotation);

    _projectile.GetComponent<Rigidbody>().velocity = ((player.position - projectileSource.transform.position).normalized) * projectileSpeed;
}
```

Figure 51: Rocket bot attack code.

```
protected override void NavigateToPlayer()
{
    Vector3 P2 = player.position;
    Vector3 P1 = transform.position;
    float circumferenceVariation = Random.Range(-20.0f, 20.0f);
    Vector3 P3 = P2 + Vector3.ClampMagnitude(Quaternion.AngleAxis(circumferenceVariation, Vector3.up) * (P1 - P2), attackRange);

    navAgent.SetDestination(P3);
}
```

Figure 52: Code used to navigate into the face of the player, where the attack state will be activated.

the behaviour used to maintain distance by the Rocket Bot could simply be plugged into the Caterpillar, causing it to maintain a set distance from the player at all times. Similarly, the Rocket Bot could continue to maintain a far distance, but its attack could be swapped out for the charge attack of the swarmers. This is on show in the form of a shielded robot - who generates a forward-facing array of shields the player must destroy or dash around, shown in figure 53, and the boss enemy, which utilises the base Missile Bot behaviours but replaces the attack behaviour with a new salvo of missiles, displayed in figure 54.

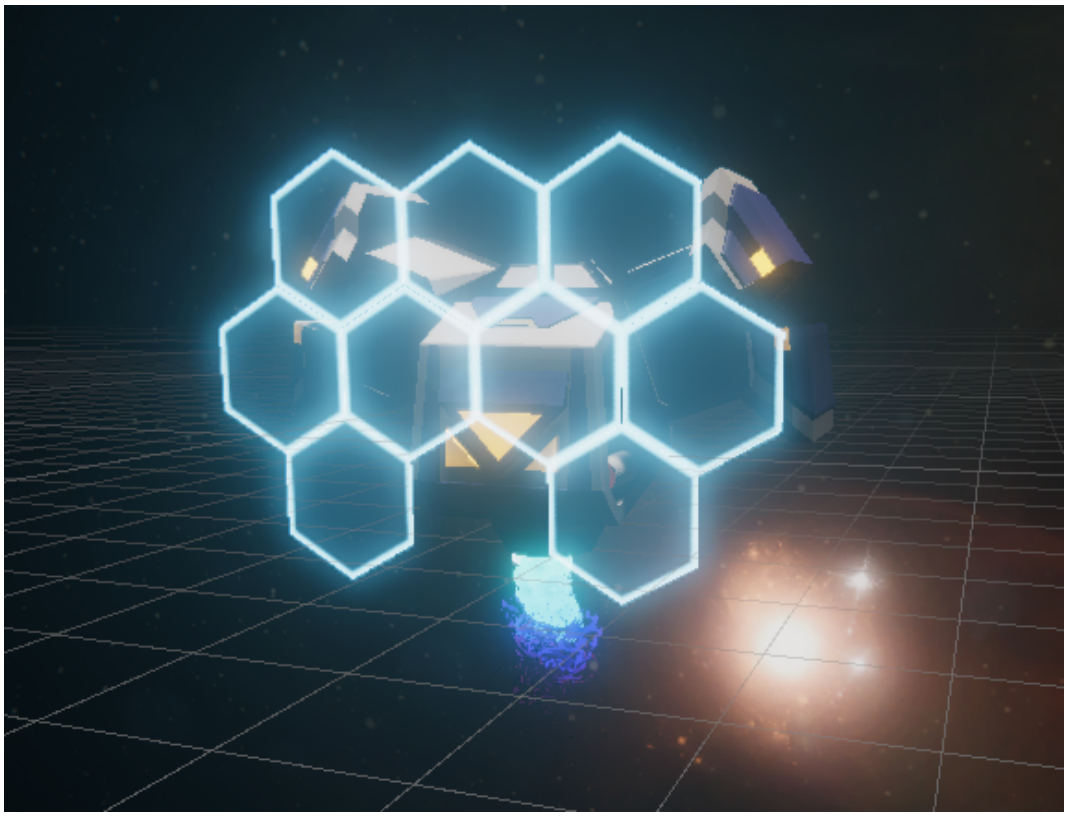


Figure 53: The shielded enemy.

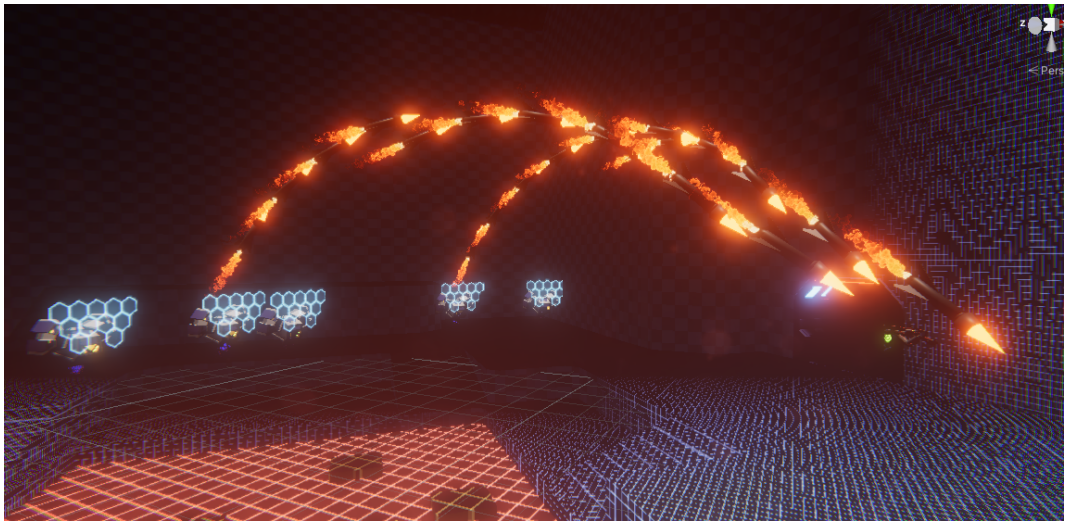


Figure 54: The salvo from a wave of boss enemies - typically the player would only expect to fight one.

6.5. Upgrades

A variety of upgrades are available, both within the dungeon and in the hub-zone. These were created using scriptable objects - a data container used, in this case, to store data in the editor for use at runtime. These objects could be used to represent any form of upgrade, randomly chosen and displayed in a selection/shop screen, and carry out the upgrade process. By finishing rooms, the player is presented with orbs which grant one of three randomly chosen upgrades from specific trees, which are chosen by entering the relevant door. The player can also be rewarded with gold to spend at shops, where they are presented with similar choices, and gems which they can spend on upgrades from a store at their hub, where upgrades are persistent.

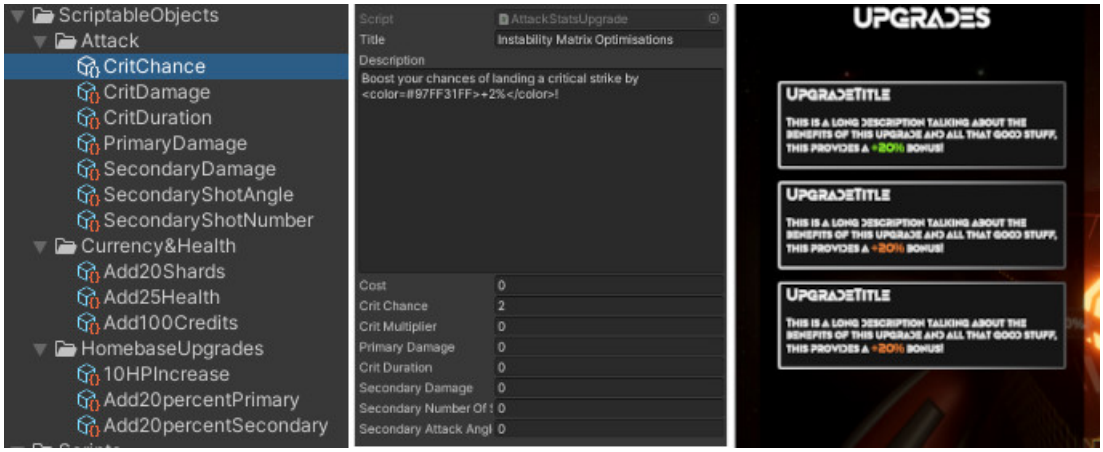


Figure 55: The method of utilising scriptable objects and the display screen.

6.6. Accessibility

Implementation of accessibility guidelines was a priority during development. To facilitate this, the earlier mentioned Game Accessibility Guidelines were utilised, with the aim of implementing all the basic accessibility features, and as many intermediate features as reasonable. Some of these guidelines were implemented as a result of the Usability Testing in Section 5.8, while others were in place before assessing the accessibility guidelines checklist, due to their implementation simply

being solid game design choices - for example, using images in place of test where reasonably possible, and not relying solely on one sensory input to get across vital information. The full checklist, and descriptors, of the accessibility guideline components are found both on the cited website[7], and the checklist used in this project, complete with completion status and whether the component is relevant, is found in the Appendix C. The guidelines on Motor Control disabilities and skill levels is displayed in figure 56.

Motor		
(Control / mobility)		
Basic		
Allow controls to be remapped / reconfigured		No
Ensure controls are as simple as possible, or provide a simpler alternative		yes
Ensure that all areas of the user interface can be accessed using the same input method as the gameplay		yes
Include an option to adjust the sensitivity of controls		yes
Ensure interactive elements / virtual controls are large and well spaced, particularly on small or touch screens		yes
Intermediate		
Support more than one input device		yes
Make interactive elements that require accuracy (eg. cursor/touch controlled menu options) stationary		yes
Ensure that multiple simultaneous actions (eg. click/drag or swipe) are not required, and included only as a supplementary / alternative input method		yes
Ensure that all key actions can be carried out by digital controls (pad / keys / presses), with more complex input (eg. analogue, speech, gesture) not required, and included only as supplementary / alternative input methods		yes
Include an option to adjust the game speed	no	
Avoid repeated inputs (button-mashing/quick time events)		yes
If producing a PC game, support windowed mode for compatibility with overlaid virtual keyboards		yes

Figure 56: The checklist of components, at the basic and intermediate level, for the motor control/mobility accessibility requirements, with relevance in the second column and state of completion in the third.

6.7. Level Designs

Issues discovered while attempting to create level designs are discussed in sections 7.1 and 7.2, for an explanation on why the levels presented here differ from the proposed levels aesthetic in the design document. In general, due to technical and time limitations, the levels needed to be changed to fit a new aesthetic which required no real-time lighting - the chosen aesthetic was that of a holographic level.

With the new aesthetic established, exploration needed to be performed into what makes a holographic level look good. A shader was created to create procedural grids across surfaces, and some complex geometry was loaded into Unity to get a better idea of how the old design plans would work with this new shader, as shown in figure 57. As shown, the complex nature of the geometry was no longer visible without looking very closely - the grid gave the effect of flattening down the girders and edges of the wall segments. Instead, work began on creating much boxier, flatter arenas - and with such simple geometry required (but the integration of cut out holes for doors still necessary) all work was done in Blender instead of Unity.

After level structures were created in Blender, they were imported into Unity and further assets were added. A variety of dynamic and harmful environmental assets were created: moving platforms, floors which hurt the player, and floors which would remain safe for a defined period after

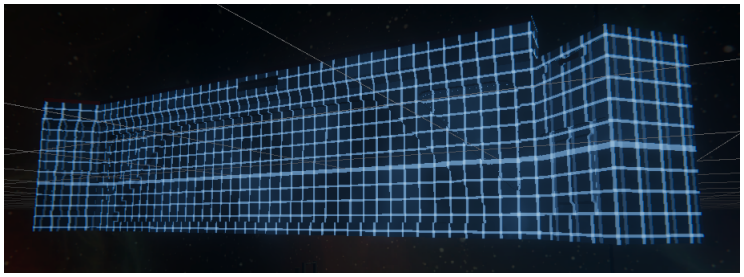


Figure 57: The procedural holographic grid shader applied to complex geometry created previously to build levels from.

the player lands on them, before transforming to a damaging state. These could be added to the levels to create further variations from one structure - the Room1Exit2 prefab, for example, became four individual prefabs with their own unique layouts and obstacles, with very little additional effort. These components would then be strung together through the procedural generation process described earlier, and shown in figure 58.

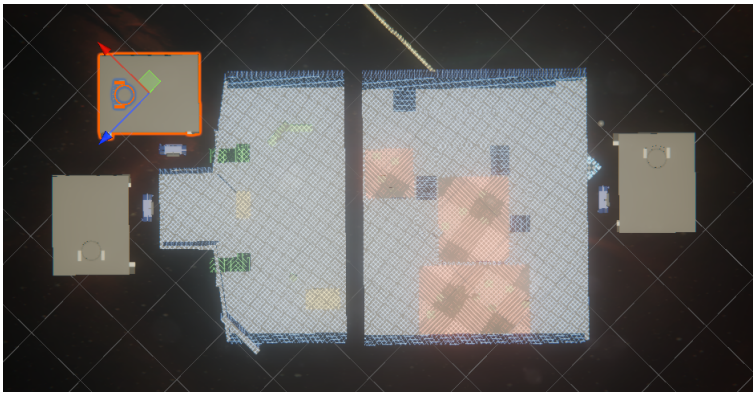


Figure 58: A procedurally generated level with each component exploded outward, to show how they interconnect at runtime.

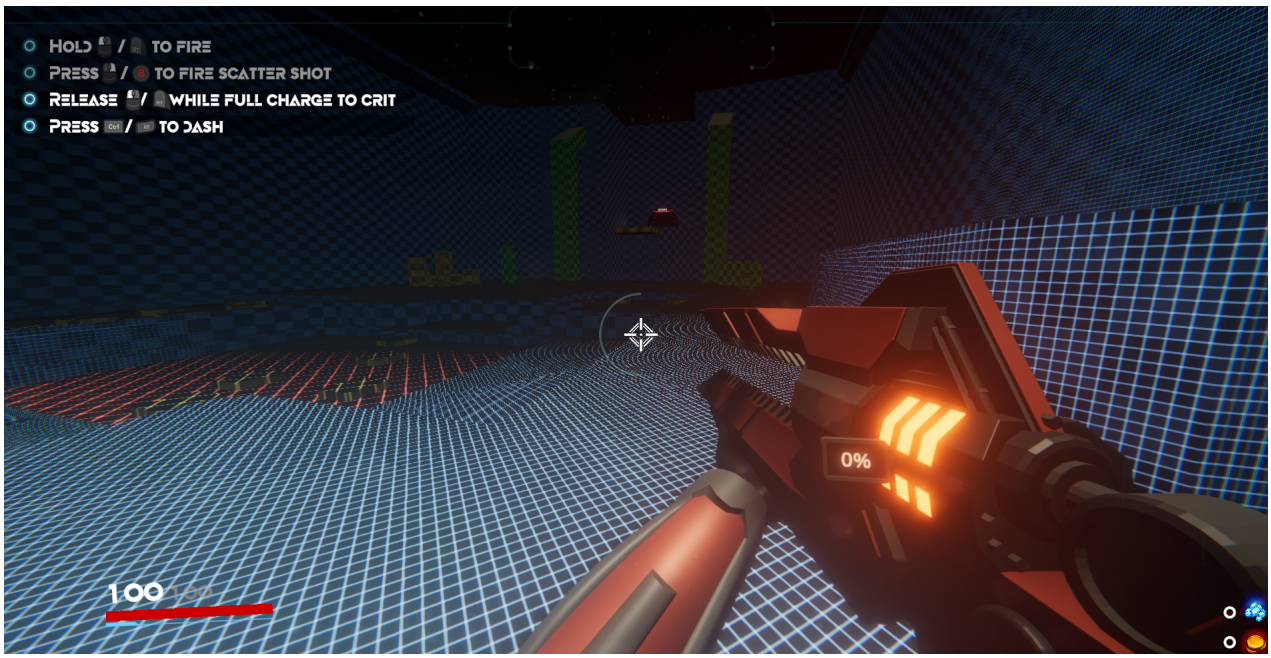


Figure 59: A procedurally generated room, from the viewpoint of the player.

6.8. UI

The UI was created using Krita, with largely simple geometric shapes. The final designs are displayed in the figures below, and followed the planned designs aside from some minor alterations. Due to the planning, there were no major iterations aside from the originally used placeholder assets.

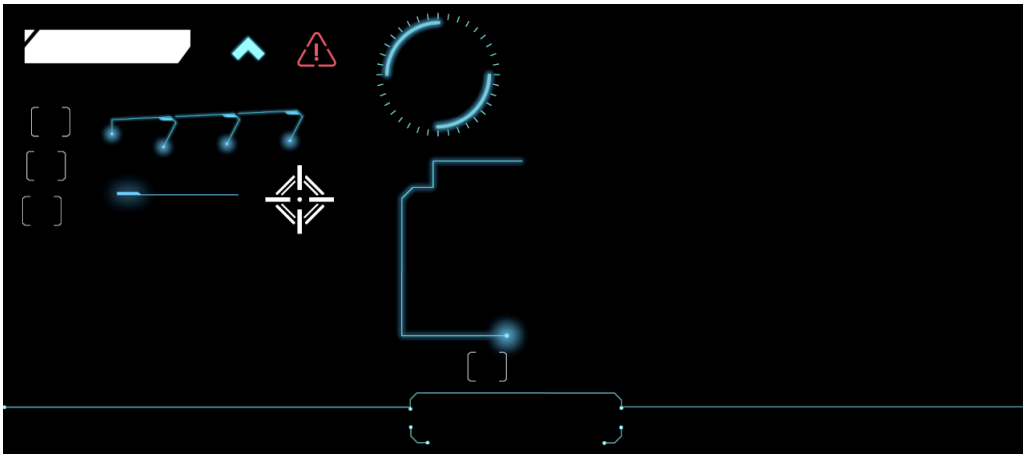


Figure 60: The independent UI assets created for this project

Figure 61: The assets used on the in-game UI

6.9. 3D Models

Many of the 3D models were custom designed in Blender, and all VFX and shaders were custom made. Shown below is the holotable, where the player selects their next mission to undertake in a flashy 3D-hologram environment, two variants of the missile robot with and without shields, the main door asset and the procedural damage state for enemies, where upon taking enough damage, they begin to disintegrate into nothing. The railgun and player arms, with animation keyframing, are shown in the last figures. All models, unless explicitly static (such as the immobile table of the holotable itself), are fully animated with animation state graphs.

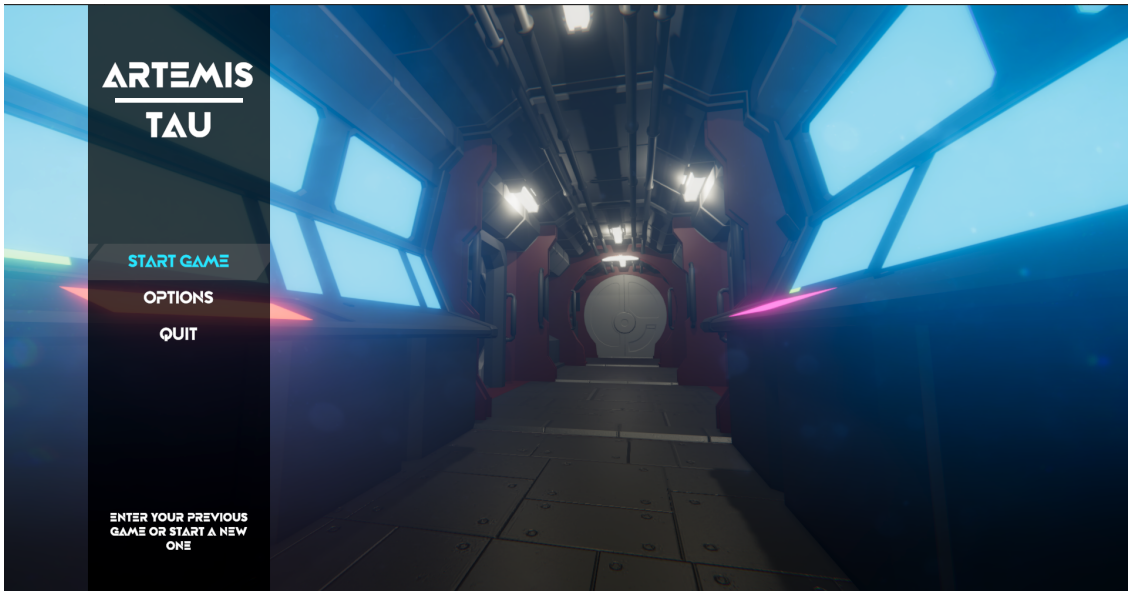


Figure 62: The final main-menu and pause menu design

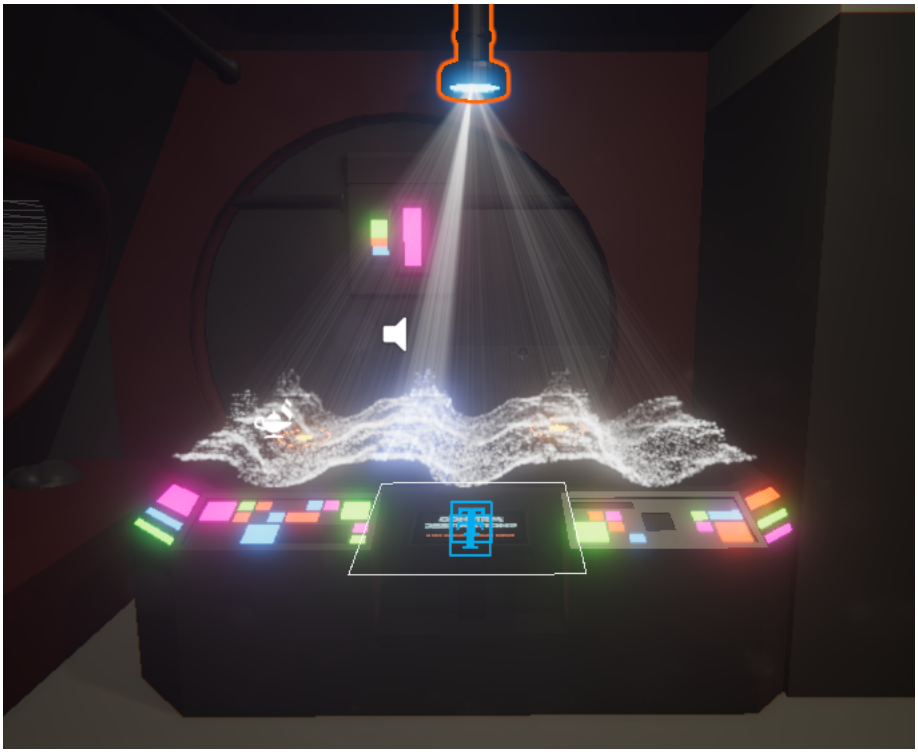


Figure 63: The holotable VFX and model

6.10. Audio

All audio used in this project is not original - the combat track is a part of a Unity Asset Pack from Cafofo[6], while the home-base music is a track purchased from Egosoft. Both are free to use for non-commercial projects under the Creative Commons License, and the Egosoft track would require replacing if this project were to become a commercial product.

Audio implementation itself was rather simple. Non-spatial audio, such as UI audio, could simply be dropped in the scene and triggered to play as required. Spatial audio required placing - but given the procedural creation of the game worlds, they really only needed placing at the source

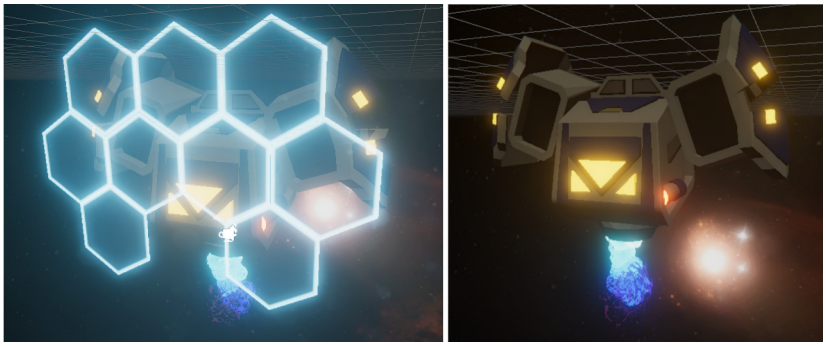


Figure 64: Two versions of the missile robot, with and without shielding which suffers procedural damage

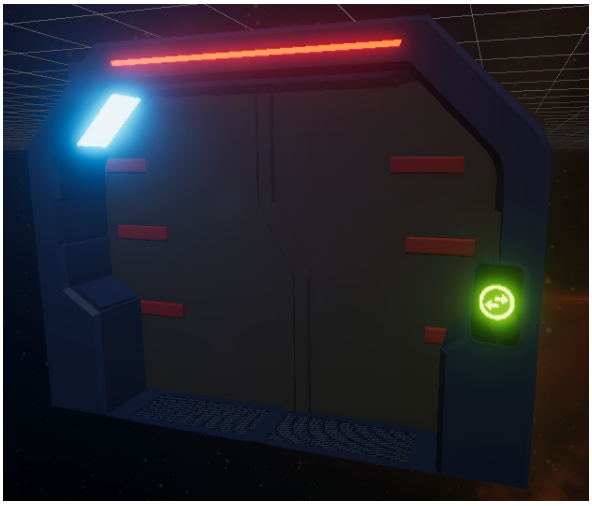


Figure 65: The primary door used in the room generation

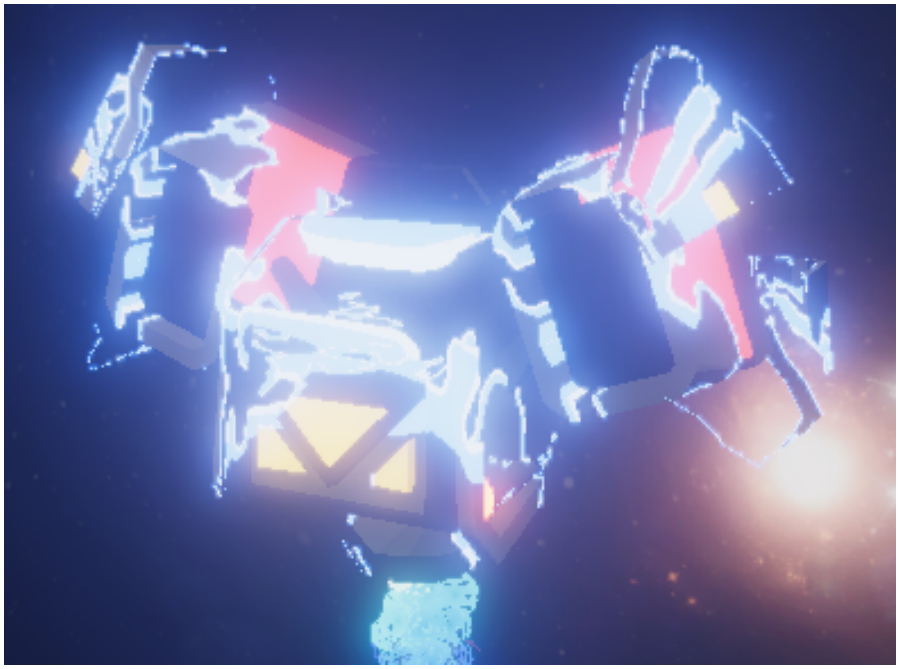


Figure 66: The procedural damage state of enemies

point of any object which required instantiation. The Unity animation suite also provided means to keyframe audio clips to certain points in animations, so footsteps choreographed to the actual animated footsteps were very easy to implement. All audio was tied to either a music or SFX volume mixer, themselves attached to a master volume mixer, allowing the options menu to edit

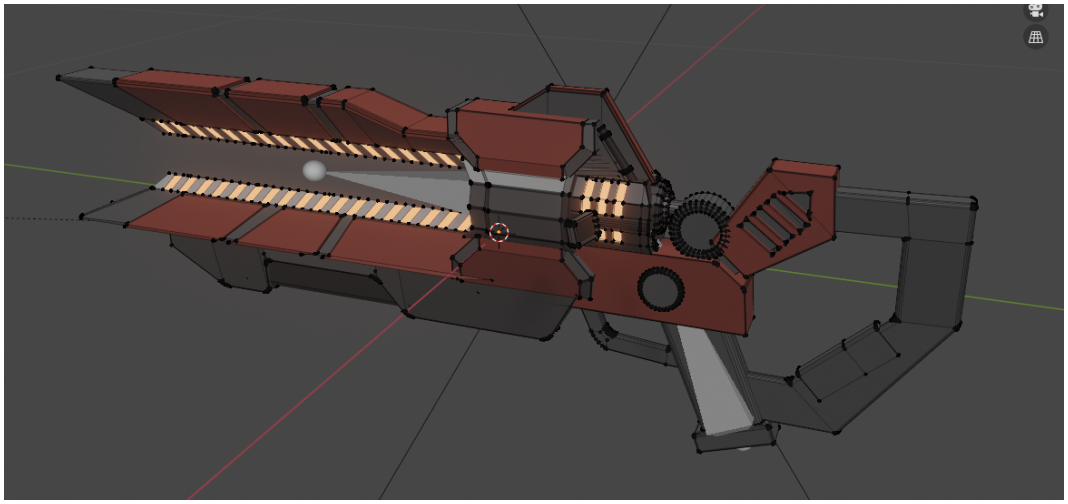


Figure 67: The created model for the player-held railgun

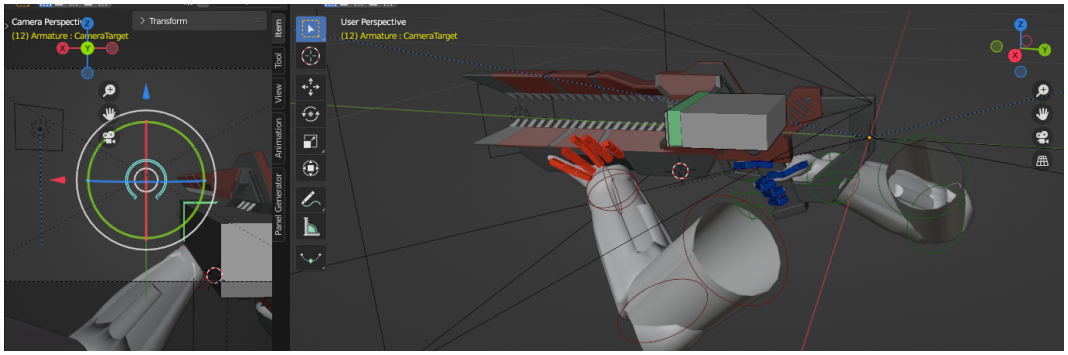


Figure 68: The rigged player arms holding the railgun, with visible armature bones

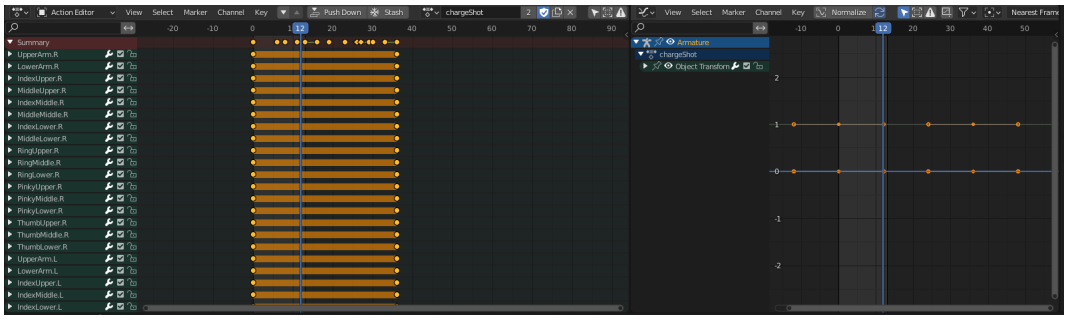


Figure 69: The keyframes for the charging shot animation

the three volumes as needed.

If this project were to progress further, a more sophisticated audio mixing technique would be hugely beneficial. A common modern technique for managing music is to key certain segments to actions in game - spearheaded by Doom 2016 and composer Mick Gordon. "By creating 50 variations on a verse and dumping them into a container, we can randomly pull verses until the game decides to kick up into a chorus, play a transition and start taking chorus variations, until we transition back down into the verse"[14]. This method, however, requires a lot of time and effort applied to creating the music tracks themselves, and requires much more infrastructure within the game to correctly detect the player's actions and interpret those into musical pieces which should play. A simpler, but less effective, addition would be the older technique of simply layering tracks on top of each other, adding more for more intense situations, or creating different variations (combat, quiet) of a track which can then be started up and spooled down depending

on the active situation, a technique used in Deus Ex Machina.

7. Development Issues

Two major issues were encountered during development - the limitations of the engine and chosen render pipeline removing the ability to utilise real-time lighting as planned for originally, and the knock-on effects of this limitation on the development of the procedurally generated rooms. While this may seem like a strict engine limitation which could not be resolved, it is important to note that more thorough research into the documentation of the available render pipelines would have revealed this vulnerability earlier into development, and allowed for workarounds or new original design assumptions to be made.

7.1. Engine Limitations

Multiple engine limitations were discovered shortly after the Christmas period which provided serious issues developing the rest of the game. The Universal Rendering Pipeline, chosen due to its focus toward smaller development teams providing less graphical quality but an easier development pipeline, restricts realtime lighting visibility on a single object to eight sources. The intended alternative is baking lightmaps onto the scene, which removes the need for realtime lighting in most cases - but prefab objects cannot contain lightmap data. This provided a difficult to resolve issue - the game was too far along to swap rendering pipelines, yet the chosen aesthetic would not function with such a limited number of lights. A solution was found in altering the aesthetic of the generated rooms to be a 'holodeck' style from the likes of Star Trek or Orville. A procedural shader was created which used emissive lines to create the illusion of lighting, and a single global light facing downward was applied over the scene, resolving the issues provided in a somewhat satisfactory manner, as shown in figure 70

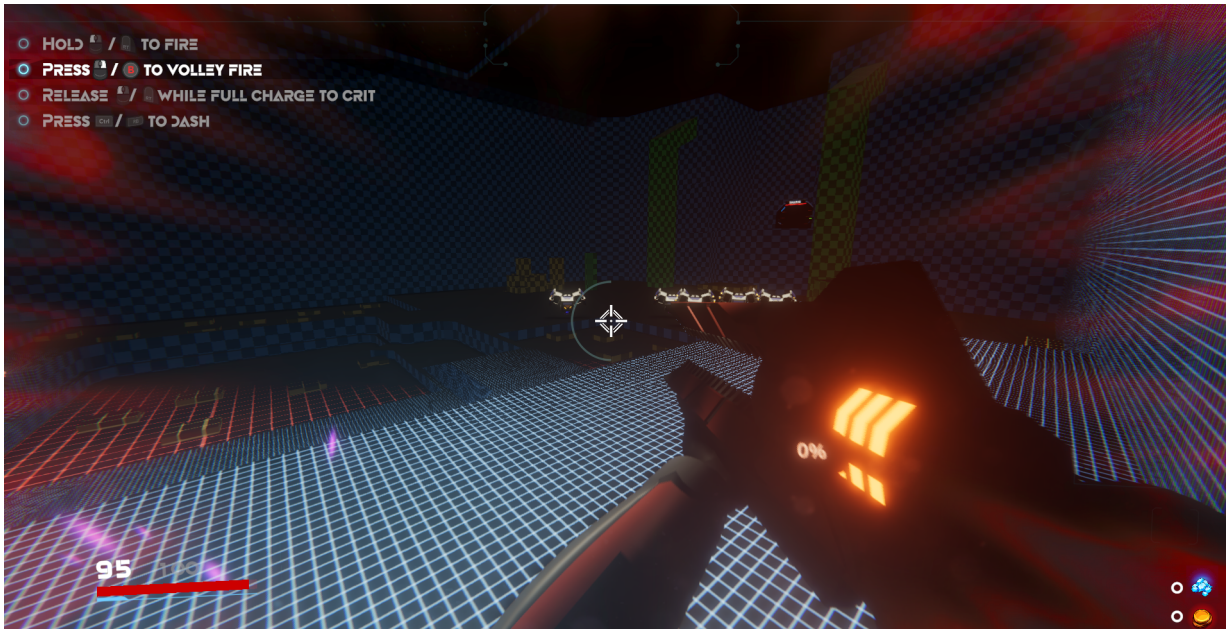


Figure 70: The new 'Holodeck' styling of the procedural rooms.

A secondary issue exists in all forms of Unity: NavMeshes, which the AI uses to navigate spaces, cannot be generated in prefab objects by default. Some research resulted in an experimental package available on GitHub[26] which provided access to components which can be attached

to prefabs, and allow the generation of NavMeshes at runtime. While this provides a necessary function, it does increase the time required to generate a new room by a small amount and being experimental, carries some inconsistent issues - one of which is AI with larger dimensions than the standard 0.5m radius occasionally cease functioning. This issue has been logged with Unity.

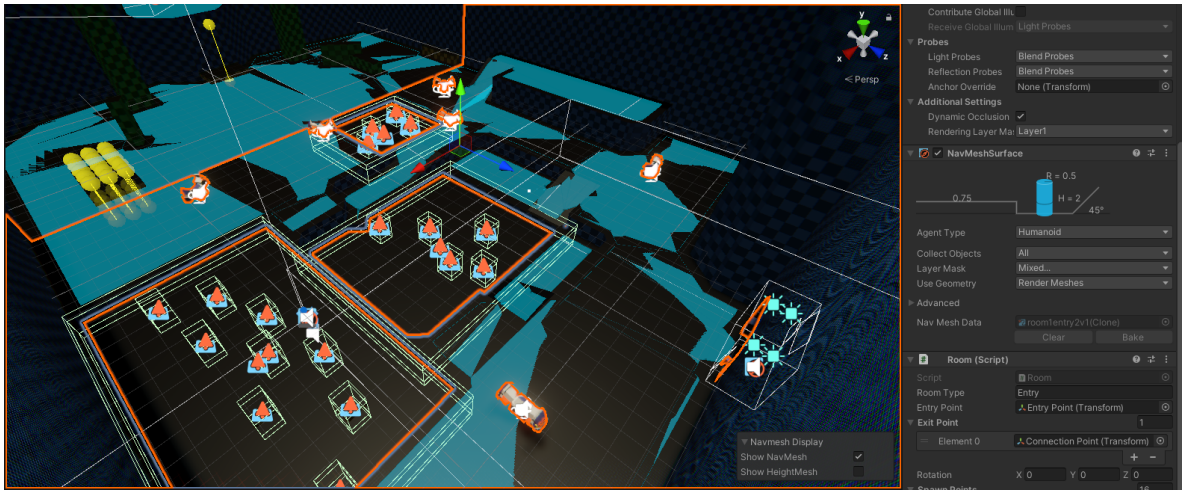


Figure 71: Somewhat erroneous display of the NavMesh grid overlaying the prefabs in blue, with the experimental component on the right side.

Expanded research into the render pipelines available in Unity would have helped to prevent the lighting issue - though given the additional complexities of using the HDRP over URP, it would likely not have been considered possible regardless. Modifications to the design brief before work started would have been possible, however. The NavMesh limitation is seemingly undocumented, and there is very little correspondence with it online - and at least an experimental solution is available. Regardless, further research at the start of the project would have proven useful.

7.2. Level Designs

Level designing proved particularly difficult to resolve. Creating level halves generic enough to be procedurally applied to each other, but designed well enough to create interesting gameplay experiences was a point of difficulty throughout the latter half of the project, with levels needing to be initially laid out before testing that they fit a good looking scale, with further work being required in Blender (a requirement due to the non-block form of the doors which needed cutting into the walls) to fill out the level with obstacles and height variations before testing with AI could continue. At this point, if a level proved unsuitable for the AI to navigate around (the caterpillar bot being so large was a particular issue here) the level would need to be restarted from scratch or largely deleted and reworked up from the original base. A half-solution found was to create the very basic level shape in Blender, then plant more interesting detail, obstacles, damaging floors and dynamic components in the Unity editor which is far more geared toward modular level design. More research into level design processes and actively playing games with well-regarded level design during the development process would have minimised this issue.

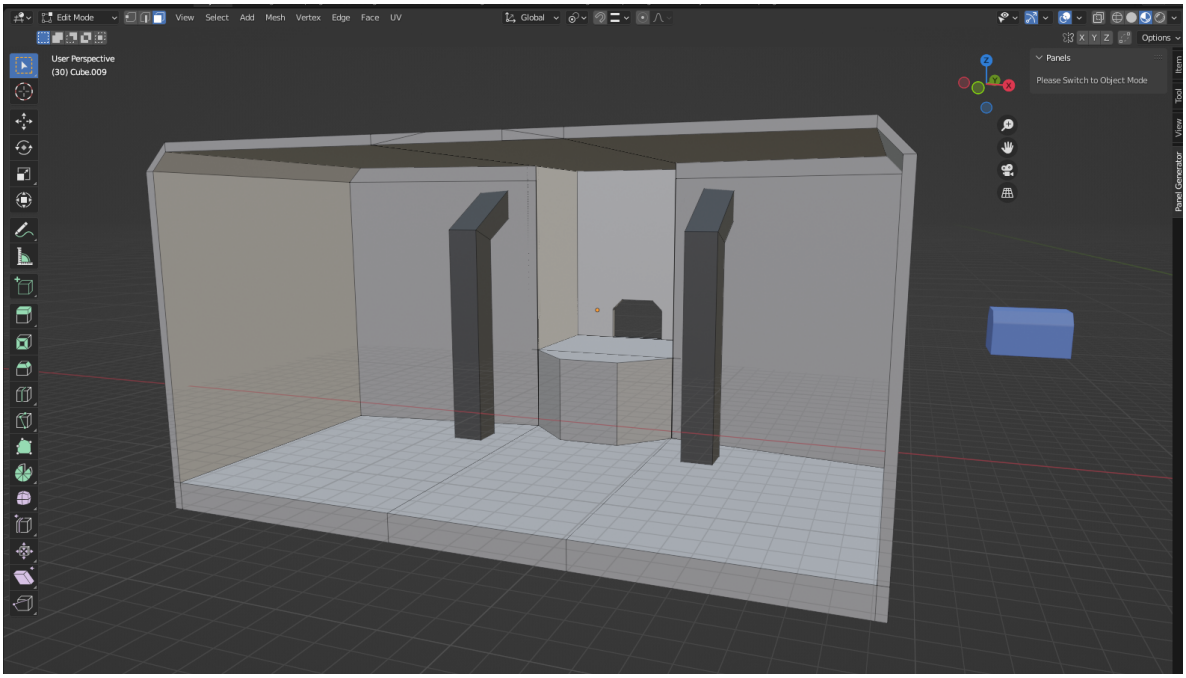


Figure 72: The empty level made in Blender, ready to import into Unity.

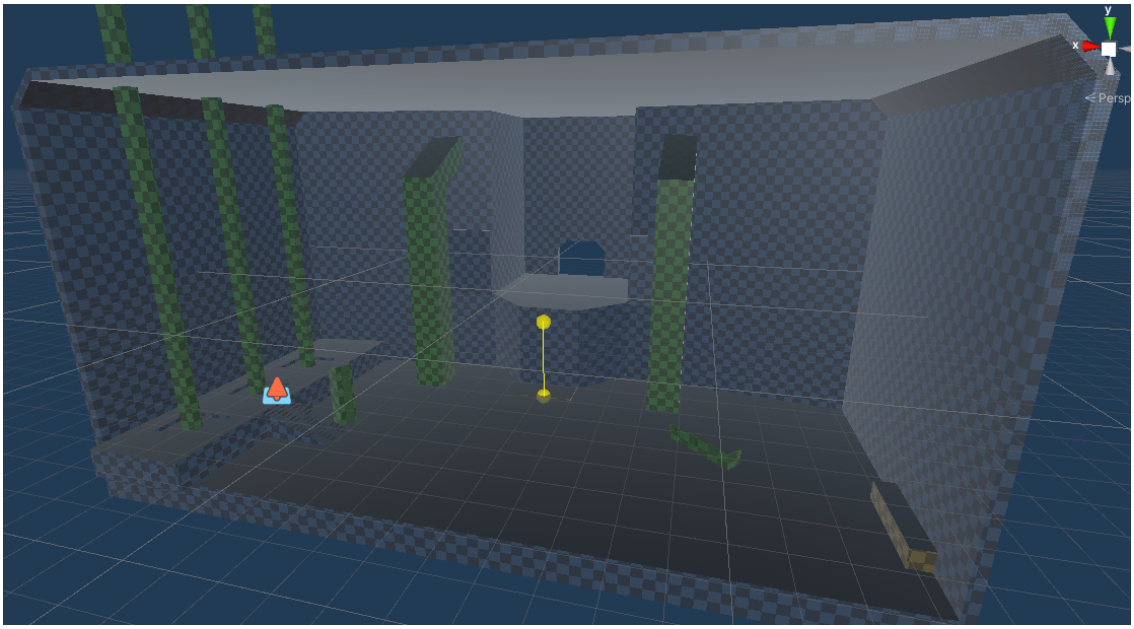


Figure 73: The level from fig 72 imported into Unity with materials and extra objects applied.

8. Evaluation

8.1. Achievements

Hopefully, following the software lifecycle presented through this report should have provided enough detail to summarise the progress made in this project. An analysis of the progress made in the project against set objectives - the design document and component breakdown - will be undertaken, alongside comments on lessons learned, usability testing and evaluations, and final conclusions. Much of the evidence for the completion of components is found in the Implementation section, and will not be repeated here.

Of the categories of identified components found in the component breakdown, 19/21 (90%) were fully realised, as shown in figure 74, while the remaining two were mostly complete. Broken down further into individual components, as shown in the Component Breakdown section in figures 18 and 19, 104/108 (96%) components were finished - with three of the remaining four components, the randomly generated room halves and the hub ship, being complete but in a way differing to a large extent from the original design document - resulting in 99% of identified work being completed to an acceptable standard, plus extra work which was discovered to be necessary in production.

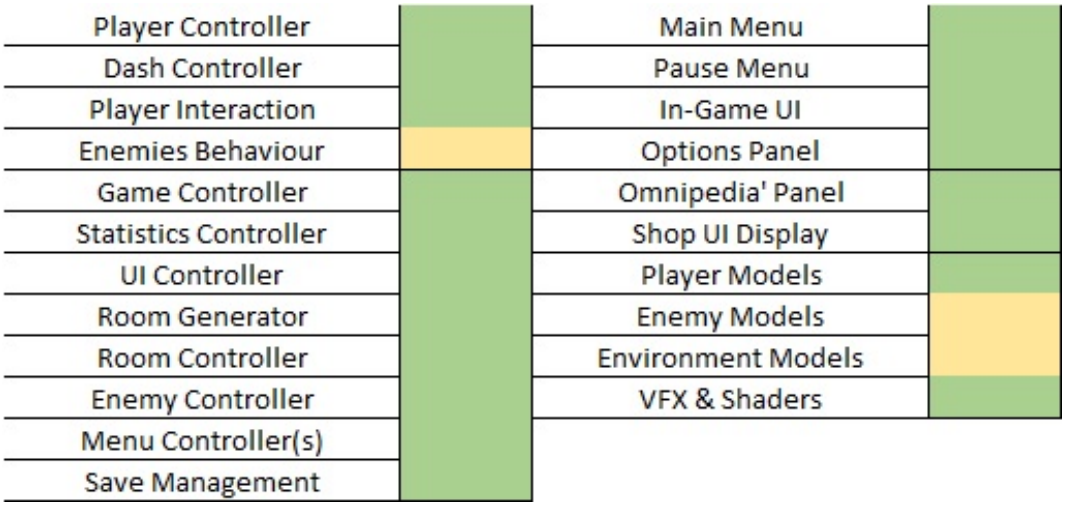


Figure 74: Breakdown of the categories the components fell under, displaying which achieved full completion. The enemies category (combined) is missing one enemy type, the Holder, while the environment category reflects the change in aesthetic explained previously.

Comparing the features of the game against the Game Accessibility Guidelines outlined earlier, 20/22 (91%) applicable criteria are present in the final game. The two left out, the ability to remap controls and ensuring that no information is conveyed by sound only, were limited by the already wildly long time required to set up the options menu, and the lack of time to set up a fully functional subtitle system which would be able to keep up with the runtime generation of the game space. Additionally, with 23/33 (70%) of intermediate guidelines implemented - the intermediate guidelines being more difficult to implement - it was important to prioritise the most critical, such as adaptive difficulty over ensuring screenreader support, and controller aim assist over audio descriptive tracks. While it would have been great to achieve a higher percentage of accessibility features in the intermediate category, given the time investment required to implement them in a meaningful way, 70% is a strong value.

8.2. Lessons

Despite the general successes of the implementation, there are downfalls. The aesthetic of the generated rooms is far simpler compared to other assets - there is a large visual discrepancy between the teleporter rooms, the player model, and the holographically styled random rooms. Given more time, transferring the project over to the High-Definition Render Pipeline to support the necessary runtime lighting, and recreating the rooms from the already built wall and floor modules would provide a major boost to the visuals of the game, and tie together the enemy, player and other environmental models created for the game.

Specific lessons learned are that more research is required into the render pipelines of the engines before choosing the tools - what seemed like a minor choice at the start of the project had large ramifications for the final product, and took a lot of time to overcome the issues inherent with the Universal Rendering Pipeline chosen.
Another lesson is that integration of the options menu - or more abstractly, any user-driven settings or variables - should be performed far sooner. Choosing to implement the options toward the end of the project resulted in a large amount of reworking of the statistics controller, the way the game loads and closes, and resulted in a far messier UI internally than if it had been made earlier - in total taking four days of work to finalise and polish.

Positive lessons were reinforced as well, however. The more free-form style of testing inherent in iterative development and 'fail often, learn faster' ideology proved especially useful in responding to the needs of the project. As every part of the project was required to be as disconnected from the rest as possible to support the random generation aspects of the game, it would have been impossible to perform structured testing without a large team available - the number of scenarios, issues arising from the random decisions made by the game at any point and already highly free-form environment of a game would simply make it impossible to form any kind of consistent structure.

8.3. Further Development

From this position, there are multiple paths down which the game could be taken. Development of more levels and enemy types is, of course, necessary - an enemy which summons other enemy minions, flying swarms of enemies, and enemies which utilise more advanced projectile weapons would be great additions, while more room layouts will exponentially build upon the level variety in the game. A deeper adaptive difficulty system would be next up - with enemies assigned a difficulty value instead of simple minimum and maximum amounts, the room generator could assemble a more directly tailored experience with a wider variety of enemy present in each room, though this would require no small amount of research into developing the perfect mathematical algorithm by which to achieve this.

Alternatively, exploration could begin into transferring the game to a new render pipeline capable of supporting the runtime generation requirements. With this, a new aesthetic would need to be established to replace the holographic environments, which would take a substantial amount of time to complete up to the standard of the other created assets.

8.4. Conclusion

In general, the project has been a success. The few failures are major, but have been circumvented to an acceptable level. Nearly every laid-out objective has been achieved, and the one objective which was not implemented in any capacity, the Holder enemy, was removed due to concerns about its obstruction to enjoyable gameplay instead of an inability implement it. Other additions, discovered during development, more than make up for the loss of that one component, and go some way to cushioning the reduced success of some other implementations, such as the aesthetics of the generated rooms. The high level of accessibility integrated into the project is another success: while some accessibility features simply come with a well-designed game, others require deep reaching integration with the game systems and revamps of existing features to conform to the guidelines.

As a second reminder, the project source files (for Unity, Blender and the ethics documentation) is found at the link in Appendix D.

References

[1] Aggregated review scores - MetaCritic. *DOOM Eternal for PC reviews*. URL: https://www.metacritic.com/game/playstation-4/doom-eternal. (accessed: 07/11/2021).

[2] Aggregated review scores - MetaCritic. *DOOM for PC reviews*. URL: https://twitter.com/SupergiantGames/status/1307744738552938496?s=20&t=PCeoLCt0iojS2E2gMv_TwQ. (accessed: 07/11/2021).

[3] Lukas Barinka. "Inverse Kinematics - Basic Methods". In: *DOCS&E* (Mar. 2002).

[4] Joakim Bergdahl et al. "Augmenting Automated Game Testing with Deep Reinforcement Learning". In: *CoRR* abs/2103.15819 (2021). arXiv: 2103.15819. URL: https://arxiv.org/abs/2103.15819.

[5] Hans Theobald Beyer. "Implementation of a Method for Hydraulic Erosion". In: *DOI* (Nov. 2015).

[6] Cafofo. *Sci-Fi Sound Pack*. URL: https://assetstore.unity.com/packages/audio/sound-fx/sci-fi-sound-pack-154257. (accessed: 11/08/2021).

[7] Barrie Ellis et al. *Game accessibility guidelines*. URL: https://gameaccessibilityguidelines.com/why-and-how/. (accessed: 22.09.2021).

[8] ESRB. *Ratings Guides, Content Descriptors*. URL: https://www.esrb.org/ratings-guide/. (accessed: 18.10.2021).

[9] FourQuarters. *The wonderful Loop Hero hits 500,000 sold in a week*. URL: https://www.eurogamer.net/the-wonderful-loop-hero-hits-500-000-sold-in-a-week#:~:text=Wonderful%5C%20role%5C%2Dplaying%5C%20game%5C%20Loop,in%5C%20publisher%5C%20Devolver%5C%20Digital's%5C%20history.. (accessed: 07/11/2021).

[10] Supergiant Games. *Hades has now sold more than 1,000,000 copies*. URL: https://twitter.com/SupergiantGames/status/1307744738552938496?s=20&t=PCeoLCt0iojS2E2gMv_TwQ. (accessed: 07/11/2021).

[11] Superdata Research GamesIndustry. *Animal Crossing: New Horizons sold an estimated 5m digital units in March*. URL: https://www.gamesindustry.biz/articles/2020-04-22-animal-crossing-new-horizons-broke-record-for-most-digital-units-sold-in-a-single-month. (accessed: 07/11/2021).

[12] Fredrik Häggström. "Real-time rendering of volumetric clouds". In: 2018.

[13] Leopold Haller and Hernan Moraldo. *Quickly Training Game-Playing Agents with Machine Learning*. June 2021. URL: https://ai.googleblog.com/2021/06/quickly-training-game-playing-agents.html. (accessed: 18/04/2022).

[14] Warren Huart and Mick Gordon. *Mick Gordon Interview - Warren Huart: Produce Like A Pro*. Youtube. 2016. URL: https://www.youtube.com/watch?v=-bsXuaIVMB4.

[15] David Kuri. *GPU Ray Tracing in Unity - Part 1*. URL: http://blog.three-eyed-games.com/2018/05/03/gpu-ray-tracing-in-unity-part-1/. (accessed: 20.11.2021).

[16] Tomoyuki Nishita et al. "Display of The Earth Taking into Account Atmospheric Scattering". In: *Computer Graphics* 27 (Oct. 1996). DOI: 10.1145/166117.166140.

[17] NVidia. *NVIDIA RTX Ray Tracing*. URL: https://developer.nvidia.com/rtx/ray-tracing#:~:text=Ray%5C%20tracing%5C%20is%5C%20a%5C%20method,to%5C%20pioneer%5C%20the%5C%20technology%5C%20since.. (accessed: 14.10.2021).

[18] Sean O'Neil. "GPU Gems 2". In: NVidia, 2005. Chap. 16.

[19] Sean O'Neil. "GPU Gems 3". In: NVidia, 2007. Chap. 1.

[20] John Opfer and Clarissa Thompson. "Even early representations of numerical magnitude are spatially organized: Evidence for a directional magnitude bias in pre-reading preschoolers". In: Jan. 2006.

[21] Jeff Orkin. *Three States and a Plan: The A.I. of F.E.A.R.* URL: https://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf. (accessed: 20.11.2021).

[22] Inigo Quilez. *Distance Functions*. URL: https://iquilezles.org/www/articles/distfunctions/distfunctions.htm. (accessed: 20.11.2021).

[23] Arthur Rahteenko. *Bakery - GPU Lightmapper*. URL: https://github.com/Unity-Technologies/NavMeshComponents. (accessed: 18.10.2021).

[24] Samuel Shaki, Martin Fischer, and Silke Göbel. "Direction counts: A comparative study of spatially directional counting biases in cultures with different reading directions". In: *Journal of experimental child psychology* 112 (Feb. 2012), pp. 275–81. DOI: 10.1016/j.jecp.2011.12.005.

[25] SteamSpy. *DOOM - SteamSpy*. URL: https://steamspy.com/app/379720. (accessed: 07/11/2021).

[26] Unity Technologies. *High Level API Components for Runtime NavMesh Building*. URL: https://github.com/Unity-Technologies/NavMeshComponents. (accessed: 03/01/2022).

[27] Unity. *NavMeshComponents*. URL: https://assetstore.unity.com/packages/tools/level-design/bakery-gpu-lightmapper-122218. (accessed: 15.01.2022).

[28] Unknown. *Finite State Machines*. URL: https://isaaccomputerscience.org/concepts/dsa_toc_fsm?examBoard=all%5C&stage=all. (accessed: 28.11.2021).

[29] VGChartz. *Doom(2016)*. URL: https://www.vgchartz.com/game/83086/doom-2016/. (accessed: 07/11/2021).

[30] Danny Weinbaum. *Genre Viability on Steam and Other Trends - An Analysis Using Review Count 2s*. URL: https://www.gamedeveloper.com/business/genre-viability-on-steam-and-other-trends---an-analysis-using-review-count. (accessed: 23.02.2022).

[31] Jamie Wong. *Ray Marching and Signed Distance Functions*. URL: http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/#the-raymarching-algorithm. (accessed: 20.11.2021).

[32] Wesley Yin-Poole. *The Four Quarters team is thrilled and thankful for the Loop Hero community as our game has pushed past 1 million copies sold on Steam!* URL: https://twitter.com/_FQteam/status/1468647408888434692?s=20&t=e7EFYHWw_iQqbCo_45bbkw. (accessed: 17/01/2022).

A. Project Definition Form

The project definition form is available at the link below, and is also provided as images, in figures 75, 76 and 77.

`https://drive.google.com/file/d/1TCZWWj-gsaUPjdUibxILrI2UvkIhl2k5/view?usp=sharing`

Final Year Project Definition Form

Student's name
Matthew Ahearn

Course
Computer Science

Project title
3D Procedural Dungeon FPS Game

What is the project about?
This project will involve the development of a part hand-crafted, part procedurally generated dungeon / series of rooms to support an FPS mechanic with evolving abilities and persistant upgrades gained from attempting to complete the dungeons. The project will be done in 3D space and will involve my own assets created using Blender for objects and animations, and modified, publicly available sound effects.

Additionally, the project will require research into existing game design choices and methods to choose which would be best to implement into my game, including software architectures and designs to support the creation of the game, as well as actual game design choices such as the gameplay loops, balancing of gameplay and reward systems.

What is the project deliverable?
The deliverable will consist of the software component (a finished, playable and polished game), along with secondary reports on the game and software design research and processes chosen, and evaluation of the project.

What is original about this project?
This project will be an original software component made by myself. Additionally, there are very few 3D roguelike games utilising procedural generation, and the sci-fi environment would be an original setting.

Figure 75: Project Definition Form part one.

Timetable showing main stages in work plan
A Minimum Viable Project will be completed by the December Term 1 Report deadline, encompassing all gameplay systems implemented in the game to a functional level. By this point, the secondary reports on research and design choices will also be started, and an aggregation of all research performed will be finalised ready for use in those reports.
User testing will be performed by a group of 3 testers covering a range of experience and abilities with video games starting from this MVP product, and will test new/updated features as they are added.

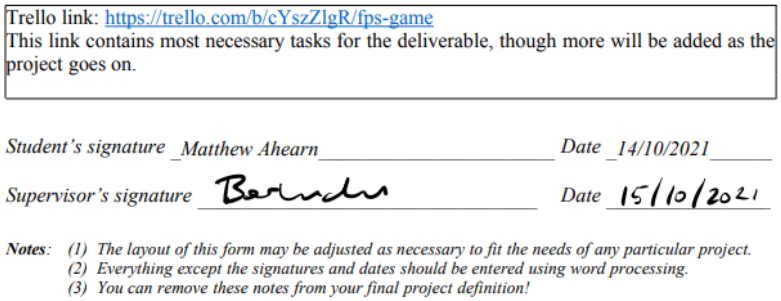
By February all 3D models and art assets will be in a finished or greybox state, ready for final polishing over the next two months before April. There should be no placeholder assets by this point. Any of the secondary reports which are not already finished should be finished by now, so that time can instead be dedicated to starting the final report.

The final project will be completed by the 18 April open demo deadline. The period from February to April will be spent polishing assets and code, removing bugs, and ensuring the gameplay experience as as good as possible.
The final report will be completed by 25 April.

Figure 76: Project Definition Form part two.

B. Usability Testing Ethics Paperwork and Transcripts

The Ethics Self-Declaration form is presented below, in figures 78 and 79:


Trello link: https://trello.com/b/cYszZlgR/fps-game
This link contains most necessary tasks for the deliverable, though more will be added as the project goes on.

Student's signature _Matthew Ahearn_ *Date* _14/10/2021_

Supervisor's signature _____ *Date* 15/10/2021

Notes: *(1) The layout of this form may be adjusted as necessary to fit the needs of any particular project.*
(2) Everything except the signatures and dates should be entered using word processing.
(3) You can remove these notes from your final project definition!

Figure 77: Project Definition Form part three.

The Usability Test Outline document is available at the following Google Drive link as a word document:

`https://docs.google.com/document/d/1k7uVSQSE6srOf5WFTetGvO66xnzWClH5/edit?usp=sharing\&ouid=103227884882966632138\&rtpof=true\&sd=true`

The participant information sheet is available at the following Google Drive link as a word document:

`https://docs.google.com/document/d/17eMvbjlcd8aWZKn1PxYG7Ndmj-O3XtNB/edit?usp=sharing\&ouid=103227884882966632138\&rtpof=true\&sd=true`

The recruitment message sent to all approached participants is available in the following Google Drive link as a word document:

`https://docs.google.com/document/d/1PDhBmdYUndk4qtx6mlLcAByEnt4zVBO9/edit?usp=sharing&ouid=103227884882966632138&rtpof=true&sd=true`

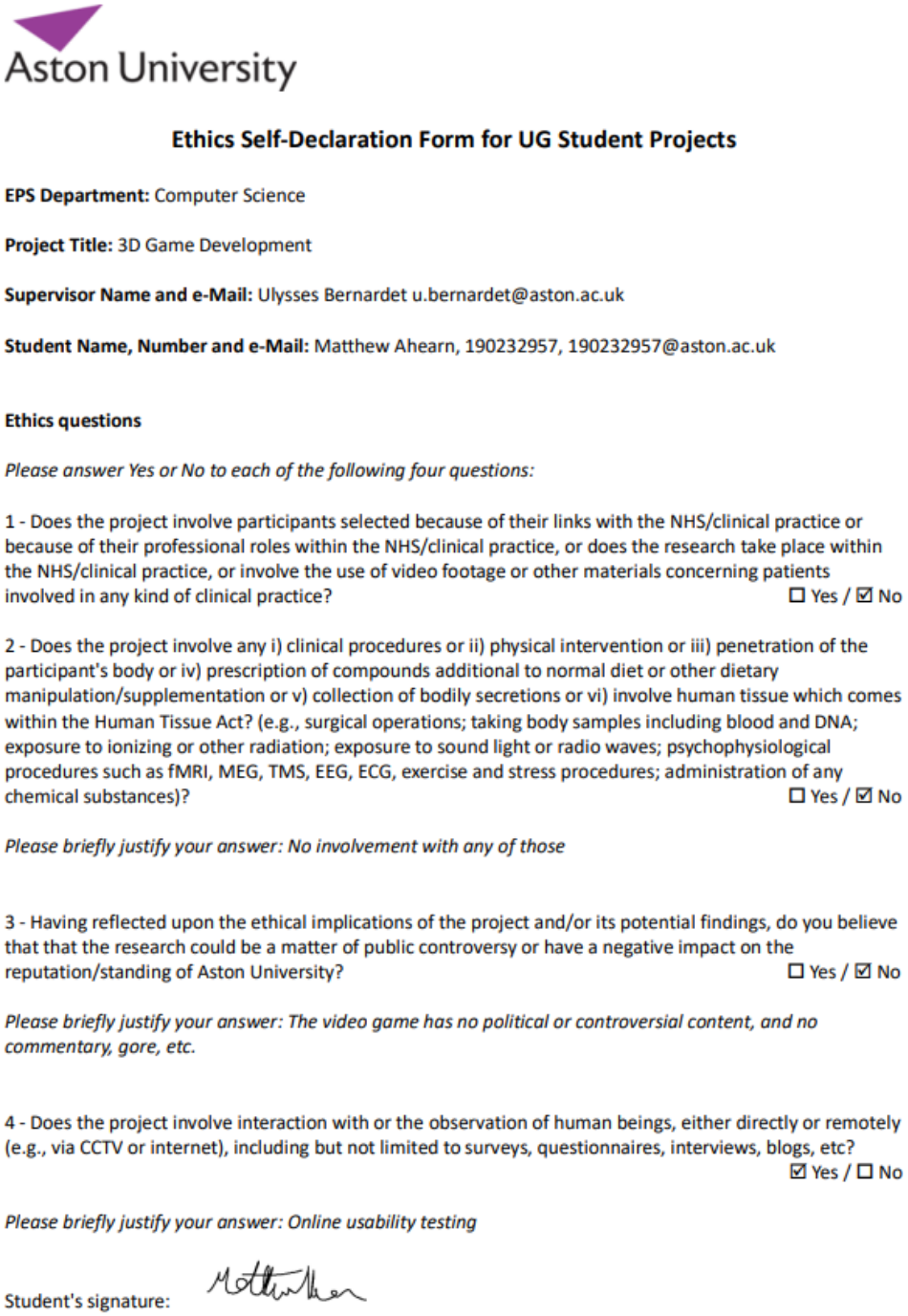
The four participant Consent forms are available in the following Google Drive links as PDF documents. Due to the lack of printing availability for the participants, their signature was photographed by them, and attached onto the forms they filled out digitally. The forms are also available as images in figures 80, 81, 82 and 83.

`https://drive.google.com/file/d/141BmnOXbYeMQCoyC5NGxXA7hVkWONDKG/view?usp=sharing`

`https://drive.google.com/file/d/1-J4Mr4pfPupB5PwMxtO-yaIbbx3nuGmL/view?usp=sharing`

`https://drive.google.com/file/d/1aGcMRcEoojSa1roS2XNDxgRjZFI6pMg2/view?usp=sharing`

`https://drive.google.com/file/d/1lQwOytP-6xwiHXS_ZnTnc-Acf6BLssaW/view?usp=sharing`



Aston University

Ethics Self-Declaration Form for UG Student Projects

EPS Department: Computer Science

Project Title: 3D Game Development

Supervisor Name and e-Mail: Ulysses Bernardet u.bernardet@aston.ac.uk

Student Name, Number and e-Mail: Matthew Ahearn, 190232957, 190232957@aston.ac.uk

Ethics questions

Please answer Yes or No to each of the following four questions:

1 - Does the project involve participants selected because of their links with the NHS/clinical practice or because of their professional roles within the NHS/clinical practice, or does the research take place within the NHS/clinical practice, or involve the use of video footage or other materials concerning patients involved in any kind of clinical practice? ☐ Yes / ☑ No

2 - Does the project involve any i) clinical procedures or ii) physical intervention or iii) penetration of the participant's body or iv) prescription of compounds additional to normal diet or other dietary manipulation/supplementation or v) collection of bodily secretions or vi) involve human tissue which comes within the Human Tissue Act? (e.g., surgical operations; taking body samples including blood and DNA; exposure to ionizing or other radiation; exposure to sound light or radio waves; psychophysiological procedures such as fMRI, MEG, TMS, EEG, ECG, exercise and stress procedures; administration of any chemical substances)? ☐ Yes / ☑ No

Please briefly justify your answer: No involvement with any of those

3 - Having reflected upon the ethical implications of the project and/or its potential findings, do you believe that that the research could be a matter of public controversy or have a negative impact on the reputation/standing of Aston University? ☐ Yes / ☑ No

Please briefly justify your answer: The video game has no political or controversial content, and no commentary, gore, etc.

4 - Does the project involve interaction with or the observation of human beings, either directly or remotely (e.g., via CCTV or internet), including but not limited to surveys, questionnaires, interviews, blogs, etc? ☑ Yes / ☐ No

Please briefly justify your answer: Online usability testing

Student's signature:

Figure 78: First part of the Ethics Self-Declaration form.

The transcripts for the four Usability Testing sessions are available in the following Google Drive links as PDF documents. As a note, the middle of the transcripts - between the opening introduction and confirmation of consent, and the interview - has been removed, as no useful data was gathered from those sections, unless otherwise specifically noted in the transcripts. The transcripts are also available in the subsections below.

`https://drive.google.com/file/d/1yub0BXO4ouuip6J7NdCtix7wCIhleGAa/view?usp=sharing`

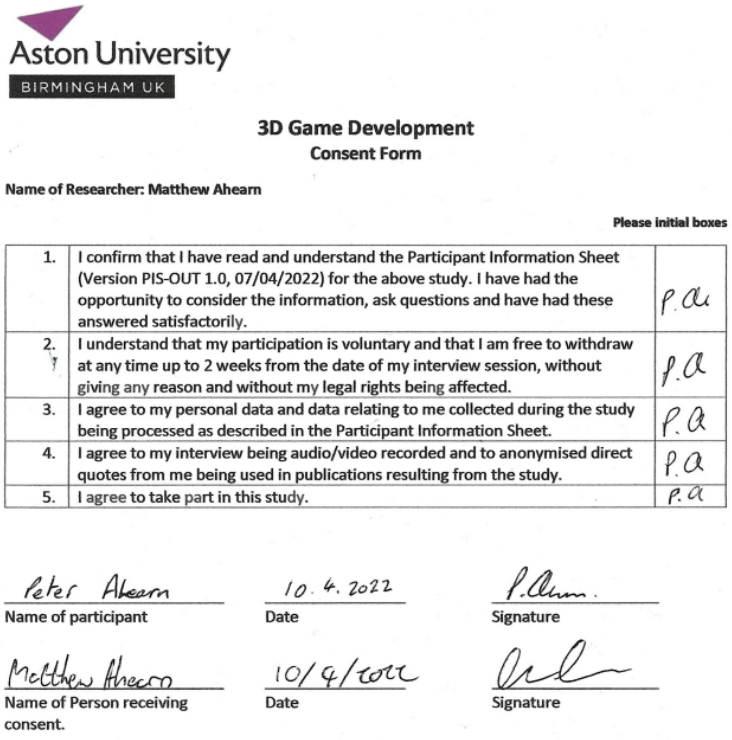
`https://drive.google.com/file/d/1MWkIPOMRatIxCfxDCZ6Y5Pjwb-aQ7yAY/view?usp=sharing`

Supervisor's signature: Bernadette

Date: 24/04/2022

Please submit this form as part of your Term 1 Progress Report. ***If any of the answers are "yes", you will need to submit an application for ethics approval as per the instructions provided on the BB site for CS3010.***

Figure 79: Second part of the Ethics Self-Declaration form.



Aston University
BIRMINGHAM UK

3D Game Development
Consent Form

Name of Researcher: Matthew Ahearn

		Please initial boxes
1.	I confirm that I have read and understand the Participant Information Sheet (Version PIS-OUT 1.0, 07/04/2022) for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.	P.A
2.	I understand that my participation is voluntary and that I am free to withdraw at any time up to 2 weeks from the date of my interview session, without giving any reason and without my legal rights being affected.	P.A
3.	I agree to my personal data and data relating to me collected during the study being processed as described in the Participant Information Sheet.	P.A
4.	I agree to my interview being audio/video recorded and to anonymised direct quotes from me being used in publications resulting from the study.	P.A
5.	I agree to take part in this study.	P.A

Peter Ahearn	10.4.2022	P.Ahearn
Name of participant	Date	Signature
Matthew Ahearn	10/4/2022	
Name of Person receiving consent.	Date	Signature

Figure 80: Consent form.

https://drive.google.com/file/d/1r-S1qHAN87KtI9oN5Altz8Nv7DFXK_Yg/view?usp=sharing

https://drive.google.com/file/d/1BAPO_uYDW7FlOBTe9qNpzhSX5WNRD_ot/view?usp=sharing

B.1. Interview 1

Researcher (R): Welcome to the Usability Test. Before we begin, I need to inform you that this session will be recorded, that an anonymous transcript will be created as soon as possible, and that direct quotations may be taken from this session for use in the study. Do you have any

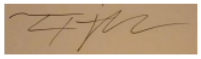
Aston University
BIRMINGHAM UK

3D Game Development
Consent Form

Name of Researcher: Matthew Ahearn

Please initial boxes

1.	I confirm that I have read and understand the Participant Information Sheet (Version PIS-OUT 1.0, 07/04/2022) for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.	**T.T**
2.	I understand that my participation is voluntary and that I am free to withdraw at any time up to 2 weeks from the date of my interview session, without giving any reason and without my legal rights being affected.	T.T
3.	I agree to my personal data and data relating to me collected during the study being processed as described in the Participant Information Sheet.	T.T
4.	I agree to my interview being audio/video recorded and to anonymised direct quotes from me being used in publications resulting from the study.	T.T
5.	I agree to take part in this study.	T.T

Trent Taylor_____ 10/04/2022_____ 
Name of participant Date Signature

Matthew Ahearn 10/04/2022 
Name of Person receiving consent. Date Signature

Figure 81: Consent form.

Aston University
BIRMINGHAM UK

3D Game Development
Consent Form

Name of Researcher: Matthew Ahearn

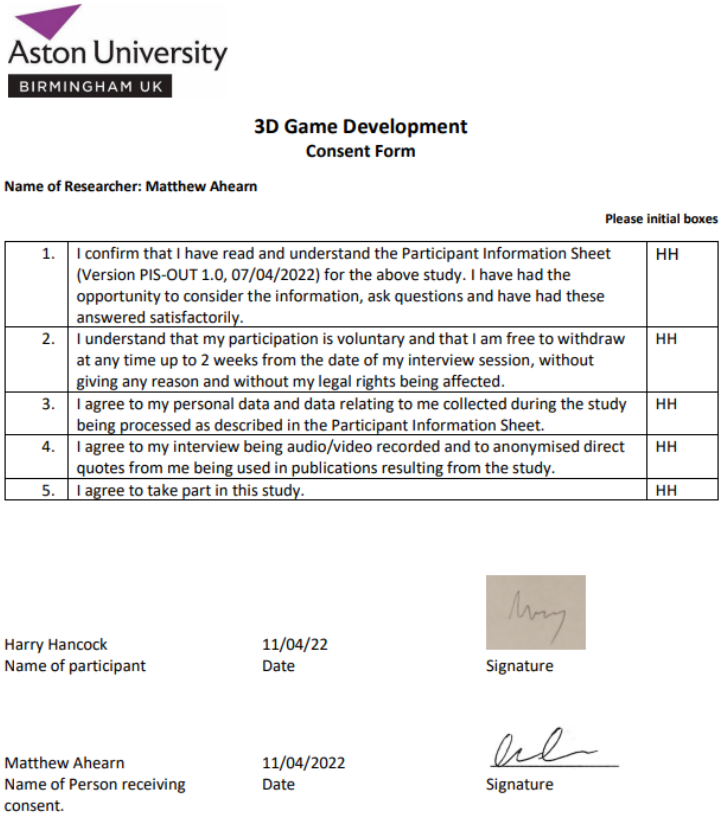
Please initial boxes

1.	I confirm that I have read and understand the Participant Information Sheet (Version PIS-OUT 1.0, 07/04/2022) for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.	**LT**
2.	I understand that my participation is voluntary and that I am free to withdraw at any time up to 2 weeks from the date of my interview session, without giving any reason and without my legal rights being affected.	LT
3.	I agree to my personal data and data relating to me collected during the study being processed as described in the Participant Information Sheet.	LT
4.	I agree to my interview being audio/video recorded and to anonymised direct quotes from me being used in publications resulting from the study.	LT
5.	I agree to take part in this study.	LT

__Ludwig Thorben_____ __10/04/2022_____ 
Name of participant Date Signature

Matthew Ahearn 10/04/2022 
Name of Person receiving consent. Date Signature

Figure 82: Consent form.



Aston University
BIRMINGHAM UK

3D Game Development
Consent Form

Name of Researcher: Matthew Ahearn

Please initial boxes

1.	I confirm that I have read and understand the Participant Information Sheet (Version PIS-OUT 1.0, 07/04/2022) for the above study. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily.	HH
2.	I understand that my participation is voluntary and that I am free to withdraw at any time up to 2 weeks from the date of my interview session, without giving any reason and without my legal rights being affected.	HH
3.	I agree to my personal data and data relating to me collected during the study being processed as described in the Participant Information Sheet.	HH
4.	I agree to my interview being audio/video recorded and to anonymised direct quotes from me being used in publications resulting from the study.	HH
5.	I agree to take part in this study.	HH

Harry Hancock — Name of participant — 11/04/22 — Date — Signature

Matthew Ahearn — Name of Person receiving consent. — 11/04/2022 — Date — Signature

Figure 83: Consent form.

questions, or understand these terms and consent to the recording?
Participant (P): I understand and consent.
R: I am sharing my screen with you now then. Do you see the screen-share and hear audio?
P: It's working fine, I can see.
R: I would like you to give directions on starting the game from this main screen. Do you prefer the use of a keyboard and mouse, or a controller?
P: I prefer keyboard. Can you open the options menu with the mouse?
...
Transcript of the recording has been deleted between this initial segment and the interview, as no relevant information is gained from it.
...
R: How challenging would you say you found the tasks?
P: I don't think they were very challenging. The game is very simple to understand, but I can't say how hard the actual gameplay is without playing it personally.
R: Were there any points where you were stuck, even momentarily?
P: After killing the first room, I didn't notice the reward thing drop. I only saw it a bit later after you looked at it.
R: How would you rate the challenge of completing the tasks?
P: Maybe a 1 or a 2 out of 10.
R: Okay, thanks. What issues did you find starting the game?
P: No issues. The animation after starting was pretty cool, and it all seems well explained on the menu how to start and what every button does.
R: Great. I know it's difficult to tell from observations only, but how would you say the default control scheme fit your preference?
P: It seemed okay to me. I noticed the second attack was tied to the B key on a controller, which

seems a bit odd, but the keyboard scheme looks about the same as any game I've played.
R: Would you suggest any alternatives?
P: I don't use controller, so I couldn't say, but none for the keyboard.
R: Thank you. What thoughts do you have about the UI design and flow?
P: The UI seemed well designed mostly. I liked the main menu and pause menus, they looked quite sci-fi, the aesthetic fit. The in-game UI was nice as well, everything was placed where I would expect to find it and it looked very modern. The upgrade menu seemed a bit weird, was it just circles around the buttons?
R: It was just circles with a gradient; a bit different to the assets used for the other menus. How did you feel about the flow of the UI?
P: What do you mean by the flow?
R: How easy did you find it to navigate the UI, did it make sense where everything was located?
P: Oh, yes, that all seemed fine. Maybe a confirmation before quitting the game, or an option to quit to the main menu?
R: Thanks for those suggestions. Having seen the game played, can you describe fully how the main weapon functions?
P: I think so. Holding down left click charges it up, and there's a bit at the end where you can get a critical hit if you let go. The right click fires a spread of bullets, and the mouse aims while WASD moves. Oh, there was a dash too, but I didn't use it. Control?
R: Yeah, left control. Could you also explain what each enemy you encountered does?
P: There was the little things on the floor, they seemed to just run at you a lot. And a few flying robots, one of them just fired missiles, and I saw a red one which had more health. And one with shields. And there was the really big one that threw bombs around.
R: I think you got it all. Finally, do you have any further comments relating to the usability of the game?
P: I think in general it's pretty good. I wouldn't have known some of the controls without looking at the options menu first though, like the dash. And the UI for the upgrades menu was a bit weird, looking different. I also think that at first, it's difficult to see the orb that gets dropped, until you know you have to look for it. Other than that, I did like how everything worked, it made sense and I think if I was playing I'd have gotten what I needed to do faster than watching your viewpoint.
R: Thank you for the feedback. Anything else you want to add before I end the recording?
P: No, that's all.
End of recording.

B.2. Interview 2

Researcher (R): Welcome to the Usability Test. Before we begin, I need to inform you that this session will be recorded, that an anonymous transcript will be created as soon as possible, and that direct quotations may be taken from this session for use in the study. Do you have any questions, or understand these terms and consent to the recording?
Participant (P): Yeah I consent.
R: Can you see my screen?
P: It's all there, I hear it too, it's loud.
R: I'll turn that down for you in a minute. Do you prefer a keyboard and mouse, or a controller?
P: Keyboard and mouse please.
R: Okay, let's get started.
...

Transcript skips here due to lack of relevant material, to 5:31 into the recording.
P: Go stand underneath that flying thing there.
R: The elevator? Got it.
R stands underneath the elevator, which pushes the player through the floor into the abyss
R: Oh, that shouldn't happen.
P: (laughing)
R: I'll just restart here, and we can try that again.
P directs R to stand underneath the elevator 5 more times, but each time the player is pushed upward onto the elevator instead.
. . .
Transcript here skips to the interview, due to lack of relevant material
. . .
R: Okay, now that's all done – I have a few questions I need to ask you. How challenging did you find the tasks?
P: I found the bit where we fell out of the map challenging, I don't think that's beatable. But really I didn't find it very challenging, it didn't break very much and the enemies didn't seem too hard.
R: Did you find it challenging understanding what needed to be done to progress the game?
P: Oh, no, it seemed quite easy. Just kill all the guys, pick up the green black hole and exit through a door, then go again.
R: Did you find any issues while starting the game?
P: No, but I wish I checked the controls before starting. I saw the menu, I just hoped I could see controls in-game or something.
R: That's a good point. Does the default control scheme fit your preferences? Try positioning your hands on your keyboard to cover the controls you know about in the game if it helps.
P: They seemed normal to me. Some games have the interact key on F instead of E, but I guess it doesn't matter.
R: Would you make any changes?
P: I don't think so, personally. There should be an option to change the controls yourself though.
R: That might be hard to implement, but I'll make sure it's thought about carefully. What thoughts do you have about the UI design and flow?
P: The UI design was good, I liked the hexagon shapes for the buttons. The other menu was a bit different but I think it was good anyway, I think in-game menus should look different to pause menus and etc. They also were very easy to navigate, not something really complicated like those Ubisoft games these days.
R: Can you describe to me fully how the gun works?
P: Yeah, there's that dash which lets you dodge stuff which is pretty fun. Holding down left click powers up your bullet, and there's that second at the end of the charge where you can shoot a super shot. And there's the shotgun attack which seemed a bit overpowered.
R: I'll take note of that last comment. Could you also describe to me how each enemy you encountered functions?
P: Sure. There were a few of that one enemy, which shoots missiles. One that just shoots missiles right at you, one with shields, and one with shields that also fired a lot of missiles from above at you. And there was the big caterpillar thing, it fired bombs, though it didn't seem to aim at you which is weird.
R: Yeah, it has some troubles getting around the maps currently. Finally, do you have any further comments?
P: It seemed pretty good to me. That part where you fell through the floor was funny but you probably should fix it. And maybe balance the enemies a bit more. It was easy to direct though, and most things seemed clear. Maybe put the controls on-screen while playing, so you don't need

to check the options menu first.
R: Thank you for that feedback, it will be very useful. I'm ending the recording now.
End of recording.

B.3. Interview 3

Researcher (R): Before we kick this off, I need to make sure you're fine with the fact that this session will be recorded, and an anonymous transcript will be made of the recording before the deletion of this recording. Quotes may also be directly taken to support the study. Do you understand and agree with this?
Participant (P): I understand and agree.
R: Okay, I'm going to share my screen with you now then.
...
Transcript of the recording has been removed between this initial segment and the interview, aside from some select moments, as no relevant information is gained from it.
...
On loading the second room and killing some enemies, some of the crab bots are found to be flying in place above the ground.
P: Are those supposed to be flying?
R: No, that's not supposed to happen. Just ignore it for now.
...
Skipping to interview
...
R: How challenging did you find those tasks?
P: Not especially, I thought it was pretty obvious. Having played games before I found it quite easy to start the game.
R: What issues, if any, arose while starting the game?
P: No issues I can think of.
R: Does the default control scheme fit your preferences?
P: Yes, it was very similar to other games of this type I have played.
R: Would you suggest an alternative controls?
P: No.
R: What thoughts do you have about the UI design and flow?
P: Yeah, the UI design was fine, it suited the style of the game and was very intuitive.
R: Okay. Having seen the game played, would you be able to describe how the main weapon functions fully?
P: Yes. Would you like me to?
R: Yes please.
P: So, the main gun has a basic attack which is just firing normally by pressing left click, you can also charge that attack by holding left click to create a more powerful shot, and you can right click to fire a shotgun burst which was helpful for clearing large groups of enemies. As far as I know, the right click did not have a charge up mode.
R: Sure. Would you be able to describe the functions of each enemy type you encountered?
P: Yes. You had the basic missile robots, they seemed to come in two flavours, both fired slow moving projectiles which targeted where I was standing so I could dodge. One flavour was shieldless, and one flavour had shields. There was a really big guy, who seemed to drop like mines when I got too close to him, which did a lot of damage. There were little bug guys, really small guys who would just charge you from across the map.

R: Yeah. And do you have any further comments relating to the usability of the game?
P: No, I found it very usable. Some minor bugs, but nothing that overly affected gameplay.
R: Okay, thank you.
End of transcript

B.4. Interview 4

Researcher (R): Hey, welcome to this testing session. As noted in the outline form, the session is being recorded, though the recording will be deleted after an anonymous transcript is made. Do you understand and consent? The recording is currently on.
Participant (P): I understand, thanks.
R: Would you prefer to direct a keyboard and mouse, or a controller?
P: A controller please.
...
Skipping to the interview, as no relevant conversation is held between these points
...
R: So how difficult would you say you found those tasks?
P: Very obvious, simple, searching around the two rooms and then it was obvious to go through the main door to get into the game.
R: Did you find any issues starting the game?
P: Nope. No actual issues.
R: Does the default control scheme fit your preferences?
P: Yes, it's almost... it's very standardised with other games that I'm aware of, a bit like Doom or any first-person shooter, so it all fitted in with that.
R: Would you suggest any changes?
P: Not for myself, seeing as I used a controller rather than a keyboard, so that suited me very well.
R: What thoughts do you have about the UI design and flow?
P: The user interface was all sort of stock looking, so when you went through it it was obvious you had the options, start or quit, and the pause menu seemed standardised I should say, instead of stock.
R: Would you be able to describe how the main weapon functions fully?
P: Yeah, it was all pretty straight-forward. You had the trigger, you could energise the weapon, or you could quick fire, which wasn't so destructive so you had to do it more. Then you had the other right bumper which fired the shotgun. Nice and straight forward, and simple. Jump in, and use it.
R: Can you also describe each enemy?
P: There was the basic enemy, which was shooting straight at you. Then you had the other one which sent missiles sort of at you, looped over and aimed at you. Then you had the sort of boss one, the big one which was getting in your way. That's basically it I think.
R: Do you have further comments relating to the usability of the game?
P: No, it was nicely controller, the motion was fluid, having had issues on proper games, fully built ones lately, this one all seemed to flow together, it all worked quite nice. There were no, sort of, nasty surprises.
R: Thank you.
Transcript end

C. Game Acessibility Guidelines

The guidelines checklist used in this project is available at the following Google Drive link, in the form of an Excel spreadsheet:

https://docs.google.com/spreadsheets/d/11zPSUxVvsVbAvkGuWO8SW1Nm0tHsywKs/edit?usp=sharing\&ouid=103227884882966632138\&rtpof=true\&sd=true

D. Project Source Files

The Google Drive link below will download a zip file containing the Unity project (important - use Unity version 2021.1.21f1, later versions break and earlier versions are untested), the blender source files and FBX files and the ethics forms and project definition form.

https://drive.google.com/file/d/1qQzdG0Bx0RUz2wcE60-W3c8cJax0hlkG/view?usp=sharing