

A-LEVEL COMPUTER SCIENCE PROGRAMMING PROJECT

Random level generation game // HARSH BRAHMBHATT - 5124

WATFORD GRAMMAR SCHOOL FOR BOYS - 17655

CONTENTS

CONTENTS	2
ANALYSIS	7
Description of the problem	7
Computational methods & why the project is amenable to a computational approach	9
A computational approach VS Physical traditional games	9
Required Computational methods to be used	10
Stakeholders	12
Client Interview	13
End-users' Questionnaire	16
Analysis of data	19
Research: Existing solutions	22
Features of the proposed solution	28
Limitations	29
Requirements & Success criteria	31
Software & Hardware requirements	37
DESIGN	40
Problem Decomposition	40
Program structure	49
Data flow	50
Algorithms	52
Main menu	52
Player	58
Environment generator	74
Enemies	83
GUI & Usability features	95
Main Menu	95
Settings	96
Pause menu	98
Weapons menu	98
Website feedback form	100
Post development test plan	101
Overall program user testing (Beta testing)	101
Testing requirements	101
Graphical user interface testing	103
Client acceptance testing	105

IMPLEMENTATION	106
Introduction	106
Development Plan	107
Main menu	108
Gvr modules	112
Player	112
Progress Review 1	159
Environment generation	160
Progress Review 2	181
Enemies	181
Weapons	202
Progress Review 3	208
UI	208
Website	219
Progress Review 4	234
Completion: Final Review	234
Validation	235
Future maintenance	239
EVALUATION	241
Testing to inform evaluation	241
Overall program user testing (Beta testing)	241
Testing requirements	243
Graphical user interface testing	255
Graphical user interface comparison with designs	258
Robustness testing	261
Client acceptance testing	263
Evaluation of solution	265
Evaluation of project completion	265
Possible improvements to meet unmet criteria	267
Evaluation of usability features completion	269
Possible improvements to meet unmet usability criteria	269
Maintenance & future development limitations and issues + possible improvements to aid	270
BIBLIOGRAPHY	272
APPENDIX	274

BLANK PAGE

BLANK PAGE

BLANK PAGE

ANALYSIS

Description of the problem

VR is described to be the future. VR technology can be incredible in terms of the level of immersion players can achieve, although still fairly new VR technology is incredibly advanced in terms of player experience but still has many limitations. One of the biggest problems with VR headsets is the lack of access for players due to the extremely high prices of these headsets, meaning that fewer players can experience the technology. Google posed a solution to this in 2014 by releasing the google cardboard, a simple cardboard VR headset alongside cardboard kits which are kits so that people can even DIY make their own headsets/viewers. The key idea of cardboard VR is that it makes good use of your mobile phone, since such a large percentage of people, and at least households, have access to a mobile phone. Being made out of cardboard and the fact that they can be built yourself means that overall the product is extremely cheap, and therefore a much greater amount of players have access to the VR experience.

Young people often spend their free time playing some sort of video games as a form of escaping their lives into another world, more than 90% of American children play video games and overall 2 in 3 millennials play too. However, not only young people, but it's seen that over half of all Americans (58%) play video games. By taking advantage of this highly accessible platform, Cardboard VR, it's possible to allow people to enhance their experience by diving into an even more immersive version of a hobby they already love.

Cardboard VR games currently, still being quite new, do not fully take advantage of VR capabilities and therefore do not create as good a VR experience as possible. Virtual Reality is completely based on the fact that as humans, there are vast amounts of things that we'd all like to do or experience but are not able to do so due to our bodily limits. My VR game aims to implement exciting mechanics that create an experience which the user would never be able to do in their lives ordinarily. For example, by implementing grappling gear as a means of manoeuvring, the player can experience the thrills of travelling at high speeds in 3 dimensions without feeling the extreme g-force levels associated with this means of movement in the real world which would ultimately cause a human to at least pass out or possible death.

Overall, using Cardboard VR means that all people are able to access and enjoy such a key development in the latest technology by using it to view and interact with arguably their favourite hobby from a whole different perspective, experiencing what it's like to immerse themselves one step closer to the future.

Computational methods & why the project is amenable to a computational approach

A computational approach is ideal for video games since otherwise, they would have to be manufactured as traditional board games. By taking a computational approach you are also able to create an appealing Graphical User Interface (GUI), due to the opportunities you have to design graphical models, animations and even include sound using a computer. In terms of an algorithmic design/approach, the problem lends itself due to the interactive elements of the game and how the game has to dynamically interact. Interactions with enemies in the form of combat or manoeuvring using 3D grappling gear which depends on the position of the player and the physics (gravity etc.) which applies at the position in space. The algorithms must be able to interpret inputs for these interactions and allow the game to react appropriately and also consistently - this is possible due to the mathematical nature of algorithms.

A computational approach VS Physical traditional games

Feature	Physical	Computationally
Environment	2-dimensional board.	3-dimensional word space with varying and realistic terrains and environments which can be changed extremely quickly to travel to completely different worlds if the player desires.
Movement	Physically move a piece to where you want to move, using your hand on the board in 2 dimensions.	Using a controller with buttons or keys on a keyboard and mouse to map movement with controlled velocity in all 3 dimensions.
Combat	E.g. Knock a player's counter off the board if allowed to.	Use of many different virtual weapons with varying attributes and special qualities - extremely realistic.
Visuals & Sound	Still image on the board. No sound.	Provides video interaction. Able to display every known colour, as well as special effects such as particle effects. Provides sound for greater immersion (Overall both senses fully captured)

Therefore, we can come to the conclusion that developing a game is much more suitable for a computational approach.

Required Computational methods to be used

Computational method required	Description	Justification
Abstraction	By abstracting unnecessary details the overall project requirements will be shorter along with the development time. This means that the much more important features and functionalities can be focused on with more time given to them so that they are of much higher quality.	Unnecessary information and requirements can be abstracted since the algorithms will only take into account and understand the essential data. Simplifying the entire project also makes it clearer what I need to do in terms of the features of the game, then other aspects of the game which I may not be responsible for can be ignored. Irrelevant features should also be abstracted so that the project does not become cluttered with many complex algorithms as this will heavily increase the overall complexity of the program and so, will not run as well due to it being very expensive. Lots of small features, e.g. lots of Heads Up Display (HUD) information would also draw the player away from the main aim, possibly being annoying /distracting, therefore should be abstracted.
Decomposition	Taking a more modular approach - used to break the project up into many much smaller sub-tasks, each can be completed individually and even by different people if required.	My game being very object-oriented, decomposition is essential to be able to break down aspects of the game such as player, enemies, environment etc, into their own respective objects/tasks. Each of these classes can also be broken down into smaller tasks, that being the attributes and methods associated with each one. This means that each one can be worked on and tested separately. Decomposing my problem into smaller tasks means that parallel processing in the form of pipelining can be applied since each task is at its most basic level.
Pipelining	Pipelining makes use of parallel processing - decreasing development time by completing multiple tasks at the same time if they can be.	Pipelining can be applied after decomposition, for example, enemies can be developed separately from the player, and therefore simultaneously. This means that they can both be developed at the same time so that afterwards, the interaction between them can be quickly tested, instead of doing one, which is left completely alone while the other is developed before they can be tested together.

Pattern Recognition	Using algorithms to recognise patterns in anything from environmental changes to inputs from the user, then analysing these to produce specific results.	Pattern recognition will be required for analysis of the environment, for example, for the enemies' movement, the enemies movement AI would need to be able to see patterns of where it can and can't walk and act accordingly. The system will also need to be able to recognise and understand the pattern I implement for the automatic generation of each level so that each level/floor has some sort of structure each time.
Visualisation	Providing a Graphical User Interface (GUI) allowing the user to easily interact with the system but also appear more visually appealing to produce a better user experience, which is less static.	My system's GUI will aim to be as intuitive as possible to provide a seamless experience for the user, i.e. not confusing, no time should be spent trying to figure out how to work the GUI. GUI interactions will be simple clickable buttons/tabs to ensure that it is clear to the user how to navigate the game.
Performance modelling	Being able to analyse and act on the efficiency of the system through mathematical models to then be able to improve the efficiency best possible, e.g. analysing to find a function that has a big O of $O(2^n)$ and improving it to $O(n^2)$ which would be much more efficient.	Performance modelling will be an essential computational method to ensure that the system is not very processor or data expensive, i.e. the processor can smoothly run the game on all machines well. For example, generating terrain may need to consider generating it along 2 dimensions therefore most probably generating a worst-case expense of $O(n^2)$, which is not too bad for generating the main environment, but could be better. When implementing, I must analyse the expense to ensure it is firstly not too expensive, and secondly to see if there are any improvements that could be made that will make the program more efficient.
Digital Data storage	Making use of databases allows efficient storage of large amounts of data securely, especially if encryption is implemented.	For storing the data, for example, how many floors a player has completed, can be exported and stored in an external database using languages such as SQL. This secure and accurate data storage will be essential for ensuring that the app can quickly and easily receive/retrieve data, being an essential part of its functionality. Another use of a database would be for the secure storage of sensitive user data from their account data.

Stakeholders

There are 2 main groups of stakeholders to my system. The first being the client who's essentially hired me as a developer for their company, which wants me to develop the game for market release. The second group of stakeholders to my system would be the end-users, i.e. the player who will end up playing the game / using the system regularly.

I have chosen my friends as well as close family (younger siblings, cousins etc) to be the primary stakeholders for the end-users/players group, although many others will also be considered to try and replicate the variety and size of a real-life video game player base, this is also to try and aim for the greatest amount of range as possible in terms of age, however, this factor (age range), as well as some others, will have to first be considered with the games company client to meet their preference.

In terms of determining the needs of both client groups, I will enforce 2 methods of identifying the needs as well as the main general inputs and outputs.

For the games company client, I will interview them to get their honest and open thoughts about what they want the game to be. Although time taking, since there will only really be one representative of the games company to interview, it'll be worth getting the direct opinions and thoughts of the client company. An interview will also allow room for discussion and building questions from responses, which is not possible with questionnaires for example, and therefore lots of honest information can be gathered quickly.

For the end-users who will play the game, I will roll out a questionnaire, this will be most appropriate as it will allow me to gather information as direct responses of the people who will ultimately play the game from a much larger group of people than is possible to interview. Ultimately, the game will aim to have a very large player base and therefore it is important and most useful to try to replicate that by getting the opinions of as many players as possible. An electronic questionnaire means people who I may not even personally know or cannot meet due to logistical constraints can still be useful in providing information. Overall, the large amount of data that can be collected using a questionnaire will be most useful to directly represent the opinions of the end-users.

Client Interview

1. What general role would you like me to take in this project?

Your main role here is, of course, game developer, I'd like you to undertake the job of our development team, developing the systems and programs of the game.

2. How much experience do you have with computer systems, game design and programming?

We're of course, unable to develop the game due to our lack of experience with code and programming, however, we'd be happy to look at and give our input on any systems written in simple English, still specifying exactly what they do. Testing the system will be completely within our ability, of course playing the game is fine but navigating given game engines should also be within our grasp.

3. What areas of the project would you like to oversee completely and what input would you like to have?

Yeah, so you are the lead developer, however, there are a couple of aspects of the game that we like to completely take over. That's mainly the graphics and animations side of things, we understand that you're more of a developer than an artist so we'd like to focus skills towards what you specialize in, that is the systems and programming.

- 3.1. Oh okay, so you have other personnel or a team looking after that side of things? And also, what about sound/music?

Yes, graphics and animations will be designed and developed by another team, if you could add basic graphics and animations, to show they, like, work, as placeholders for the actual ones that would be great. Sound and music is a similar story, if you do have any experience working/implementing music then it would be nice if you could include some, but is not necessary as we will be sourcing music from elsewhere.

4. What approach would you like me to take, i.e. using a game engine, or developing one from scratch, etc?

Right so, this is actually something we'd like to leave mostly up to you. We have little knowledge of what the best approach would be to developing the game. The approach must meet our standard of quality, though. The game must be 3D and 3D models/graphics designed using software such as blender must be usable as an asset/model in the game. Oh and, of course, the approach must also allow the game to be converted/developed for a Cardboard VR viewer. From what we know, game engines such as Unity and Unreal engine allow you to meet such requirements easily, and especially in the short amount of time that is available, they may be viable approaches. However, as I said, we lack knowledge of all the different ways or approaches, so we'd like to leave this up to you to decide.

5. What age demographic would you like the game to appeal to?

The age range should be quite broad really. As you already know, we'd like everyone to experience VR so we want the age range to be as big as possible. This also means you know, we would have a greater scope for potential buyers of the game, a greater player base. This is required for the greatest profit whether we decide the game is suitable to be sold for a price or for maximising in-app purchases. Oh yes, however, one thing we must ensure, I nearly forgot, the cardboard viewer is rated a 7+ age restriction. Although we are unsure of the officialness of this or if it matters that much, we must ensure a positive reputation of the company. So if we had to give an age range something from around 7 or 8 years to the age of like, young adults, 25 years.

5.1. What kind of difficulty should be aimed for then, to appeal to this wide age range?

Well, that's a weird, difficult question, games are all about balance really, although the age range is large, this shouldn't change the fact that as long as the game is well balanced. The difficulty level that should be aimed for depends wholly on the game mechanics, for example, weapons shouldn't be so easy to use that the game is too easy and therefore boring. Grappling hooks allow the player to move quickly, however, they shouldn't be so quick that they are almost invincible, as this makes the game too easy and boring. Try to find balance within the gameplay so that all age ranges are accounted for. In the future, we may ask you to implement different difficulty levels to further improve this, however, that is not essential.

6. The game is to have randomly generated levels, which are different from each other. How different would you like these levels to be and in what way? Should they play completely differently too?

Yes, we want a key aspect of the game to be that there are an infinite number of levels so the game never ends, however, the player must not know this. Therefore, the levels should differ in terms of design/layout of the game objects which make up the environment, enough so that the player does not feel like they've played the exact level before. Levels looking fairly similar is fine, but not too much. We don't want levels to be too big, so multiple rooms within a level is fine but there should really be no more than around 3. In terms of how each level will play, there should not be too much of a difference and the mechanics should remain the same. Gameplay should only differ due to the difference in environment. The player should be able to capitalise on, for example, very tall trees, allowing them to gain more height using the grappling hook compared to a town type area with just buildings.

7. With the Cardboard VR having only one button for inputs, do you have any ideas already of how the inputs will work?

Again, we'd like to leave this up to you, inputs/controls should feel natural and dynamic, however, with the cardboard VR viewer only having the one button, we advise you to try to implement as many mechanics as possible with automatic triggers, if that's the right term. What I mean is, things like maybe shooting, using a sword should be done automatically if the player is aiming in the right direction and is in range. This means that the few inputs can be saved for things that cannot be implemented in this way. We'll leave this up to you to try out different options and confirm with us what controls we finalise.

8. What kind of weapons/items systems should be implemented?

Yeah, so there don't have to be a crazy amount of weapon systems or different items, however, they should all work differently so that the game plays out differently depending on the things you use. Abilities shouldn't have too major of an effect, but they should just keep the gameplay feeling fresh, different and fun. There can be a number of different primary weapons, and within this, there can be different rarities to unlock. Items can be maybe scavenged from levels or earned from killing enemies, or by unlocking them by levelling up.

9. How much emphasis should be placed on the storyline aspect of the game?

No, actually, we don't really want to focus on the story aspect of the game, we believe players have more fun playing the game than becoming invested in a storyline, an extremely simple one would be nice to include, but not essential.

10. What kind of colour scheme would you like for the game to have?

Since we're trying to appeal to all ages, we want the colour scheme to just be simple, making it too bright may make the game appeal more to younger children but possibly deter older people from playing the game. And possibly visa versa if it is made too realistic. You don't need to worry about graphics as discussed, and colours can easily be tuned later, just keep it simple.

11. How will you make use of/sell/distribute the final product, are there any specifics such as this that I need to be aware of?

The final product should be an application, of course, and therefore we aim to distribute the game through the Google Play Store and the Apple App Store. However, as you know, we would like to have a website which advertises the game and links people to where it can be downloaded.

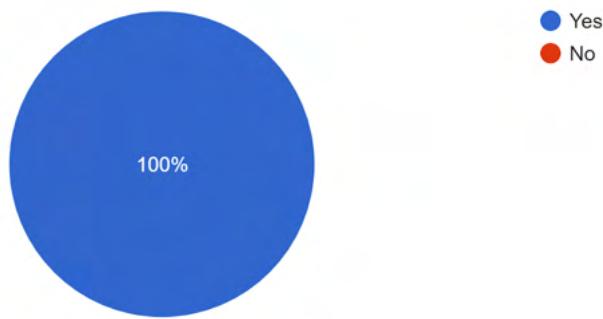
Cool, so the game as well as a website, anything else that needs to be developed?

Actually, yes we did have one more desirable application, a companion app for players showing their account stats, this would be great but if it's not possible in the time, it's not essential and can be left out.

End-users' Questionnaire

1.

Have you ever played a Dungeon crawler / Level type game? A 2D example of a dungeon crawler is shown below, essentially, all enemies in a small floor must be killed before moving on to the next floor.
10 responses



2.

If you answered yes, what is your 1 most favorite thing about these types of games? (can be anything, even the simplest of features like movement or combat)

9 responses

The mystery of figuring out what is about to happen

I enjoy the rewards obtained from putting in so time and effort into a game after finishing a battle

Combat and controls

Puzzles they are pretty cool

I like dodging attacks and feeling like a hero

Combat

The variety and large amount of levels

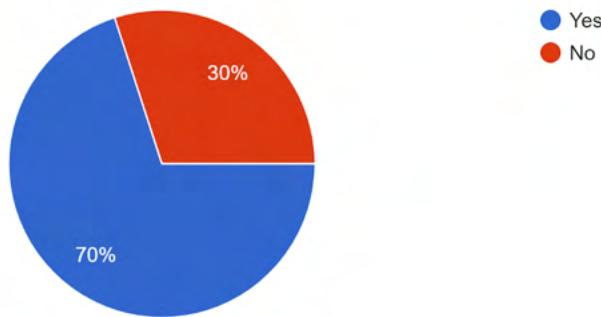
New cool swords and weapons

Movement

3.

Have you ever played a VR game?

10 responses



4.

Harsh Brahmbhatt
Watford Boys Grammar School

5124
17655

If you answered yes, what is your 1 most favorite thing about the VR gaming experience? (Can be very simple features, but try be specific)

6 responses

I haven't answered yes but, VR is so cool since it has immersive gameplay and feels like you're in the world.

Realistic

Movement mechanics that simulate real life as well as things that aren't possible in real life

The immersive experience of how close it is to real life.

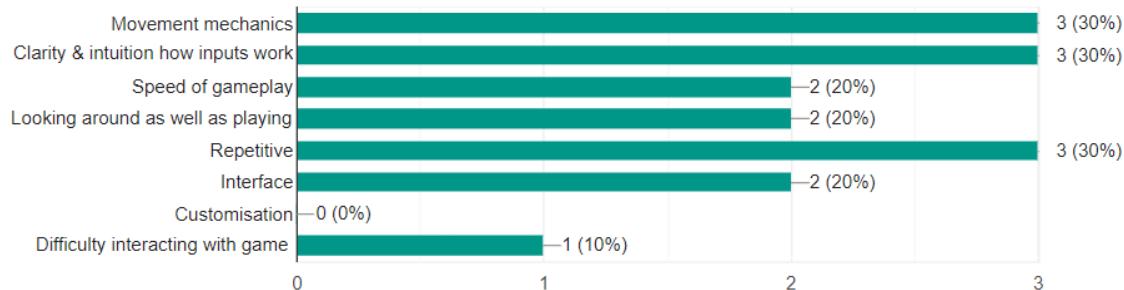
- I think it's very cool how you are able to surround yourself and "pretend" like you're in the game itself and being able to interact as a character itself rather than someone who is controlling the character

Immersiveness

5.

Again, if you answered yes on Q3, what is your least favorite thing(s) about VR games, general features you think need to be focused on more. Choose 1 or 2.

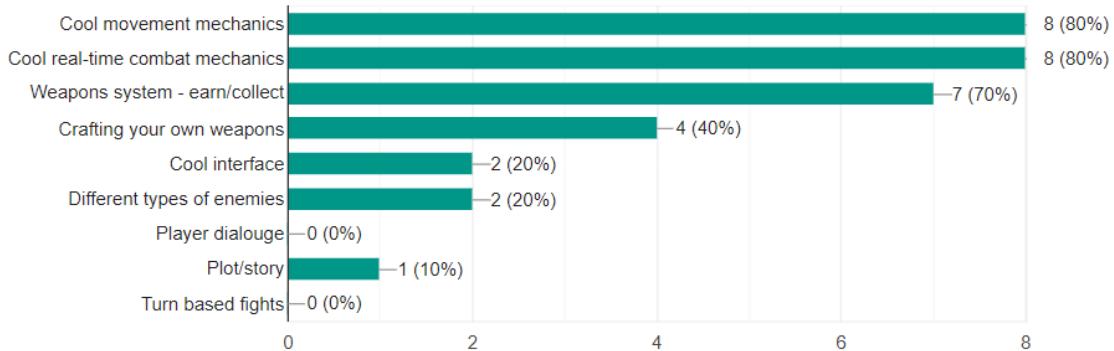
10 responses



6.

Which of the following features would you MOST like to see in this 3D VR dungeon crawler that would attract you to the game? Choose around 2 or 3:

10 responses

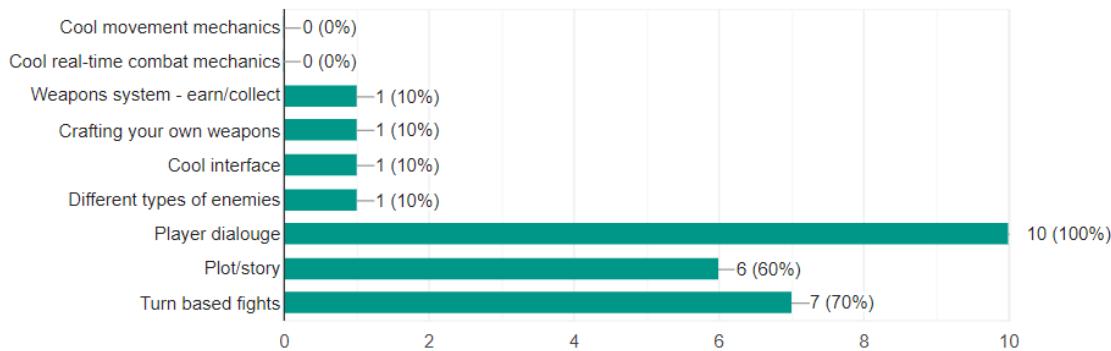


7.

Which of the following features would you be least interested in or would not want to see?

Choose around 2 or 3:

10 responses



8.

Any comments/suggestions/requests you have that you'll think improve/elevate the quality of the game?

3 responses

I think you can try to make the end goal of the game to make the player feel like a god with insane abilities and weapons but balance it with just as strong enemies. This makes the game have a strong power fantasy

It would be cool to be able to have the game as customizable as possible

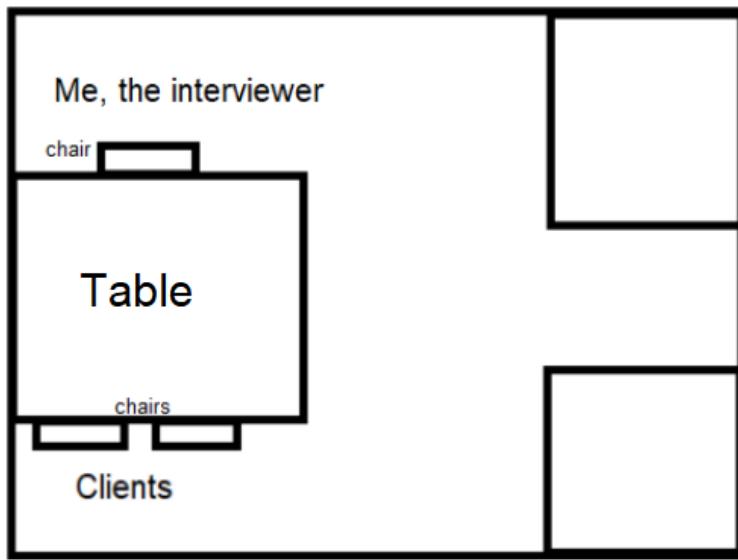
- UI should be simple and easy to use. (appearance doesn't really matter it just shouldn't be hard to navigate through what the user wants to do) - Long intensive fights that have rewards based on how long the user has spent doing that challenge would be a + for me personally (like a grinding game)

Analysis of data

1. Client Interview

Shown below is the room plan of the interview, using a public coffee shop, it was ensured the interview was fairly relaxed meaning that the clients were comfortable

enough to happily share their views and provide information.



From the interview I was able to gather sufficient information that can be summarised as the following to produce a basic initial requirement list:

- **Logistics:**
 - I make up the development team to develop the game
 - Clients are happy to look over systems written in English (a form of pseudo-code)
 - Need to only develop the systems, clients do not need me to develop any graphic models or animations or music
 - Can be developed using a game engine or any means
- **Gameplay & Players:**
 - Large age range, 7-25 years
 - The game should be balanced and this will form the difficulty
 - Difficulty levels are a desirable feature
 - Maps should be randomly generated such that players never feel like they've played the exact level
 - Players should be able to use the environment to their ability, therefore having randomly generated levels means that levels may play out differently
 - Game mechanics should be the same across all levels, only the levels themselves should change
 - Maximise use of automatic inputs since Cardboard VR viewer only has 1 button
 - A fair amount of balanced weapon systems that all play out differently
 - The storyline is not important

2. End Users' Questionnaire

Avid gamers filled out the questionnaire and results can be summarized as the following:

- All the opinions of the respondents are valid since all of them have played a dungeon crawler style game before.
- Only 70% of respondents had played a VR game before, this supports the point that VR games are not as accessible as they should be, or not as much as other types of games.
- Those that had played VR games before believed that the experience was realistic and immersive supporting the reasons to why making a VR game is beneficial, however, from the next question I asked what the potential problems with VR were. Really spread out, there was no general trend and this means that all of the aspects of will need to be carefully considered.
- Most popular and least popular proposed features were gathered:
These 2 questions, and questions from earlier allowed me to develop a basic initial requirement list:

Features to focus on:

- Mystery of progressing through the game
- Rewards
- Real-time Combat
- Diverse weapon systems
- Large amount of levels
- Movement mechanics
- Intuition of controls
- Game speed
- Game interactions
- Interface
- Making game less repetitiveness

These features should all be focused more and refined as much as possible

Features to not include / focus less on:

- Player dialogue
- Plot/story
- Turn based fights.

These features should either not be included or included to a minimal extent such that even if they were not there it wouldn't matter.

- A couple of comments suggested that the game should make the player feel powerful while still keeping the game balanced, having lots of customization and also a really intuitive UI. Another comment I found interesting was that rewards should depend on how long the user has fought an enemy. I guess this makes winning the fight more of an achievement which links to the other comment about making the player feel powerful. This not being a required question meant that these few people went out of their way to suggest these features. They probably reflect the opinions of all the respondents and gamers in general, therefore should be considered even more with a very high priority and focus.

Research: Existing solutions

Last epoch



Last epoch by Eleventh Hour Games (2019) is a standard dungeon crawler role playing game, however it has been received extremely positively due to how well some of the features that you'd expect to see in a dungeon crawler have been uniquely implemented. These are features I can learn from and then adapt ideas into my own game.

An example is the classes and skill tree system. With my game, I was initially thinking of implementing different weapons and abilities the player can choose from to make their own class, however Last epoch has a classes system which means that you choose one from a few different classes and it will have dedicated weapons and abilities. Then, using the skill tree, the player can expand the base class by upgrading weapons and abilities as well as unlocking new ones. Having distinct player classes like this, all with different types of weapons and abilities means that the player can find a playstyle within a class and then expand it to better fit their own playstyle. Multiplayer has been announced but not released yet for Last epoch however, another key benefit of player classes is that different classes work well together and a team should be encouraged to have a variety of playstyles among them. This makes the game more dynamic and brings more strategy to the game, which was something the end users commented on as something they would like. So if I were to implement multiplayer, this would be a key benefit. Instead of having the player choose a weapon and abilities, being forced to choose these within a player class of a certain play style means the player has to be more creative and ultimately makes the game more fun.

A key aspect of the game which I hadn't really considered was the boss fights. Each floor finishes with a boss fight and this is beneficial since it gives the player a much higher sense of accomplishment rather than moving on to the next level by simply defeating a certain amount of the same enemy. The boss at the end of the floor is what the entire level builds up to so there is a sense of anxiousness and adrenaline when you reach the boss as it all comes down to this one fight. Another comment from one of the provisional end users was that the player should feel a sense of accomplishment after a tough fight so this is certain something that I should aim to implement. It also gives the player something to look forward to in a level.

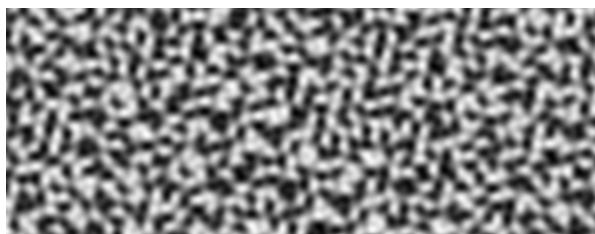
Last epoch also has a *mana* system, which works kind of like a stamina system with some extra influence. You need a certain amount of *mana* to do anything from attack to abilities, and then for some special abilities/attacks, you'll need a certain, high amount of mana to perform/use. This can be implemented into my game as a simple stamina system and have a certain threshold to be able to do certain things such as attack or use the grappling hook. This will be beneficial in maintaining a certain game speed, something which the end users suggested I focus on as well as a really important aspect of any game anyway. It will also make sure the game feels challenging enough to be fun.

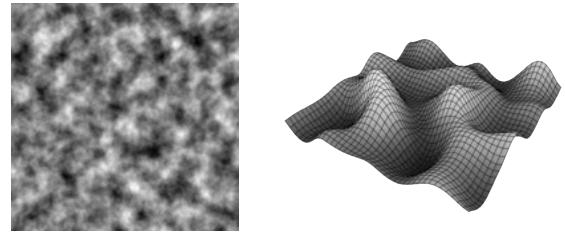
In terms of the levels, Last epoch has an extremely interesting random map feature. The player can actually make different choices and complete levels in different ways and this opens up multiple timelines of getting to the end of the game, and the player gets a random map depending on the way they complete the previous level. Although this seems contradictory, the randomness comes from the fact that the player does not know which paths will lead to which level and so two different players will complete the game through different routes and levels just due to the fact they make independent decisions. Although this is not the type of randomness I am aiming for, it is an interesting system whereby decisions affect your route in the game and so is definitely an approach to consider to keep the randomness aspect.

Minecraft



Minecraft, released by Mojang as their debut game in 2011, is highly applicable to my game due to its procedural terrain generation. Minecraft worlds are infinitely large, randomly generated worlds, generated using procedural generation. The world is meant to randomly form realistic terrain to create landscapes such as mountains, caves, hills, flats. This is done using noise, and combining multiple layers of noise. Noise is a way of randomly generating values for every point on a map. However, a regular static noise map is completely random and if it were used to generate terrain it wouldn't be very realistic, so quite unusable. Instead you can combine layers of different noise functions such as Perlin noise to generate smoother, much more realistic terrain. The Perlin noise function, used by Minecraft, takes a pair of x and z axis values inputs, and will return a pseudo-random value between 0 and 1, which can then be used for the y value (height) of a node or block, depending on what these x, z values are. This value appears to be random, however, 2 sets of x, z values which are very close to each other will return 2 values which are also very close to each other. This means that smooth hills, etc. can be created if you consider 2 points on a terrain, their heights should be similar for the terrain to seem realistic.





Shown is a regular static noise map and then a Perlin noise map respectively, note how the Perlin noise is smoother and more gradual than the static noise map which has some bright white points right next to dark black points, i.e. random and not gradual.

Furthermore, if we consider Minecraft worlds as finite maps that are fairly large, we can consider how such a large map can be generated. Instead of generating the entire map, Minecraft has to, and does, procedurally generate terrain as the player visits it - there is no point generating terrain which the player is many miles away from. This is done using chunks, a chunk is a section of the map which contains a certain amount of, in Minecraft, blocks and then chunks are loaded into RAM and the players FOV as they get closer to it. Of course, Minecraft worlds are infinite so this is actually essential - an infinitely large map cannot be loaded into RAM. But this is incredibly useful as it means only a certain, set amount of chunks, and therefore blocks, are loaded into RAM at any given time, Minecraft has 10 although can be changed by the user. This makes the space (+time) complexity of running minecraft much lower, meaning that it's much more efficient.

So overall, I can learn 2 main terrain features from Minecraft that will help me to develop interesting environments in my game. The Perlin noise function can be used to procedurally generate realistic terrain and, like Minecraft, I can further develop this by combining multiple layers of noise. This will also allow me to randomly generate terrain which builds upon the random, infinite levels feature of my game. And secondly, to make my game more efficient, I should aim to procedurally generate terrain in chunks, however the levels may not be large enough to gain full benefit of this, the same concept can be applied to generating the infinite levels, Minecraft worlds have 1 infinitely large map, whereas I need to generate an infinite amount of separate maps. Only the current and/or the next level, for example, that the user is playing should be loaded into RAM.

The Elder Scrolls V: Skyrim VR



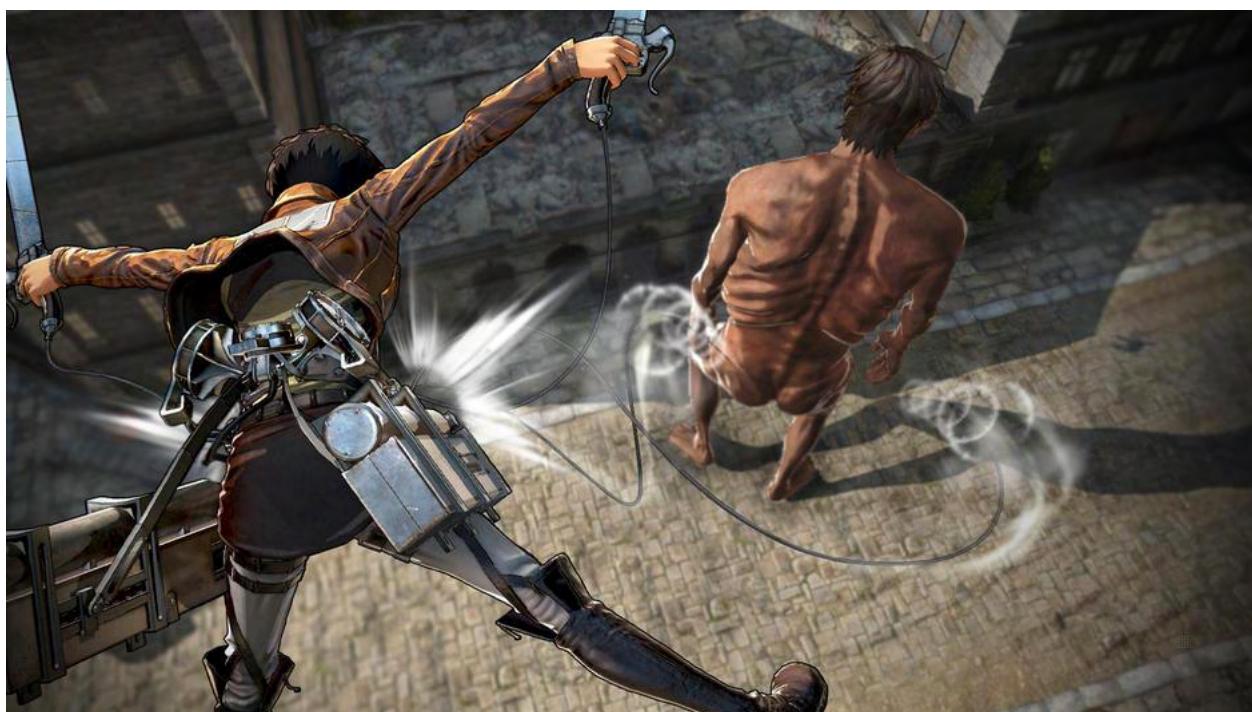
The Elder Scrolls V: Skyrim VR was released as a VR adaptation from the non-VR version of the game in 2017. Skyrim VR is a really dynamic game since it was originally a fully fledged role playing game which was then developed into a VR version. Therefore, unlike some VR games, Skyrim VR looks to incorporate all the features of a full RPG into the VR version, which is also what I am aiming to do. This means that the developers had to find ways of incorporating all the essential features of the game in a way that would allow them to work with a VR approach.

Skyrim VR has an interesting approach to VR which I can take inspiration from. Instead of the default just one, Skyrim VR has 2 movement methods, although one of them is still quite new and not as good as it could be. The first method, the original movement method when the game released, is the point-to-teleport method, this is where the player aims their reticle at where they want to move and then presses a button to teleport to that point. Therefore movement is not realistic and quite clunky, which is certainly a drawback, however it does provide the user full control of when and where they want to move, however some viewing angle limitations do restrict the ability to move wherever you want slightly. The other method that Skyrim VR has is the move forwards method in which the player holds down a button, pointing down the direction they want to move in. This means that the player actually moves instead of teleporting which means it'll feel much more natural and therefore immersive. However, this method also has its own set of drawbacks, obviously, it means the player has to hold down a button to move which

could be frustrating and limit the amount of other inputs the player can make at the same time, however, if the movement was automatic for example, it could mean the player cannot stand still, limiting the amount of control they have over their movement. Another drawback would be that the player has to move in the direction they are looking, this isn't too big of a problem as it's natural to move in the direction you are looking however, during combat this would mean dodging and retreating maybe a bit harder and/or counter intuitive.

Overall, I can take these movement methods acknowledging their flaws and adapt them into a method which aims to be intuitive, natural and retains the immersiveness of VR. This could be using automatic movements, since the Cardboard VR viewer has limited inputs, and then maybe incorporating a system where the player can stop if they are looking at a certain angle.

Attack on Titan 2



Attack on Titan 2 is a story mode game where you fight and kill giant titans and has extremely interesting and applicable game mechanics. Specifically the grappling gear, the game has implemented 2 grappling guns attached to the body of the player. This is an interesting and unique mechanic compared to the standard one grappling hook coming out of a single gun. Along with the basic grappling mechanics, the game has implemented a gas/fuel system which is used to boost and accelerate the player as they grapple, however this is a limited resource and has a cool down. This makes the grappling a lot smoother and slicker, ensuring the gameplay remains fast. In terms of the physics of the grappling hook, the game actually uses a variety of different physics models depending on the situation and parameters of the player at a given moment. Sometimes, grappling onto buildings the player may swing with real life type

physics, however, if using the gas boost, the player may accelerate directly to the point that they have grappled in a straight line instead of swinging. Grappling to the titans, the game uses a aiming mechanic where the player aims with a scope to where they want to attack. Then the player will grapple directly to that point instead of the grapple behaving in a Newtonian way, i.e. no physics apply. This is a benefit as they player can quickly accomplish what they want to do without having to consider physics, however, due to the fact that physics apply sometimes and not other times, can make the game feel unnatural and the player has to get used to when physics will and won't apply since they must grapple at a height accordingly. Overall, I feel the game could definitely benefit from having a uniform system where physics always does or doesn't apply so that the game feels consistent. Different speeds and accelerations can be experimented with to ensure fast gameplay and I liked the idea of the fuel/gas boost system tied with it's cool down to make the game more challenging and at the same time fun.

Features of the proposed solution

My game will be a 3D Dungeon crawler / level-based game developed for mobile devices designed to be played using a Cardboard VR viewer in, of course, VR. The player must defeat all enemies on a level to progress onto the next level.

In terms of the actual levels, I will create a system that can automatically generate random maps/levels each time using procedural terrain mesh generation to create a more realistic, uneven floor terrain as well as randomly generating other environment features that the player can use to their advantage. Desirably, there should be different types of biomes within or across the different maps which have similar types of features however, levels should still be distinct from each other.

The player will have some sort of weapon of their choice, selected through a weapons/class system as well as some sort of mobility item/ability, for example, a grappling hook to allow for more fast-paced, action-packed combat. These items will have restrictions, for example, weapons such as guns will need to be reloaded or swords will break and need replacing, this is to add difficulty and complexity to the game so the player has to think more strategically. The mobility gear will have a fuel system and this could mean the player has to refuel at certain stations, or it could provide a time limit for each level, where the player loses if they fail to kill all enemies before their fuel runs out. The player's movement will be optimised to minimise the amounts inputs required - since Cardboard VR games only have one input, some sort of automatic movement, perhaps in the direction the player is looking, should be implemented to save inputs for other functionalities.

The enemies will be extremely large giants to allow the player to target them easily however, the enemies must be dynamic in the way they react to taking damage. I.e. the enemies could react differently to where they are stuck, maybe they can only be killed if slashed at the neck or, if the player hits their ankles, the enemies fall and become immobile for a period of time. This will force them to use different methods of mobility such as the grappling hook to make the combat

more engaging and add layers to the difficulty. The enemies have invaded the town in each level and the player must save as many residents as possible. The enemies will also move automatically, with a mix of random movement as well as moving directly towards the player or other civilians, if they are in range, for example. If they are in very close range, the enemies will damage/kill civilians or the player. Desirably, some sort of machine learning AI could be developed here, to allow the enemies to slowly learn in which direction they can and can't move to avoid obstructions such as buildings. This would increase level difficulty as the player progresses through the game.

The game will need to provide additional UI such as the main menu, a pause menu and controls/other settings. Desirably, the game should also allow the player to log in to save stats and progress, encryption/security would have to be considered here and an API such as Google's could be used.

Desirable, but certainly not essential features include mainly a website as well as a companion app. The website would be used to promote/advertise the game while also providing a link to where it can be downloaded. The companion app would allow the player to log in and will show all the player's stats, such as levels completed, enemies killed etc. This allows players to compare with friends and make the game a bit more competitive. The last, extremely ambitious feature is multiplayer, however, this is only desirable and can be considered in future updates if it cannot be completed in the first cycle of the game's development. Multiplayer would implement some form of TCP/IP to allow players to complete levels together and this would make the game more fun as it would allow players to come up with new strategies and methods to playing.

I then asked my clients if they agreed with these features, of which all agreed.

Limitations

The main aspect of my game is the fact that it is a VR game and this can pose potential problems. Eye strain and nausea are examples of health issues that can be caused by VR games. Myopia is a growing problem nevertheless, rising from only 25% of the population in the 1970s to over 40% by 2000, however, there are concerns, especially for children, that VR could further accelerate the rate of increase of near-sightedness. However, this potential risk can be minimised if not mitigated completely by slowly introducing someone to VR who has never tried it before, and overall, limiting use time for the headset and therefore game. Another practical limitation is the fact that due to the high level of immersion, players can feel completely one with the game and become unaware of their surroundings. This can lead to damage to their body or surrounding furniture if the player does not have sufficient space in their surroundings. The effects of all these potential limitations can be reduced by including an age rating and having warnings before the game starts which could include advice to limit use time and making sure there is sufficient space around you before playing, as well as being careful and aware of your surroundings.

My VR game can also pose compatibility limitations, this is because to use Cardboard VR, you of course, need a mobile phone and with an Android phone, your phone's android version should be 4.1 or higher, and with an iPhone, your phone's iOS version must be 8.0 or higher. This poses a certain level of limitations since people without these devices and the required software versions will simply not be able to use cardboard VR and therefore play the game.

Another limitation is possibly people who suffer from visual impairment or vision loss. These people have a decreased ability to see and distinguish between different colours and objects. These people would most likely not be able to play or at least, fully enjoy the game to its fullest extent for which it is designed. This is not ideal, as the clients would like this game to appeal to as many people as possible, however, to improve their experience, I can make sure to make game objects, labels and other UI easy to see and use by using distinct/contrasting colours.

Requirements & Success criteria

No.	Requirement	Justification	Measurable success criteria		
			Desired	Acceptable	Unacceptable
1	Must be playable with a GUI, in VR with Cardboard VR viewer.	The client would like a VR game due to the high level of immersion and other various benefits previously discussed.	Fully functioning GUI working as a VR game, flawlessly with no bugs.	Functioning GUI as a VR game, possibly a few minor bugs/ lag.	Not functioning GUI and is not playable as a VR game.
2.1	The game must randomly / procedurally generate levels.	The game must have an infinite amount of levels so that the player never runs out, each level being procedurally means each level will always be different, i.e. the game will always feel fresh and new.	An infinite number of procedurally generated levels which are all clearly distinct from each other.	An infinite number of levels, all procedurally generated. However, some can be similar.	Less than 2 levels / levels are not procedurally generated, manually designed.
2.2	The environment must consist of different types of biomes, e.g. forests, towns etc. with specific environment features.	Different biomes mean that the player can play in different environments, allowing the gameplay to differ between them. This allows for different playstyles and gameplay in general, meaning that game will always be fun	Couple of different biomes which all seem distinctly different every time each biome is used due to randomly generated, specific environment features.	A few different types of biomes, but environment features of a biome may seem very similar each time that biome is used/generated.	All environments are the same, no distinct biomes.
2.3	Levels must consist of 1-2 rooms.	Classic, most successful dungeon crawlers often consist of multiple rooms to add an element of adventure and discovery to the game as the player explores through rooms.	Levels consist of 1-2 rooms, the rooms follow a certain style but are distinct / different.	Levels consist of 1-2 rooms, however, rooms may seem very similar.	Incomplete levels, i.e. not even 1 room properly generated per level.

2.4	Levels must consist of multiple enemies which must all be defeated to go on to the next.	The game must have a main objective which is to defeat all enemies in a level to progress on to the next. A clear objective means the player can focus most on this and therefore do well, making the game more fun.	Enemies spawn at intervals, i.e. not all enemies spawn at once. Player can move on after all enemies defeated.	Enemies each successfully spawn and function as required. Player can move on after all enemies defeated.	Levels do not have any, or enough, enemies, including maybe spawn issues etc.
2.5	Levels must have a boss enemy at the end of level.	Boss fights make the game more fun as this is what the entire level builds up to, acting as a climax to the level.	Boss fight at end of level, and is different to a normal enemy.	Boss fight at end of level, may be similar to other enemies, possibly.	No boss at the end of levels
2.6	Some levels must include refueling stations.	Mobility items and weapons will need to be reloaded and they may need to be reloaded at specific stations.	Fully functioning refueling stations on some levels that work 100% of the time.	Refueling stations work most of the time, may have minor bugs.	No refueling stations on any levels or they do not work at all.
2.7	Levels must include civilians who run around randomly, the player should aim to limit casualties.	Having to save civilians gives the player a reason to fight and provides a greater sense of accomplishment seeing how many lives they've saved. But also provides encouragement to save more lives next time.	Civilians run around randomly, but generally try to avoid the enemies.	Civilians run around randomly, may not avoid the enemies very well.	No civilians, or they do not move properly or at all.
3	The game should have a free-play, practise mode.	A free play mode without enemies, or ones with limited functionalities can allow the player to practise.	Free-play includes procedurally generated levels.	Includes free-play practise mode but for only 1 level.	No free-play / practise mode.
4.1	The player must be in first person and have all basic movement, such as walking, running and jumping.	The player must be in first person to access the VR aspect of the game. The player must be able to easily move, moving is essential in completing the main objective of the game - defeating the enemies.	Player plays in first person and movement is fluid and works 100% of the time.	Player plays in first person and movement is functional but may have a few minor bugs or lag.	Not in first person, and the player cannot move properly (<50% of the time) or cannot move at all.

4.2	The player must have some sort of weapons, such as swords or guns.	Weapons are essential in completing the main objective of the game, that being defeating the enemies in each level.	A variety of weapons which all work 100% of the time.	Weapons work most of the time. May have some hit register bugs.	Only one choice of weapon and/or does not function properly (<50%).
4.3	The player must have some sort of mobility items, such as a grappling hook.	Mobility items are essential in making the game dynamic and fun, as this feature capitalises on the VR aspect of the game. As it is not easily possible to use grappling hooks in real life, simulating this in VR will make the game much more fun.	Mobility items/systems work 100% of the time.	Mobility items/systems work most of the time. May have some hit register bugs.	Mobility items/systems do not work at all or <50% of the time, and therefore unusable.
4.4	Weapons and mobility items must have a refuel/reload requirement, i.e. must not be usable continuously forever.	Having to reload/refuel means that the player has to be smarter with their decisions of when to attack and when to reload, it will also require them to pay some of their attention on these factors of fuel etc. This makes the game more challenging and more fun. At times, the player may need to visit the refuelling stations to do this	Limited fuel supply and limit on weapons before refuelling / reloading required. The player is constantly made fully aware of this.	Limited fuel supply and limit on weapons before refuelling / reloading required. May be slightly inconsistent. The Player made aware of this.	No limit of fuel or weapons and they are usable forever.
5	The game should have sufficient in game UI, such as health, fuel, etc.	This UI is essential in providing the player with enough information to allow them to keep up and think ahead, i.e. plan their next move.	UI for all required information, however, limited to ensure there is not too much UI.	All required information is shown to the player via the UI.	Not enough UI such that it harms the player's experience.
6	The game must have some weapon classes which the player can unlock and choose from.	Various different weapon systems encourage and allow different play styles and techniques for completing levels. This variety makes the game more fun and keeps it feeling new.	Various weapons classes which are all distinctly different to each other.	Various weapons classes, however, may not be as different to each other as desired.	Only 1 weapon class, i.e. lack of classes such that the gameplay is the same and becomes boring.

7.1	Enemies must attack the player and civilians if they are in range.	This is essentially the obstacles to completing the main objective, i.e. the resistance to defeating the enemies. Their attacks must be avoided, making the combat more realistic and more of a fight, making it more fun.	Enemies can accurately detect the player or civilians in range and attempts to attack them.	Enemies detect the player or civilians in range most of the time and usually attempts to attack them.	Enemies cannot detect the player or civilians in range and so do not attack.
7.2	Enemies must automatically move, generally towards the player.	As the objective of the enemies is to get in range and attack the player, the enemies will need to be able to locate the player and move towards them.	Enemies can always locate and generally move in the direction of the player.	Enemies move randomly and can sometimes locate and move towards the player.	Enemies cannot move at all.
7.3	Enemies must be able to dodge obstacles such as buildings.	The enemies should appear to have some sort of intelligence and so must be able to transverse the level avoiding obstacles to make them seem more realistic enemies.	Enemies can accurately detect obstacles ahead of them and change route accordingly.	Enemies can usually detect obstacles and take a different route, but may use some trial and error.	Enemies cannot detect obstacles and so constantly bump into them.
7.4	Enemies must react differently to different attacks and where they are struck/hit.	The enemies must be reactive and behave differently depending on circumstances, e.g. if they are attacked in the ankles, they should move slower, or fall over.	Enemies accurately react to attacks depending on where they are struck.	Enemies react according to where they are struck. May have minor detection bugs.	Enemies do not react differently to where they are struck.
7.5	Enemies must come in various different forms.	There should be 3-4 different types of enemies, each with different stats to make the game more fun, as fighting the same enemies would be boring.	Enemies come in multiple, various forms with distinct stats, attacks, etc.	Enemies come in various forms, stats and abilities may not be too distinct.	Enemies are all the same. I.e. do not come in various forms.
7.6	Boss enemies should be much harder to defeat and should look	As the entire level builds up to the boss fight at the end, this fight should be a longer, harder one to create a greater sense of accomplishment	Boss enemies are very different to and are much harder to defeat due to special	Boss enemies are different enemies and harder to defeat, e.g. due to	Boss enemies are almost the same as normal enemies and not harder to

	distinctly different.	when the player defeats it.	stats and abilities.	more health etc.	defeat.
7.7	[Desirable] Boss enemies should require a special method/strategy to defeat it.	Being the boss of the level, this enemy should be the most advanced and intelligent, requiring the player to be more strategic to defeat the enemy, making the game more fun.	Boss enemies should require a special method/strategy to defeat it.	The player must attack in different areas of the enemy's body to defeat the boss.	
8.1	The game should have a main menu when started.	A main menu is essential in navigating to the main features of the game, i.e. playing the game, options, quit, etc.	Fully functioning main menu with no bugs.	Fully functioning main menu, may have some input lag or minor bugs.	Main menu does not work, buttons don't function as intended.
8.2	The menu should have 3 options; "Play", "Settings" and "Quit".	These 3 options are fundamental to allowing the player to access all the functionalities of the game.	All buttons function as intended with no input lag or bugs.	All buttons function as intended, may have some input lag or minor bugs.	1 or more of the menu buttons do not work /function as intended.
8.3	The "Play" option should be a gateway to 2 options; playing the actual game and a freeplay practise mode.	The play option should be a gateway towards the corresponding features of the game.	Play button functions as intended with no input lag or bugs.	Play button functions as intended, may have some input lag or minor bugs.	Play button does not work /function as intended.
8.4	The "Settings" option should allow the user to change certain settings to their preference.	A settings page will allow the user to change settings such as controls or sound settings to their preferences, this will mean that the player can enjoy the game to their liking/preference.	Settings button functions as intended with no input lag or bugs.	Settings button functions as intended, may have some input lag or minor bugs.	Settings button does not work /function as intended.
8.5	The "Quit" option should exit the	A quit option is essential as it allows the player to exit the application if	Quit button functions as intended, exiting	Play button functions as	Play button does not work /function

	application.	they wish to stop playing.	the application with no input lag or bugs.	intended, may have some input lag or minor bugs.	as intended.
9	A website, to advertise the game and allow downloads.	Websites are accessible and therefore can be reached by many people. Helps advertise the game and allows players to download and play.	The website should function fully with no bugs or lag, and provide a download link for the game.	Website functions fully and provides download for the game, may have minor bugs/lag.	Website does not fully function as intended, including major bugs.
10. 1	[Desirable] The game should provide a login system.	A login system means that players can save progress and stats through their account. Can also help easily create age restrictions if needed.	Players can login efficiently when starting the game application.	Incomplete log in system, to be implemented in the future with more time.	
10. 2	[Desirable] The login system must store data using an external or cloud database.	Storing data locally is not very secure and certainly not practical to store data for potentially hundreds of people or more since the game will be publicly available. A database is much more secure and practical.	Login system uses an external/cloud solution to storing data.	Login system does not use an external database to store data. Can possibly be implemented with more time.	
10. 3	[Desirable] Data transfer with the database should be encrypted.	Encryption ensures much greater security of people's data, especially if sensitive data is to be handled.	Data is encrypted.	Data not encrypted. Can possibly be implemented with more time.	
11	[Desirable] A companion app should allow players to view stats and other such information.	A companion app means that players can login through the accounts previously mentioned and view their game stats, such as level, floors completed, enemies defeated etc. They can also compare with others.	Companion app links to the game through the user accounts to show players their stats, medals etc.	App not fully developed, possibly be implemented with more time.	

Software & Hardware requirements

I had not previously discussed hardware and software requirements with my clients who want me to develop the game so I sent them the following email:

Hardware & Software requirements

Good morning Mr [REDACTED]

I am writing to you just to confirm your thoughts and preferences about hardware and software requirements for the game.

Of course, being a VR game for a Cardboard VR viewer, I understand that the game will be for mobile devices. Furthermore, I would like to know what specific operating systems you would like me to develop the game for as this will require different modules, hardware/software requirements and approaches in general.

My choice of game engine shall also provide some limitations as devices will need certain software/hardware to be able to run games developed on the game engine. Instead, another approach I could take would be developing the game from scratch completely, although this of course also poses its own drawbacks. The first and foremost being that the quality of the game will undoubtedly a little worse compared to it being developed using a game engine.

Other than that, I assume you'd like me to develop the game to be able to run on the lowest possible hardware and software requirements that Cardboard VR games can run on, on the devices/operating systems you specify. Please let me know if otherwise.

Yours sincerely,

--
Harsh Brahmbhatt

I then received the following response:

Hi Harsh,

We've been considering your queries and have come to an agreement as to what we'd like from you.

In terms of the mobile devices we'd like the game to run on we have a few things:

- Primarily, we want the game to at least run on Android mobile devices.
- We would also like you to develop for iOS, only if there is enough time at the end of the development process.

We want to be able to distribute the game out to players as soon as possible, developing for a second operating system could hold up the development and increase the overall development time, iOS can always be developed for once the Android version is already out. We understand that Android devices are usually easier/quicker to develop for compared to iOS, which is why we've decided to prioritise it.

In terms of the game engine vs scratch approach to development:

- We would like you to use a game engine for development, even at the cost of a few extra hardware and software requirements.

We want to prioritise the quality of this game, therefore using a game engine to increase quality is definitely worth the requirements drawbacks and will also reduce costs. From previous experience, we have found that using the Unity game engine can provide a high-quality experience while having requirements which are not too high, therefore we advise you consider the Unity game engine.

Overall requirements:

- Yes, we would like the requirements to be as low as possible after considering the extra requirements from above.

We understand that setting higher software and hardware requirements can help improve the quality of the game, however, with the requirements that there will already be due to the Cardboard VR modules and the use of a game engine, we believe that these requirements will already be high enough to allow a good quality game.

Thank you again.

Yours sincerely,

[REDACTED]
[REDACTED]
[REDACTED]

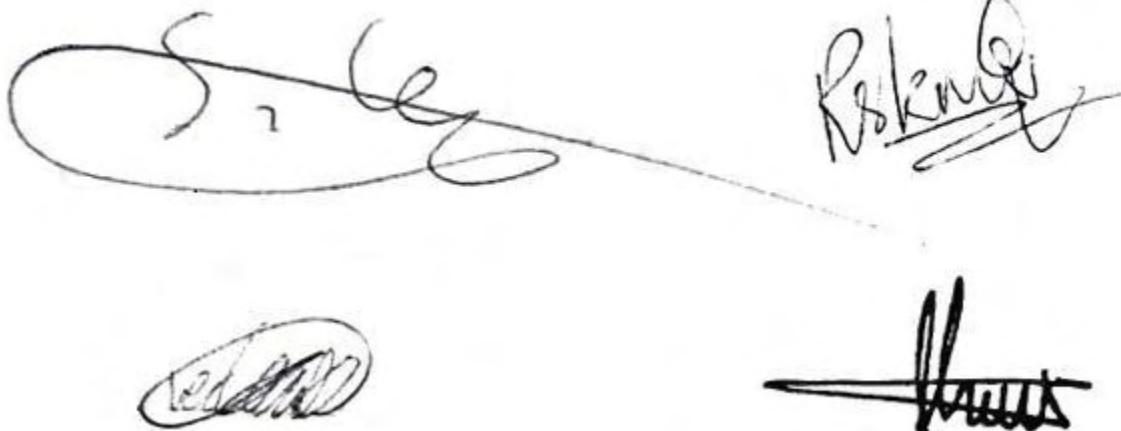
Harsh Brahmbhatt
Watford Boys Grammar School

5124
17655

I was then able to produce these hardware and software requirements:

Software / Hardware	Justification
3D Cardboard VR viewer (Hardware)	This viewer is required to be able to play the game in VR.
GVR SDK for Unity (Software)	This SDK (Software Development Kit) package is essential for some required modules of Google Cardboard VR games, e.g. for the reticle.
Android devices	
Software	
Android 4.4 'KitKat' (API level 19) or higher	The minimum AndroidOS version requirement for any Google Cardboard VR game as well as games developed on the Unity game engine.
OpenGL ES2.0+ OpenGL ES3.0+ Vulkan Graphics API	Minimum graphics API for games developed on the Unity game engine.
Hardware	
ARMv7 w/ Neon Support (32-bit) or ARM64 CPU	Minimum processor for games developed on the Unity game engine.
1+ GB of RAM	Minimum main memory for games developed on the Unity game engine.
iOS devices	
Software	
iOS 11.0 or later	The minimum Apple iOS version requirement for any Google Cardboard VR game as well as games developed on the Unity game engine.
Apple Metal Graphics API	The 3D graphic and compute shader API created by Apple, is the API required to run Unity games as well as run any games on IOS
Hardware	
Apple A7 SoC+ CPU	The minimum processor for games developed on the Unity game engine.

The requirements are all measurable (excluding software and hardware) as they have been uniformly created by me and approved by my client stakeholders with their signatures below. This means that the level of success of the product can be assessed at any time and at each stage to ensure that it is up to the standards of the clients.



Harsh Brahmbhatt
Watford Boys Grammar School

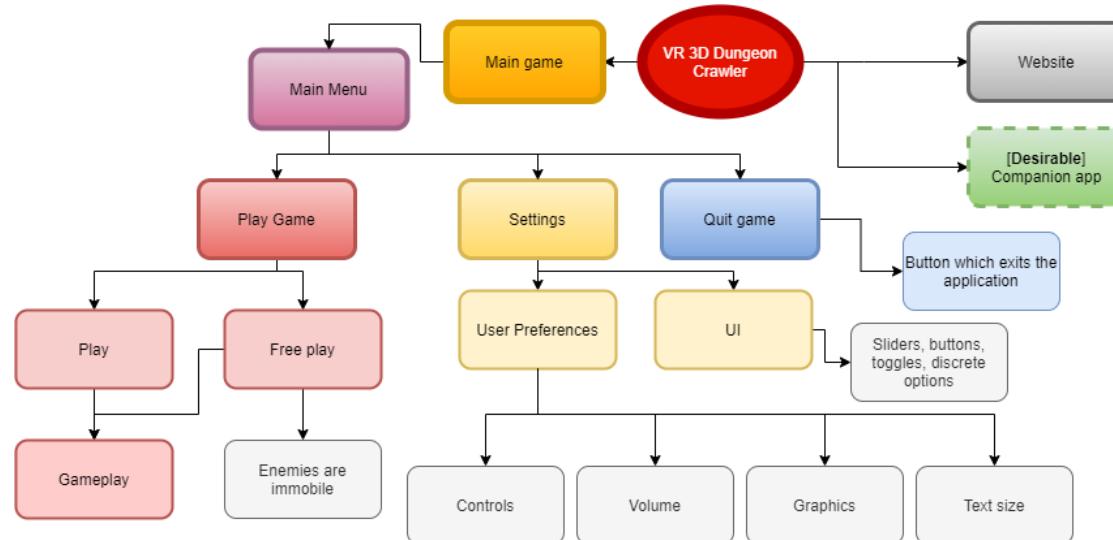
5124
17655

DESIGN

Problem Decomposition

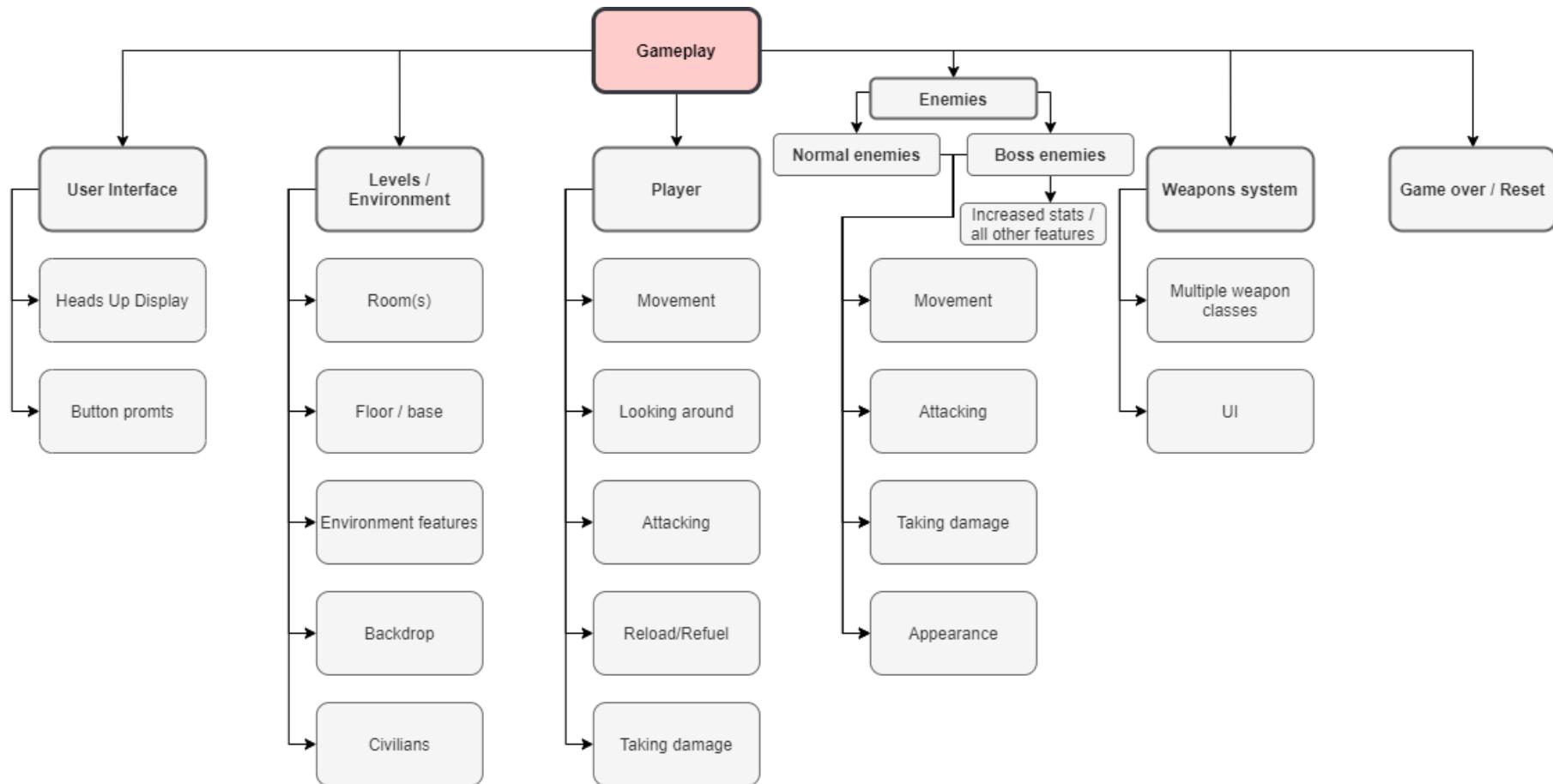
I aim to develop the solution using the Unity game engine which uses an object-oriented integration of C#. Therefore, it is useful to break down the problem, the entire game, into a series of objects and then into their methods and attributes using a top-down modular diagram. This allows me to consider the entire problem as a number of objects, i.e. sub-problems, which can each be solved individually, but then come together to form a complete solution. This makes the overall problem more solvable and the overall task more manageable. It also means that each sub-module can be tested individually meaning that testing can be much more structured and helps ensure it is done thoroughly. Being a fairly large scale problem, decomposing the problem down into sub-problems allows me to best practise concurrency, i.e. planning using pipelining to maximise the use of time. In other words, breaking down the problem allows me to complete multiple problems alongside one another if they don't depend on each other to function, and this means that the problem can be solved in a shorter amount of time.

I first dealt with the problem/project as a whole and the main components it contains:



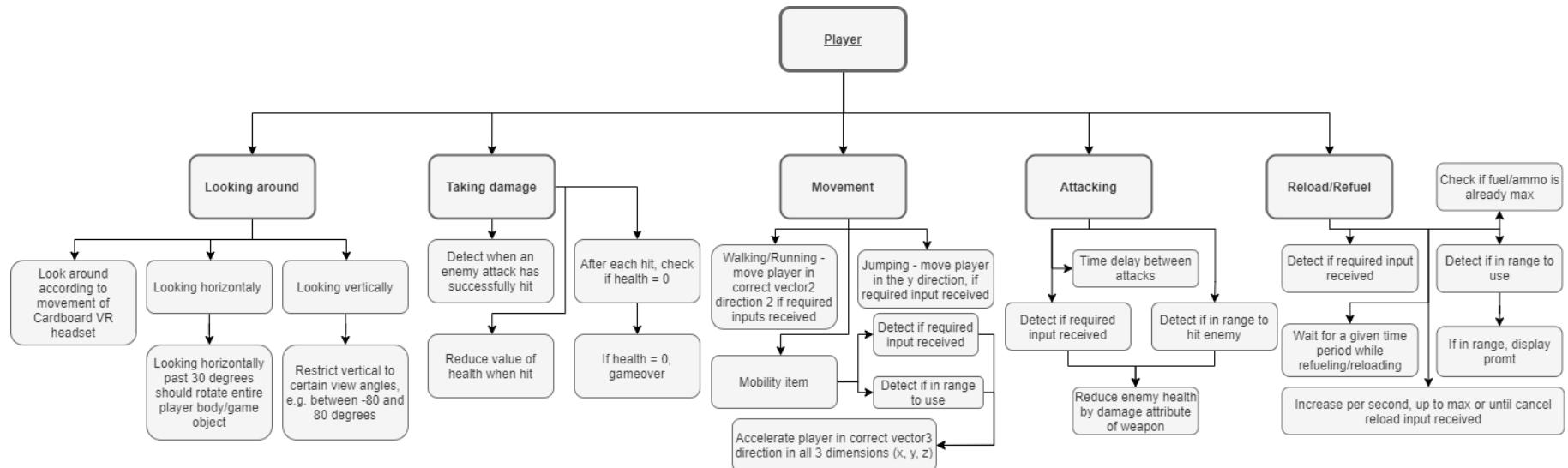
Of course, the main component of the project is the actual playable game itself which is to have 3 main options accessed through the menu; the main one being **playing the game**, the other being a **settings menu** to allow the player to alter a few options to their preference, and lastly an option to **exit** the game (these options fully decomposed here as they do not contain many sub-modules). However, as well as the game, an advertising **website** also is to be developed, along with a **companion app** if there is enough time.

Next, I further decomposed the game, this time the main gameplay side of things:

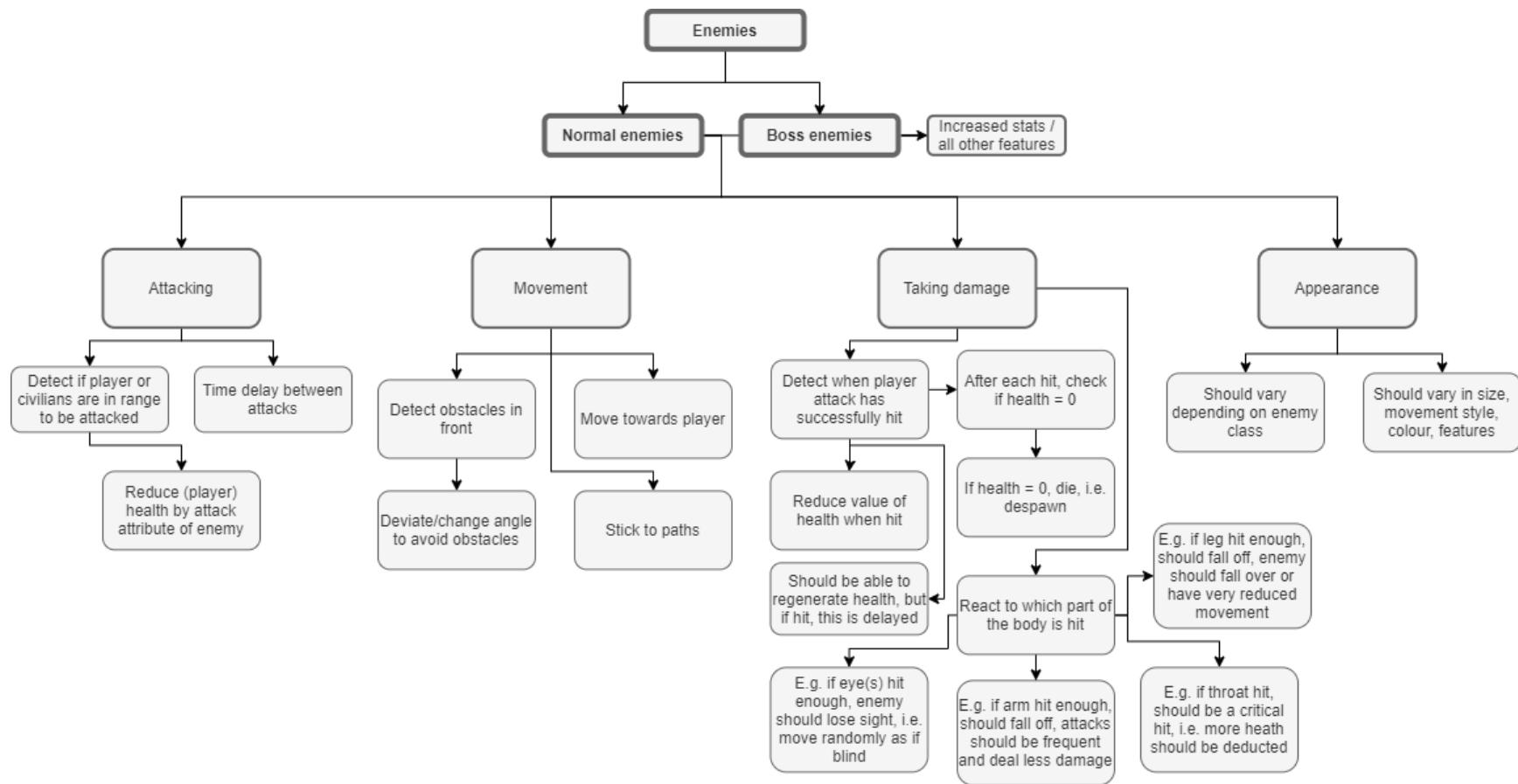


The gameplay consists of the **player** who aims to eliminate all **enemies** on a **map/level**. The player does this through the use of various **weapon classes** unlocked and selected by the player before playing a level. If the player dies and has 0 lives left, then it is **game over** and the game must either quit or reset. To allow the user to complete the main objective **UI** is essential to ensure the player always has enough information. In this way, I've split up the gameplay into 6 main objects/object types or considerations.

I next took each of these and broke them down into smaller modules, considering the methods and attributes of each object:

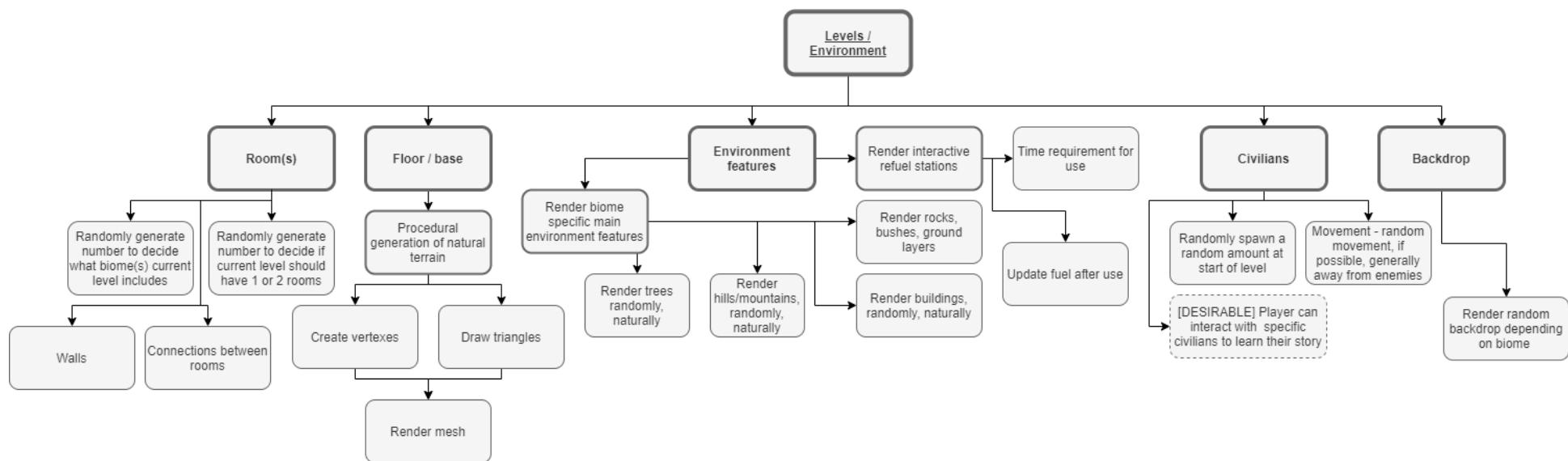


The player game object has various requirements in order for the game to function correctly, it has been broken down into the essential components. As the player object is almost fully controlled by the player playing the game, various input detecting is required for many of these features. **Movement** is required, i.e. the player needs to be able to control moving around naturally as they are simulating a human, this means that they need to be able to run around as well as jump. Furthermore, the player needs to have some sort of mobility item to allow them to move in any direction quickly. As well as movement, the player also needs to be able to see and **look around** in all directions, however, there should be a limit on the angles at which the player can view to ensure looking around is natural and realistic. In terms of combat with the enemies, the player needs to be able to **attack**, i.e. inflict damage as well as **take damage**. A couple of checks need to be in place to see whether an attack should successfully hit, i.e. if the player is in range to attack. There should also be a time delay, an attribute of the weapon, between attacks. These weapons also need to be **reloaded** and the mobility gear **refuelled** and so it is crucial that the player has a method of interacting with the refuel/reload stations. Interactions must ensure certain conditions are met, e.g. if the player is at max fuel, more fuel shouldn't be added.



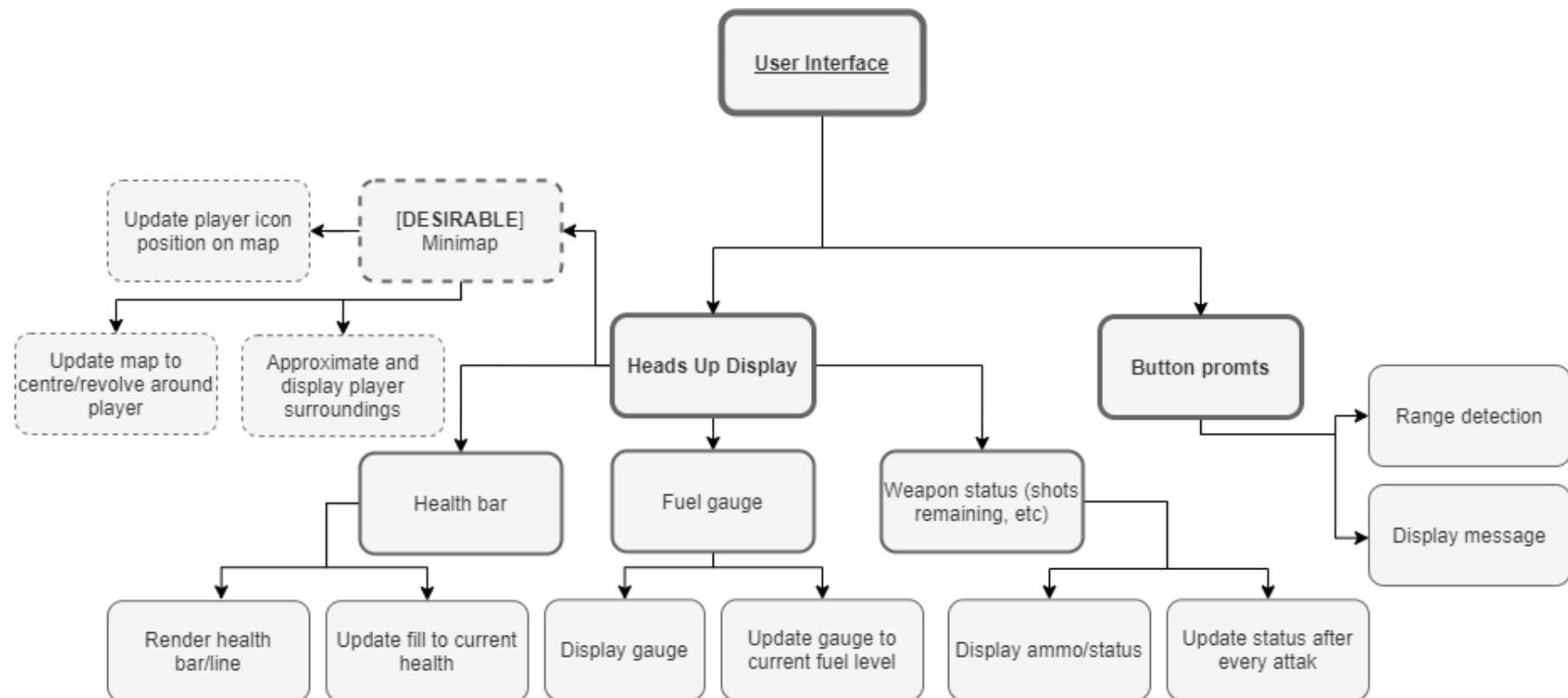
The Enemies come in 2 different types, the **normal enemies** and the **boss enemies** at the end of each level, this boss enemy can be a subclass of the normal enemy class since it shares the same attributes and methods but with a few extra functionalities, such as having more health and attacking differently as well as more frequently. The main enemy class consists of a few main functions, firstly, the enemies need to be able to automatically **move** and traverse the map. Therefore, the enemy essentially should be able to observe/analyse the surroundings to be able to move efficiently, this involves having a main path that the enemies can walk across, this would be best represented as a weighted graph with the edges being the distance between nodes, then the enemies can find the shortest path to be able to **move towards the player**. However, this graph will need to be generated after each level is procedurally generated. The enemy, being an enemy, i.e. the opposition to the player, the enemy should locate and move towards the player to

attack them to keep the gameplay intense and ensure the player has something to do and isn't bored. During combat, the enemy should be able to **attack the player** as well as **take damage**. If the player (or civilians) are in range of the enemy, the enemy should attempt to attack, the attack should take a given amount time and if the player is still in range by the end of this time, then the attack should land and the player should take damage, i.e. lose health points. As I want the player to enjoy the process of fighting the enemy, the ways in which the enemy takes damage should be dynamic. I.e. the enemy should react differently to how/where they are attacked, and this should alter their attribute values and also enable/disable certain methods such as movement. This makes the game more strategic and exciting for the player. Finally, the appearance, i.e. size, colours, etc. of the enemy should be an attribute of the enemy class and polymorphism should be used to have methods which alter the base appearance of an enemy so that enemies look different from one another. Furthermore, through inheritance, the boss enemies should extend from the main enemy class but the appearance should differ so that the player can easily tell the significance/importance of the boss enemy.

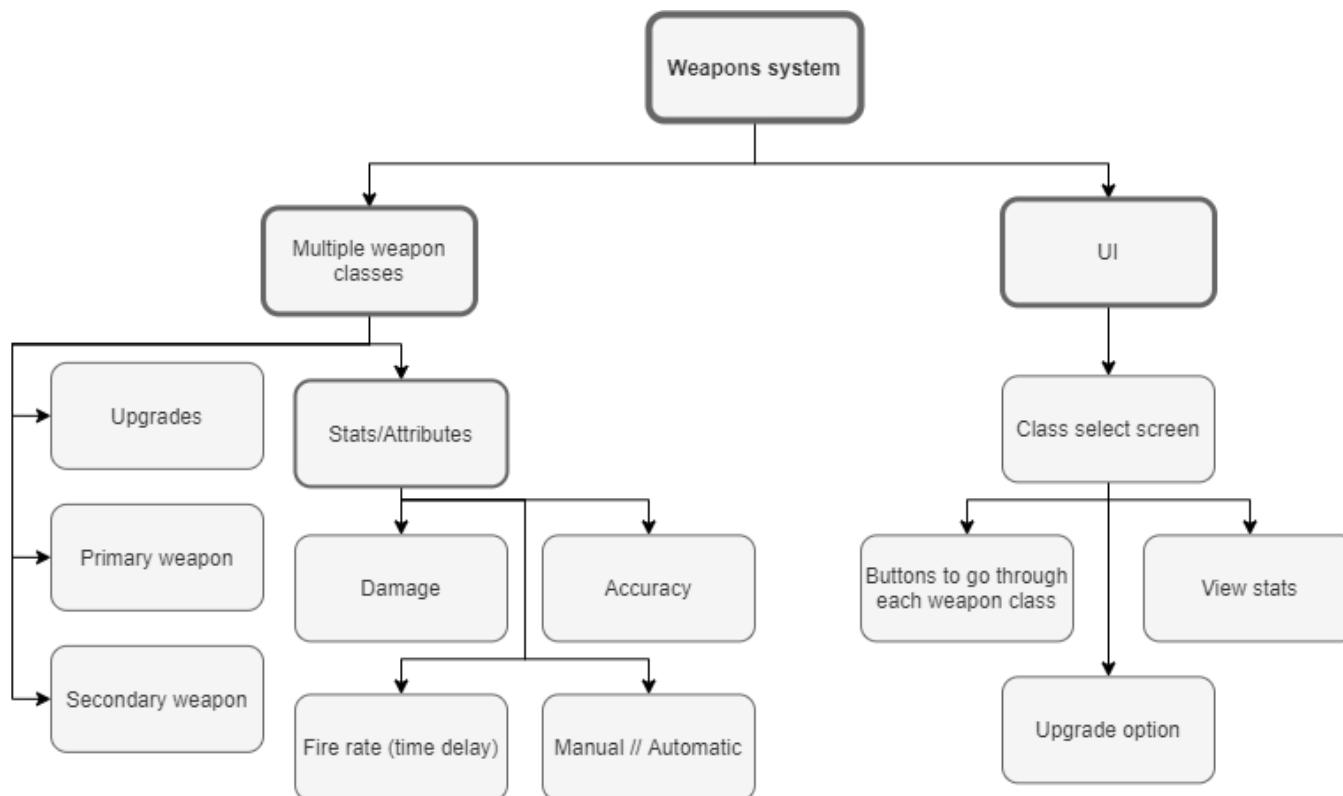


The map is an essential part of the game as it is where the player can explore and fight. It can be broken down and considered as a series of layers which build up to produce the complete map/level. Firstly, there's the **room(s)** itself consisting of surrounding walls and connections between the rooms to allow the player to actually travel between them. On top of that, the main **floor/base** of the level needs to be procedurally generated as without a floor there'd be no base for the player to traverse upon. As this is to be randomly/procedurally generated, this requires mesh/terrain generation. A mesh consists of vertices and triangles which connect to each of these vertices and this produces a mesh/terrain which the player can walk/run on and provides the base for all other

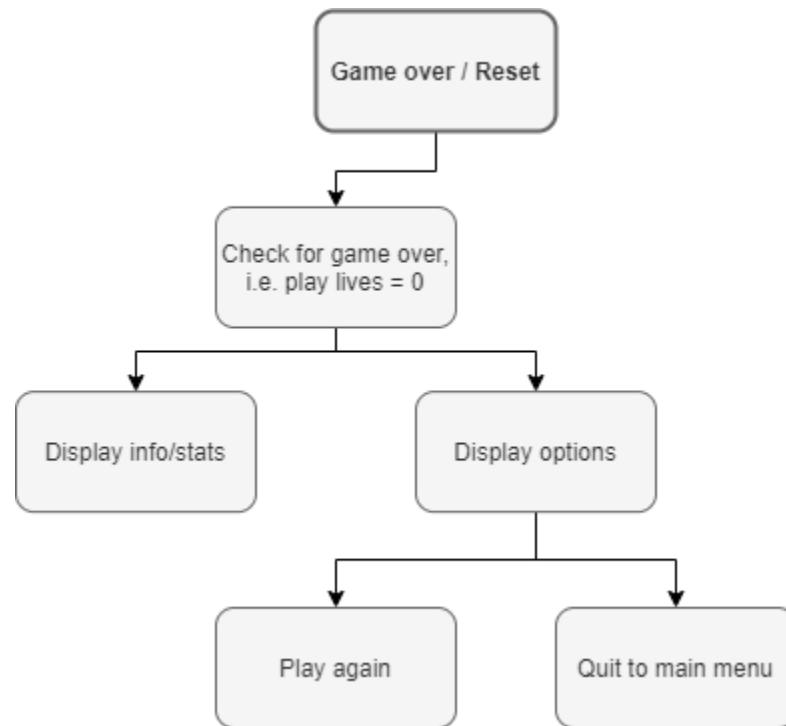
environment features. The positions of the vertices can be changed randomly to generate rough terrain, such as hills, which seems more realistic and interesting. **Environment features** include biome specific features such as buildings or trees, these need to be placed on the floor base randomly, i.e. differently every time, however these need to be placed such that they do not interfere with each other and also seem ordered/structured, leaving spaces for paths etc. Refuel stations are an essential, interactive part of the environment required to allow the player to refuel/reload. These need to be placed around the map so that the player has to be more strategic, e.g. not too far from each other but also not right next to each other, as this is useless. A suitable range should be defined so that any given level only has, e.g. 1-4 stations to create variety between levels. **Civilians** run around the map to give the player a sense of responsibility as they have to save the civilians. A random amount of civilians should spawn (within a given range) randomly and have some intelligence to move away from enemies if they are close. The **backdrop** will take up a lot of the screen and provide a sense of atmosphere, a random background should be chosen for each level so that levels feel different every time.



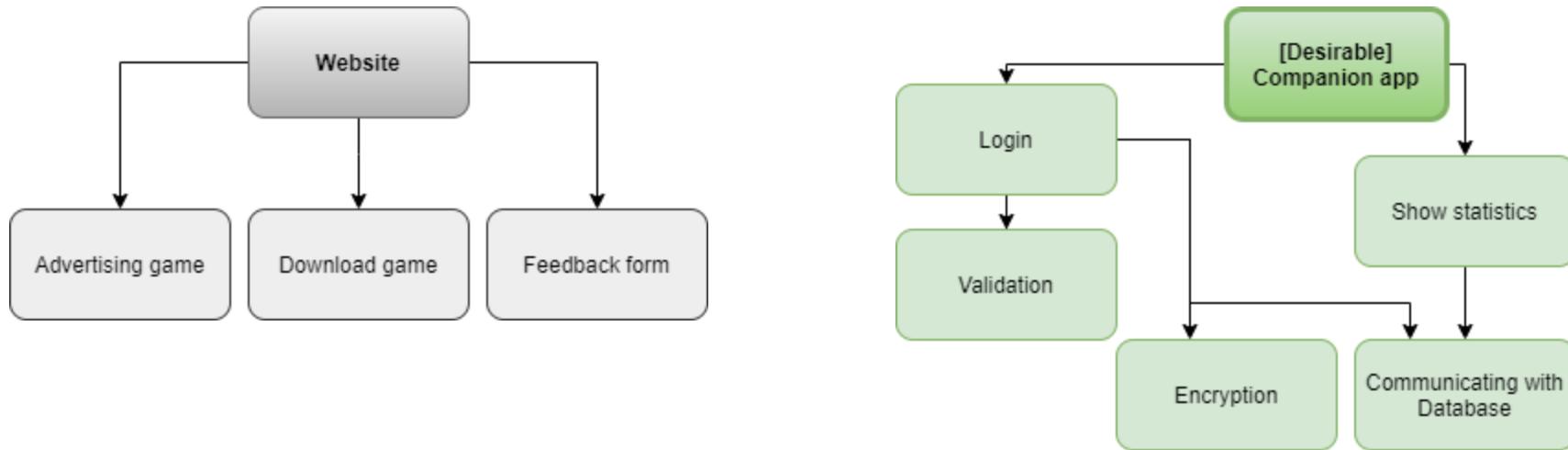
The User Interface is essential in communicating information to the player so that they have an understanding of what's going on, but should be balanced / kept to a minimum to increase the realismness and immersion of the VR experience. **Button prompts** are required to let the player know that they can press a certain button to interact with something, for example, refueling at a refuel station. These should appear as prompts if the player is in range to be able to use the interactive item. A **Heads Up Display (HUD)** is essential so that the player has enough information to play the game, this includes showing a health bar, fuel level or weapon status so that the player knows whether they can take on a fight or not, for example. These need to be frequently updated, i.e. the health should be updated every time the player takes damage, and the fuel whenever it is used, etc. A minimap is a desired feature of the game, not essential but should be implemented if there is enough time. This involves showing the player where they are relative to environment features such as buildings or trees. To render this map, a camera should be used to view the player and surroundings from a bird's eye view, then using some abstraction of fine details, the general picture of the player and surroundings should be scaled down and displayed at the corner of the player's screen.



The weapons system consists of the **UI** that the player interacts with to select a certain weapon class. The actual classes will consist of two weapons, a primary and a secondary which can be upgraded. The base weapon class will have a few attributes such as damage, accuracy, fire rate and manual or automatic option, and also methods such as fire, reload, etc. Different weapons will be instances of this weapons class with different attribute variables which will affect how the methods work, e.g. how the weapon fires. Through upgrades, the player can upgrade their weapon classes as they progress further into the game and get better and this will improve the discussed attributes.

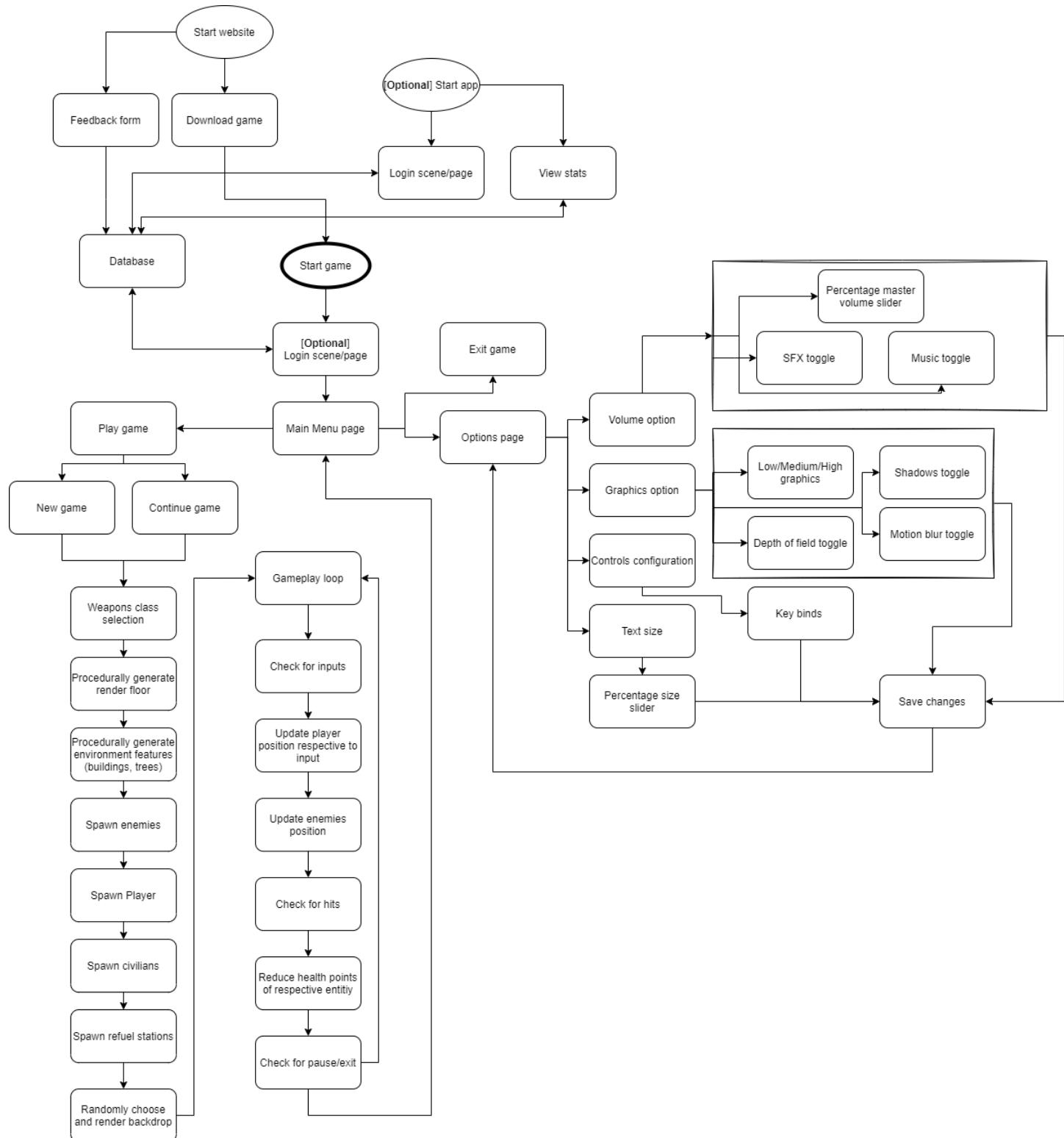


When the player runs out of lives, the game will need to be able to detect this and end, allowing the player to restart/replay or quit. This requires 2 buttons for the respective options. Depending on the choice/input, the game will need to restart with a fresh set of lives, or it will need to return to the main menu. On this screen, the stats of how the player has done in their playthrough should be displayed for the player's interest.



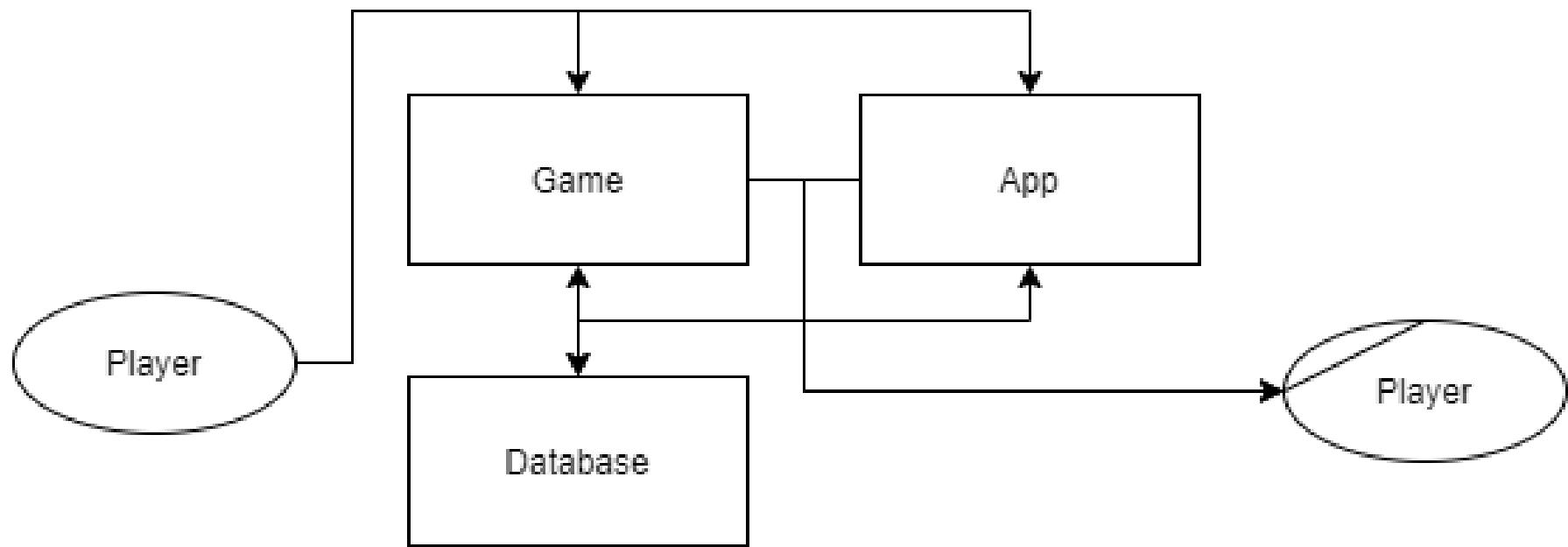
There are 2 external parts to the project not directly part of the game. This includes the promotion **website** which advertises the game and allows players to download it, as well as provide feedback and thoughts on the game. The second is the **companion app** where players can, using their account, view their stats and progress. This requires a login system, with validation to ensure security, to allow users to use accounts and the data must be stored in a database. The communication between the game, database and app must be encrypted to ensure all user's data is always kept secure.

Program structure

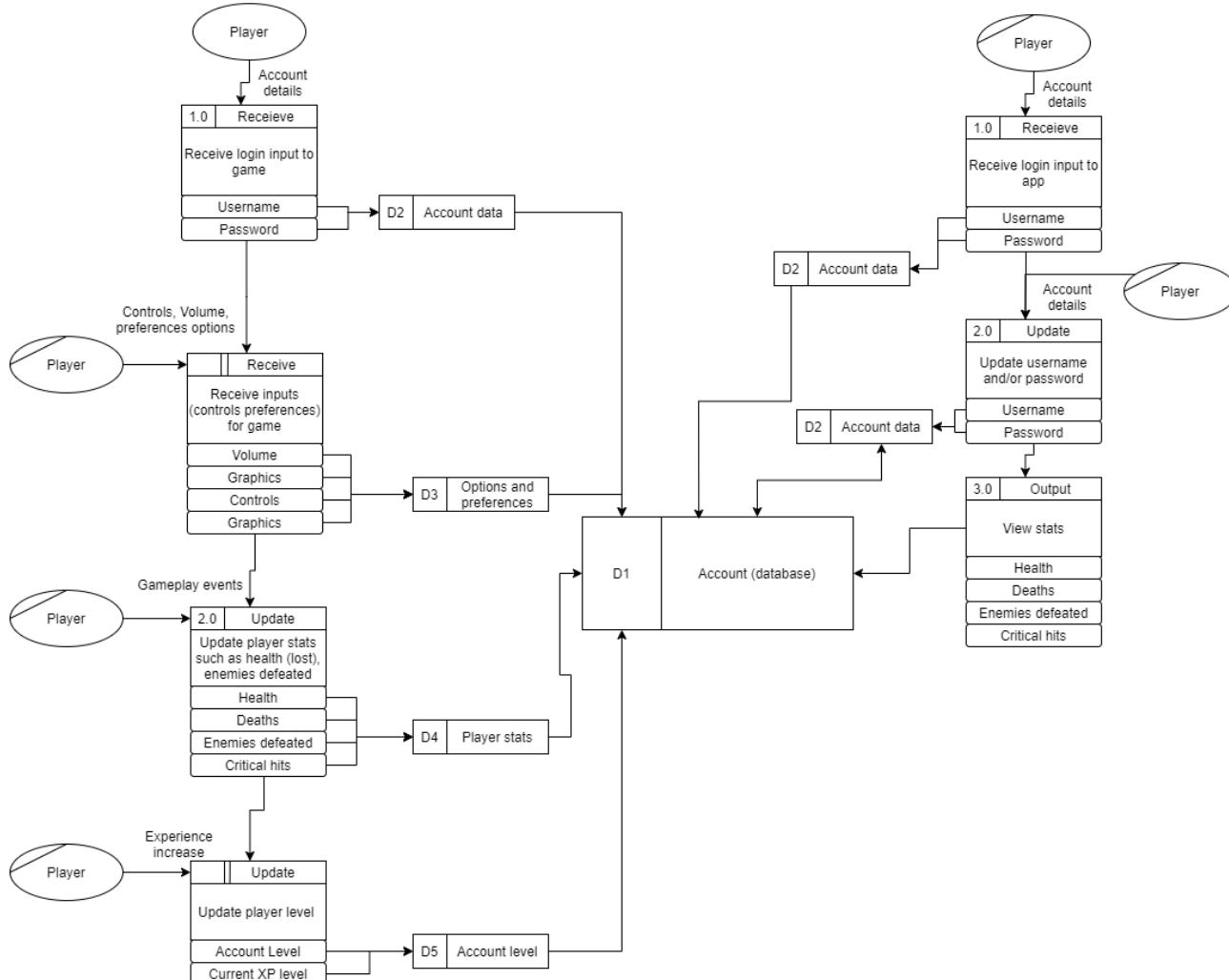


Data flow

Level 0 diagram



Level 1 diagram



Algorithms

Note: some of the algorithms here are designed in a module by module structure for the sake of design clarity and testing, however, the real program is to be implemented with a complete object-oriented approach.

Main menu

Pseudo code	Flowchart
<pre>function mainMenu() recievedInput = none if mouseDown recievedInput = mouseDown if recievedInput = play Play() else if recievedInput = settings Settings() else if recievedInput = quit Quit()</pre>	<pre>graph TD A[mainMenu] --> B{recievedInput = mouseDown} B --> C{Does receivedInput = play?} C -- Yes --> D[Play] C -- No --> E{Does receivedInput = settings?} E -- Yes --> F[Settings] E -- No --> G{Does receivedInput = quit?} G -- Yes --> H[Quit]</pre>

Justification			
The menu algorithm is the first user-interactive page the player sees and is used as a portal to navigate to the game's features. If the user selects an option on screen, this input will be detected and identified as to which option it is. The 'Play' option is the main option, used to start playing the game. The 'Settings' option takes the player to the settings page where they can configure certain options/preferences to their liking to improve their gameplay experience. The 'Quit' option is simply to exit the application when the user wants to stop playing.			
Link to other modules			
This module will be called at the start of the program once everything is sufficiently loaded Links further to 3 other modules:			
<ul style="list-style-type: none"> - <u>Play option</u> - if selected, this module will take the player into the game - <u>Settings option</u> - if selected, this module will take the user to a settings page where user can change settings - <u>Quit module</u> - if selected, this module will close the application 			
Data structures & key variables			
<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<i>recievedInput</i>	Must be the value of a button/UI input selected by the player.	Used to represent the option the player chooses so that the corresponding module can be called.	UserInput variable
Iterative development module testing			
<u>Input</u>	<u>Expected output</u>	<u>Justification</u>	
'Play' button click (acceptance data)	'Play' subroutine is called and runs	Tests to see whether the 'Play' button works and takes the player to the correct place	
'Settings' button click (acceptance data)	'Settings' subroutine is called and runs	Tests to see whether the 'Settings' button works and takes the player to the correct place - the settings page	
'Quit' button click (acceptance data)	'Quit' subroutine is called and runs	Tests to see whether the 'Quit' button works, i.e. does it successfully exit the application or not	

Quit

Pseudo code	Flowchart	
<pre>function quit() save() application.exit()</pre>	<pre> graph TD quit[quit] --> SaveData[Save data] SaveData --> END((END)) </pre>	
Justification		
Simple algorithm that ends the game/application. But before this it does carry out a final progress data save to the database in case this didn't occur earlier, when it should have, this is to ensure no data is lost. Note this is only to be implemented if the user accounts // database feature exists within the development period.		
Link to other modules		
This module links to the mainMenu() module. This module is called when the user presses the 'Quit' button on the main menu.		
Data structures & key variables		
-		
Iterative development module testing		
Input	Expected output	Justification
'Quit' button click (acceptance data)	Program ends/closes	Test to simply see if the application ends as it is supposed to when this module is called, i.e. when the player clicks 'Quit'.

Settings

Pseudo code	Flowchart
<pre> function settings() recievedInput = none i = 1 graphicsLevels = [0, 1, 2] currentGraphicsLevel = graphicsLevels[i] mainButton = mouseDown secondButton = holdMouseDown currentVolume = 50 if mouseDown recieivedInput = mouseDown if recieivedInput = toggleGraphics i = i + 1 if i = 3 i = 0 // i must be between 0 and 3 // as the array has only 3 value - 3 graphics options currentGraphicsLevel = graphicsLevels[i] if recieivedInput = bindControls buttonToBind = input ("Choose a button to bind") if buttonToBind = setMainButton mainButtonChoice = input("Enter main input") mainButton = mainButtonChoice // ie sets primary fire if buttonToBind = setSecondButton secondButtonChoice = input("Enter secondary input") secondButton = secondButtonChoice // ie sets secondary fire if recieivedInput = adjustVolume sliderLevel = recieivedInput currentVolume = sliderLevel/100 // sets the current volume as a fraction of 100 (full) if recieivedInput = back menu() </pre>	<pre> graph TD Start[settings] --> Input1{receivedInput = mouseDown} Input1 --> Decision1{Does receivedInput = toggle graphics?} Decision1 -- Yes --> Increment[Increment graphics up (if highest graphics, toggle to lowest)] Increment --> Decision2{Does receivedInput = adjust volume?} Decision2 -- Yes --> Volume[current volume = point on volume slider clicked by player] Volume --> Decision3{Does receivedInput = bind controls?} Decision3 -- Yes --> Bind1[buttonToBind = mouseDown] Bind1 --> Main1{Does buttonToBind = set main button?} Main1 -- Yes --> MainInput1[main Button = user input] Main1 -- No --> Second1{Does buttonToBind = set second button?} Second1 -- Yes --> SecondInput1[second Button = user input] Second1 -- No --> Decision1 MainInput1 --> Decision4{Does receivedInput = back to menu?} SecondInput1 --> Decision4 Decision4 -- Yes --> Menu[menu] Decision4 -- No --> Decision1 </pre>

Justification

The settings page is used to allow the player to change certain settings to their preference to improve their personal experience of playing the game. The player can change graphics level, control binds and volume level. There is also an option to go back to the main menu so that the player can access other features of the game as they wish.

The toggle graphics option is a button which, when clicked on by the player, switches between graphics modes. Every time it is clicked, the graphics level is incremented by 1 through the 3 graphics levels, if it is clicked when graphics mode is currently highest, it will reset back to lowest graphics mode.

The bind controls button presents the player with 2 more options which asks the user to specify which input type they would like to set, either the primary fire, or secondary. The user then presses their preferred input for the chosen input type.

The volume control is a slider allowing the user to choose what fraction of the max volume they'd like to set. The slider bar's length value out of the total length of the bar is used to determine this.

Finally, the settings page contains a back button which returns the player to the main menu page by calling the `mainMenu` module. This provides a way back to the main menu so that the player can then access other features of the game as they wish.

Link to other modules

This module links to the `mainMenu` module, as it is called when the player selects the settings option from the menu.

This module also calls the `mainMenu` module if the back button is selected by the player.

Data structures & key variables

<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<code>recievedInput</code>	Must be the value of a button/UI input selected by the player.	Used to represent the option the player chooses so that the corresponding module can be called.	UserInput variable
<code>current graphicsLevel</code>	Must be equal to either 0, 1 or 2, a value from the <code>graphicsLevel</code> list.	This is set to one of the 3 elements of the graphics level arrays to represent the chosen graphics level and can be changed by the user.	Integer
<code>graphicsLevel</code>	A defined constant list which must contain only 3 elements each representing one graphics level	This is an array of 3 elements which each represent the 3 graphics modes, 0 being the lowest level and 2 being the highest and is used to set the <code>currentGraphicsLevel</code> .	List (of integers)
<code>mainButton</code>	Must be the value/input selected by the player.	This is the variable which holds the input required to activate the primary fire	UserInput variable
<code>secondButton</code>	Must be the value/input	Holds the input required to activate	UserInput

	selected by the player.	the secondary fire	variable
<i>currentVolume</i>	Must be a number between 0 and 1	Value between 0 and 1 which represents the volume level out of 1 which translates to a percentage out of 100.	Float
Iterative development module testing			
<u>Input</u>	<u>Expected output</u>	<u>Justification</u>	
Toggle graphics button press (3 times) (Acceptance data)	Graphics levels increase, if max graphics reset to min	Checks whether graphics levels successfully toggle/increment between levels and if it is already set to the max option, that it resets to the lowest option, instead of incrementing further which isn't possible so would throw an index out of range error.	
Bind main button to mouseDown (Acceptance data)	Main button is set to mouseDown	Checks whether the primary fire input can be changed to whatever the user chooses.	
Bind secondary button to holdMouseDown (Acceptance data)	Secondary button is set to holdMouseDown	Checks whether the secondary fire input can be changed to whatever the user chooses.	
Volume slider to 20% (Normal)	Volume is set to 0.2	Checks that the volume slider can be set to a normal value between 0% and 100%.	
Volume slider to 100% (Boundary)	Volume is set to 1.0	Checks to see if the volume can be set to an edge limit such as 0% or 100%.	
Back button press/click (Acceptance data)	mainMenu module runs	Tests that the back button takes the user back to the main menu, i.e. the required module runs.	

Player

Pseudo code	Flowchart
<pre>function player() position = [100, 0, 100] health = 100 lives = 5 speed = 15 weapon = swordWeapon fuel = 100 enemiesKilled = 0 score = 0 // note, when implemented, this will be an object oriented base class which will be extended by many others</pre>	<pre> graph TD A[player()] --> B["position = [100, 0, 100]\nhealth = 100\nlives = 5\nspeed = 15\nweapon = swordWeapon\nfuel = 100\nenemiesKilled = 0\nscore = 0"] </pre>

Justification

This is a base player class which is then extended by other classes (modules designed here instead) to manipulate these base attributes to essentially allow/create/execute gameplay. E.g. The move class will alter the position of the player. The values currently assigned are preset values.

Link to other modules

Links to all of the player modules (classes), including:

The move class affects the player's position attribute by the player's speed attribute.

The jump class which alters the player's position attribute through reference of this module/class.

The attack class which gets the weapon the player is using, which is its own object/class, to damage enemies and also reduces the fuel attribute of the player when the grapple hook is used.

The refuel class which increases the fuel attribute of the player when they stand on a refuel station.

The damage class which reduces the player's health attribute when they are hit by an enemy.

Data structures & key variables

Variable / Data structure	Validation	Description & Justification	Type
<i>position</i>	Must be a point in 3D space representing the position.	Represents position of the player, ie. where the player is in the scene.	List (of coordinates) (floats)
<i>health</i>	Must be between 0 and 100.	Used to denote dead or alive stat (alive if $health > 0$), player loses health when hit by an enemy.	Integer

<i>lives</i>	Must be greater than 0 for the game to keep playing.	Used to check for game over, if lives=0, then game ends and can be restarted.	Integer
<i>speed</i>	Must be a constant to feel intuitive, as the player moves around on foot.	This is the rate at which the player moves around, i.e. the rate of change of distance.	Float
<i>weapon</i>	Must be a weapon object from the respective class.	This is the weapon chosen by the player	Weapon object/class
<i>fuel</i>	Must be between 0 and 100.	A value which is required to use mobility item, is reduced every time mobility item is used.	Integer
<i>enemiesKilled</i>	Must be positive or 0.	A value to track how enemies the player has defeated.	Integer
<i>score</i>	Must be positive or 0.	A value to track how much score the player has earned.	Integer
Iterative development module testing			
-			

Player Movement

Pseudo code	Flowchart

```

function lookMove(player)
    minAngle = 30.0
    maxAngle = 330.0
    halfSpeed = 7.5
    move = none
    angle = camera.x

    while True
        if angle < minAngle OR angle > maxAngle
            move = true
        else
            move = false

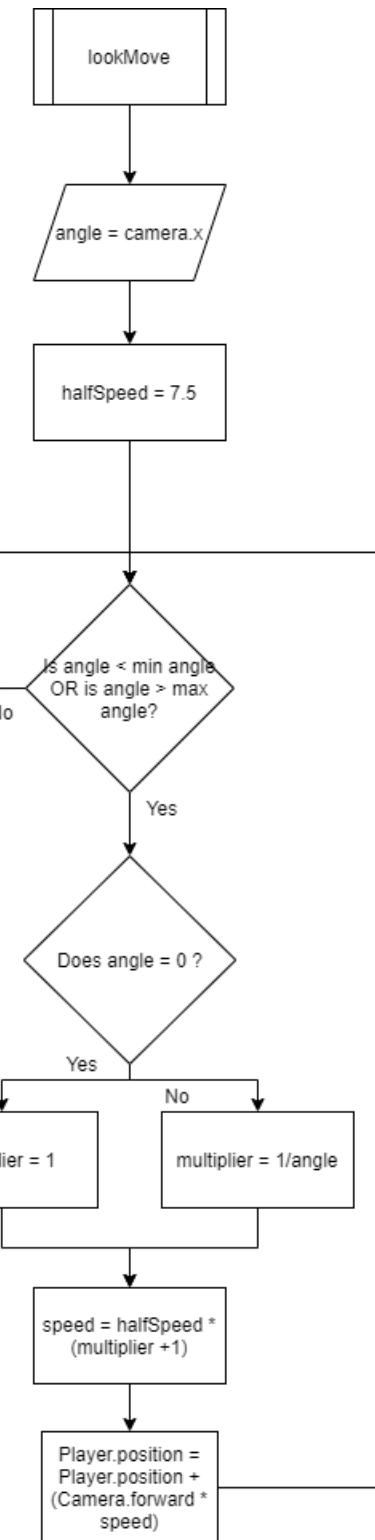
        if move
            if angle = 0
                multiplier = 1
            else
                multiplier = 1/angle

            speed = halfSpeed * (multiplier+1)
            Player.position = Player.position + Camera.forward * speed

```

// note: has parameter player however, using an object oriented paradigm, lookMove will extend the player class

// note: the player looking horizontally means the camera.x is 0 degrees, above this the angle increments, but below this the angle decreases from 360 degrees



Justification

This algorithm continuously moves the player forwards in the direction they are looking. This automatic movement is used as the Cardboard VR has limited inputs which would be put to much better use for other functionalities. However, for this auto movement to occur the player must be looking in a given angle range of 30 degrees above and below the horizontal. If the player is looking outside this range then the player gameobject will not move. This gives the player some level of control over their own movement even though it is automatic so that they can stop by looking fairly high into the sky, to look at scenery perhaps, or staring at the ground. Furthermore, to add more depth to the control the player has over their movement, the speed varies depending on the angle the player looks at. If they look straight ahead, then they will move at maximum speed, but as they look further up/down their speed will decrease by a multiplier which is worked out by considering the angle the player looks at and the speed multiplier as a reciprocal relationship.

Link to other modules

Will extend the base player class but no direct links - this is a module that runs continuously during gameplay so is called when the player enters gameplay.

Data structures & key variables

<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<i>angle</i>	Must be between 0 and 360 degrees.	This is a variable read from the camera's x value, i.e. how far up or down the player is looking relative to the horizontal.	Float
<i>minAngle</i>	A constant set to 30 degrees but must be between 0 and 360 degrees and must be less than the <i>maxAngle</i> .	Set to 30.0 degrees, this variable represents the angle at which if the player looks out of (i.e. < 30) then they will not move.	Float
<i>maxAngle</i>	A constant set to 330 degrees but must be between 0 and 360 degrees and must be greater than the <i>minAngle</i> .	Set to 30.0 degrees, this variable represents the angle at which if the player looks out of (i.e. < 30) then they will not move.	Float
<i>multiplier</i>	Must be a value below 0, then 1 is added to it. Which can cause a maximum multiplier of 2 as required	This value multiplied to the half speed to increase this base speed depending on the looking angle.	Float
<i>halfSpeed</i>	A constant set to 7.5, must be a float representing half the player's maximum speed.	Represents half the max speed player moves at, since the max multiplier that can be achieved is 2.	Float

Iterative development module testing		
<u>Input</u>	<u>Expected output</u>	<u>Justification</u>
Looking horizontally (0 degrees) (Normal)	Player moves in the direction they are looking with maximum speed	Checks whether the player moves at maximum speed when they are looking straight ahead, therefore a multiplier of 2 applied to the halfSpeed, giving a speed of 15
Looking at 15 degrees, above horizontal (Normal)	Player moves with speed 8, in the direction they are looking	Checks whether the player moves as they are within the angle boundaries to move and whether the multiplier increments the speed by the correct amount (1.0666) to provide a speed of 8.
Looking at 50 degrees, above horizontal (Normal)	Player does not move	Checks whether the player stops moving, when looking outside of the angle boundaries.
Looking at 330 degrees, i.e. 30 degrees below horizontal (Boundary)	Player does not move	Checks whether the player stops moving, as they are looking at an angle which is on the angle boundary.

Player Jump

Pseudo code	Flowchart
<pre> function jump(player) jump = false while true if !jump startAngle = camera.x waitForTime(0.5) endAngle = camera.x angleChange = startAngle - endAngle if endAngle <= 0 AND angleChange >= 20 jump = true if jump Player.velocity += Player.up * 10 jump = false waitForTime(0.5) // note: has parameter player however, using an object oriented paradigm, jump will extend the player class </pre>	<pre> graph TD jump[jump] --> startAngle[/startAngle = camera.x/] startAngle --> wait05[wait for 0.5 seconds] wait05 --> endAngle[/endAngle = camera.x/] endAngle --> angleChange[angleChange = startAngle - endAngle] angleChange --> decision{Is endAngle <= 0 AND is angleChange >= 20 ?} decision -- No --> End decision -- Yes --> velocity[Player.velocity = Player.velocity + Player.up * 10] velocity --> wait05 </pre>

Justification			
<p>This algorithm is to allow the player to jump, however, it is detected through the angle at which the player looks using the CardboardVR viewer like the automatic movement did. It does this by taking a reading of the current angle that the player is looking at before and after a given time period of 0.5 seconds. Then, by calculating and evaluating the difference in these 2 angles, a decision is made whether the player is to jump, i.e. if the difference is large enough (≥ 20 degrees) meaning the player has very quickly looked upwards, the player gameobject should jump. Furthermore, the algorithm checks if the end angle is above the horizontal for the player to jump. This simply stops the player jumping if they go from looking directly downwards to looking less downwards but still below the horizontal as jumping in this situation probably wasn't the player's intention. To actually jump, a sudden, upwards velocity is added to the player's velocity, accelerating them upwards before the gravity takes effect pulling the player back down to the floor. The jump flag is then set to false so that the player has to repeat the process of looking up very quickly above the horizontal to jump again. The program then pauses for a given time period of 0.5 seconds to ensure the player has fallen back to the floor again before checking for another jump, and this is simply to limit the player to only being able to jump if they are on the floor, i.e. preventing mid-air jumps.</p>			
Link to other modules			
Will extend the base player class but no direct links - this is a module that runs continuously during gameplay so is called when the player enters gameplay, checking for and executing jumps.			
Data structures & key variables			
<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<i>jump</i>	Must be either true or false as the player can either jump or not jump at any one time.	Used to denote/toggle whether the player can jump or not, i.e. if the required condition is met; the angle change being large enough.	Boolean
<i>startAngle</i>	Must be between 0 and 360 degrees.	Represents the first camera angle reading taken, i.e. before the time wait, so that it can be compared to the reading taken after.	Float
<i>endAngle</i>	Must be between 0 and 360 degrees.	Represents the second camera angle reading taken, i.e. after the time wait, so that it can be compared to the reading taken before.	Float
<i>angleChange</i>	Must be the positive difference between startAngle & endAngle. Therefore must be > 0 , and by definition < 360 .	Represents the difference between the startAngle and endAngle, i.e. the change in angle over the time wait. If this is large enough (> 20), it is considered a jump, so jump is set to	Float

		true.	
Iterative development module testing			
<u>Input</u>	<u>Expected output</u>	<u>Justification</u>	
Quickly change looking angle by a few degrees (5-10 degrees). (Normal)	Player does not jump (no upwards movement)	A small difference in angle is probably just the player looking around and is not drastic enough to be considered as a jump so player shouldn't jump.	
Quickly change looking angle by 35 degrees. (Normal)	Player jumps (moves upwards)	This difference in angle is large enough (≥ 20 degrees) to be considered an attempt at jumping by the player so the player should jump.	
Quickly change looking angle by 20 degrees. (Boundary)	Player jumps (moves upwards)	This difference in angle is on the boundary to be considered a jump, however, due to the " \geq " condition, this should be considered a jump.	
Quickly change looking angle by over 25 degrees, during a jump, i.e. while the player is mid-air. (Erroneous)	Player does not jump (no upwards movement)	The player should not be able to jump during a jump as this would allow the player to essentially fly which is not a feature of the game. So, here it should be ensured that the player doesn't jump.	

Player Attack

Pseudo code	Flowchart
<pre> function playerAttack(weaponRange, weaponDamage, weaponDelay, player) hit = false grapple = false while true if mouseDown hit = true if holdMouseDown grapple = true else if MouseUp grapple = false if hit playAnimation(swordHitAnimation) hitRay = castRay(player.position, player.forwards) attackPoint = hitRay.hitPoint hitDistance = attackPoint - player.position if hitDistance < weaponRange enemyHit = attackPoint.gameObject enemyHit.damage(enemyHit, weaponDamage) waitForTime(weaponDelay) if grapple grappleRay = castRay(player.position, player.forwards) grapplePoint = grappleRay.hitPoint grappleDistance = grapplePoint - player.position if grappleDistance <= 35.0 AND grappleDistance > 8.0 grapple = true else grapple = false if grapple drawLine(Player.position, grapplePoint) player.position += grapple.direction * 10 player.fuel -= 0.2 else grapple = false destroy(Line) // note, here parameters are passed however, // this will most likely be done by inheriting a // superclass </pre>	<pre> graph TD Start[playerAttack(weaponRange, weaponDamage, weaponDelay, player)] --> IsMouseDown{Is mouseDown?} IsMouseDown -- Yes --> PlayHit[Play swordHit animation] PlayHit --> CastRay1[Cast ray from player, forwards] CastRay1 --> HitDistance[hitDistance = distance between player and first thing the ray hits] HitDistance --> IsHitRange{Is hitDistance within the weapon's range?} IsHitRange -- No --> End[] IsHitRange -- Yes --> ReduceHealth[Reduce health of enemy hit by the weapon's damage stat] ReduceHealth --> WaitDelay[Wait for time equal to delay attribute of weapon] IsHitRange --> IsMouseHeld{Is mouse held down?} IsMouseHeld -- No --> End[] IsMouseHeld -- Yes --> CastRay2[Cast ray from player, forwards] CastRay2 --> GrappleDistance[grappleDistance = distance between player and first thing the ray hits] GrappleDistance --> IsGrappleRange{Is grappleDistance between 35 and 8?} IsGrappleRange -- Yes --> DrawGrapple[Draw grapple line] DrawGrapple --> Accelerate[Accelerate/move player in direction of the point the grapple ray hits] IsGrappleRange -- No --> DestroyLine[Destroy any existing grapple line] </pre>

Justification			
<p>This algorithm is used to allow the player to attack using their weapon and use the grappling hook. The player presses the button to use their weapon and holds down the button grapple towards the point they are looking at.</p> <p>When the player pressed the button the hit animation plays and a ray is shot out in the direction the player hits, i.e. the direction they are looking. The ray detects the point at which it hits/intersects with another enemy and if this point is in range of the weapon to hit, then the player does damage, equal to the damage stat of the weapon in use, to this enemy. (A mask can be applied to make sure the ray only detects enemies). The program then has a time wait/delay equal to the delay stat of the weapon in use so that the player has to wait a certain amount of time before they can attack again, this is to add a level of difficulty to the game.</p> <p>When the player holds down the button, a ray is cast out in the direction the player is looking and stores data of what and where the ray hits.</p>			
Link to other modules			
Will extend the base player class but no direct links - this is a module that runs continuously during gameplay so is called when the player enters gameplay, checking for and executing attacks or grapples.			
Data structures & key variables			
<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<i>hit</i>	Must only be true or false at any one time.	Used to check whether the player wants to hit, this is checked every frame, if hit input is received, hit is set to true and a hit can be executed.	Boolean
<i>hitRay</i>	Must only detect intersections with enemy game objects using a mask.	An object which is essentially a line which stores data such as length to a certain point, intersection points, etc. Used to essentially pre-fire a hit.	List (points that form a line) (floats)
<i>attackPoint</i>	Must be a point which is part of an enemy as only they should be detected by the (hitRay) ray.	This is a point in space which represents where the hitRay ray has first intersected with an enemy gameobject.	List (of co-ordinates) (floats)
<i>hitDistance</i>	Cannot be directly changed by the player, but can be indirectly changed by getting closer to an enemy. This is used to validate a hit, i.e. the value must be less than / within the weapon's	This is a value which represents the distance between the player and the enemy that they are trying to hit. It is then used to decide whether a hit should register on the enemy or not, depending on if the player is close enough, i.e. if the distance is less than / within the weapon's range.	Float

	range.		
<i>enemyHit</i>	Must be a gameobject which is an enemy. (ensured using masks and layers)	This is the enemy gameobject to be hit, given that it is in range of the player. Using the gameobject, the enemy can be hit/damaged.	Game Object (enemy class)
<i>grapple</i>	Must only be true or false at any one time.	Used to check if the player wants to grapple, i.e. if the required input is received but then the same variable is used to check if conditions for a grapple are met, to be more efficient.	Boolean
<i>grappleRay</i>	Must only detect intersections with environment objects or enemies.	An object which is essentially a line which stores data e.g. length, intersections, etc. Used to pre-fire the grapple to carry out checks	List (points that form a line) (floats)
<i>grapplePoint</i>	Must be a point which is part of an enemy or environment object as only they should be detected by the (grappleRay) ray.	This is a point in space which represents where the grappleRay ray has first intersected with the environment or enemy gameobject. This is then used to calculate distance from player to that object.	List (of co-ordinates) (floats)
<i>grappleDistance</i>	Cannot be directly changed by the player, but can be indirectly changed by getting closer to an enemy. This is used to validate a grapple, i.e. the value must be within a given range of 35 and 8 units to actually grapple.	This is a value which represents the distance between the player and the object that they are trying to grapple to. It is then used to decide whether or not, depending on if it is within a given range of 35 and 8 units. This is because the player should not be able to grapple extremely far or short distances as this is not realistic.	Float

Iterative development module testing

Input	Expected output	Justification
Press button, attack enemy a distance greater than the weapon range (Normal)	Hit animation plays, however, the enemy is not damaged.	At a distance greater than the weapons range, the player is too far to land a successful strike on an enemy so hit animation should play but should not be a successful, damaging strike.
Press button, attack enemy a distance less than	Hit animation plays and the enemy is successfully damaged.	At a distance less than the weapons range, the player is close enough to land a successful strike on an enemy so hit animation should play and

the weapon range (Normal)		enemy should take damage.
Press button, attack enemy a distance equal to the weapon range (Boundary)	Hit animation plays, however, the enemy is not damaged.	At a distance equal to the weapons range, this is on the boundary, but the player is still considered too far (due to <) to land a successful strike on an enemy so hit animation should play but should not be a successful, damaging strike.
Hold down button, use grapple hook towards any enemy or environment object, at a distance greater than 35. E.g. 40 (Normal)	Grapple hook/line does not render and the player does not accelerate towards the object they are aiming for.	At a distance greater than 35 units, the player is too far to be able to use the grappling hook so it should not render and the player should not move/accelerate/grapple towards the object aimed at.
Hold down button, use grapple hook towards any enemy or environment object, at a distance less than 35 E.g 20 (Normal)	Grapple hook/line renders from player to the point on the object they were aiming for. Player moves/accelerates towards this point (until they are too close and therefore out of boundary)	At a distance less than 35 units, the player is within the boundaries to be able to use the grappling hook so it should render and the player should move/accelerate/grapple towards the object aimed at until they are within 8 units, at which they grappling hook should be destroyed and player should stop accelerating towards this point.
Hold down button, use grapple hook towards any enemy or environment object, at a distance equal to 35 (Normal)	Grapple hook/line renders from player to the point on the object they were aiming for. Player moves/accelerates towards this point (until they are too close / out of boundary again)	At a distance of 35 units, the player is still considered within the boundaries to be able to use the grappling hook (due to the <=) so it should render and the player should move/accelerate towards the object aimed at until they are within 8 units, at which they grappling hook should be destroyed and player should stop accelerating towards this point.
Hold down button, use grapple hook towards an object which is not an enemy or environment object (Erroneous)	Grapple hook/line does not render and the player does not accelerate towards the object they are aiming for.	As this is not a grappleable object, the player should not be able to move/grapple towards it. These objects will be infrequent and have specific purposes so it is important the player cannot use them for unintended purposes which may change gameplay or objectives or even difficulty.

Player Damage

Pseudo code	Flowchart
<pre> function playerDamage(player, damage) dead = false gameOver = false player.health -= damage if player.health <= 0 player.health = 0 dead = true if dead player.lives -= 1 if player.lives = 0 gameOver = true else respawn() if gameOver gameOverScreen() // note: has parameter player however, using an object oriented paradigm, playerDamge will extend the player class </pre>	<pre> graph TD Start[playerDamage(damage)] --> Init{dead = false gameOver = false} Init --> HealthSubtraction[player.health = player.health - 1] HealthSubtraction --> HealthCheck{Is player.health <= 0?} HealthCheck -- Yes --> GameOverScreen[gameOverScreen()] HealthCheck -- No --> LivesSubtraction[lives = lives - 1] LivesSubtraction --> LivesCheck{Does lives = 0?} LivesCheck -- Yes --> GameOverScreen LivesCheck -- No --> Respawn[respawn()] </pre> <p>The flowchart illustrates the logic of the <code>playerDamage</code> function. It starts with initializing <code>dead</code> and <code>gameOver</code> to false. It then subtracts damage from the player's health. If the health is less than or equal to zero, it sets health to zero and marks the player as dead. If the player is dead, it decrements their lives by one. If the player has no lives left, it sets <code>gameOver</code> to true. Otherwise, it calls the <code>respawn()</code> module. Finally, if <code>gameOver</code> is true, it calls the <code>gameOverScreen()</code> module.</p>

Justification

This algorithm is to allow the player to get hit and take damage, i.e. undergo a reduction to their health. It reduces the player's health by the damage inflicted by an enemy. It then checks if the player's health is down to 0 in which case the player dies and loses a life. If the player has no lives left then the game over screen should be displayed through the call of the required module.

Link to other modules

This module extends the player class/module and links to the `enemyAttack` module as it is called every time an enemy lands a successful strike on the player. It also links to the `respawn()` module if the player dies but has more lives left, and the `gameOver()` module if the player runs out of lives.

Data structures & key variables

Variable / Data structure	Validation	Description & Justification	Type
<code>dead</code>	Must be only true or false at any one time, as the player can either be dead	Used to denote whether the player dies and loses a life or not, i.e. when the player takes damage, if they have	Boolean

	or alive at any one time.	taken sufficient damage to reduce their health to zero, then dead = true.	
gameOver	Must only be true or false, gameOver is defined as the point when the player runs out of lives, so the player either has more than 0 lives, or has 0 lives, ie. gameOver	Used to denote whether the player has run out of lives and the current game should end for a new one to start (if the player wishes). If the player has more than 0 lives then this is false, however if they have 0 lives, then gameOver = true.	Boolean

Iterative development module testing

<u>Input</u>	<u>Expected output</u>	<u>Justification</u>
Player starts with 3 lives, 80 health and takes 30 damage (Normal)	Player health reduced to 50.	Here, the player has taken damage so health is reduced by the damage amount as required, however, as the health is still above 0, the player does not die and lose a life, and so gameOver cannot occur either
Player starts with 3 lives, 20 health and takes 40 damage (Normal)	Player health is reduced to 0 and player loses a life and so respawns with 2 lives remaining.	Player takes 40 damage but health is less than that (20). So player dies and loses a life but respawns as they still have 2 more lives
Player starts with 1 lives, 10 health and takes 25 damage (Normal)	Player health is reduced to 0 and player loses a life. gameOverScreen() is called as the player now has 0 lives remaining.	Here, the player takes 25 damage but this is more than the remaining health (10) so the player dies and loses a life. However, the player now has 0 lives left and so instead of respawning, the game over screen is displayed.

Player Refuel

Pseudo code	Flowchart
-------------	-----------

```

function playerRefuel(refuelStations[], player)
    standing = false
    refuel = false
    i = 0
    currentRefuelStation = none

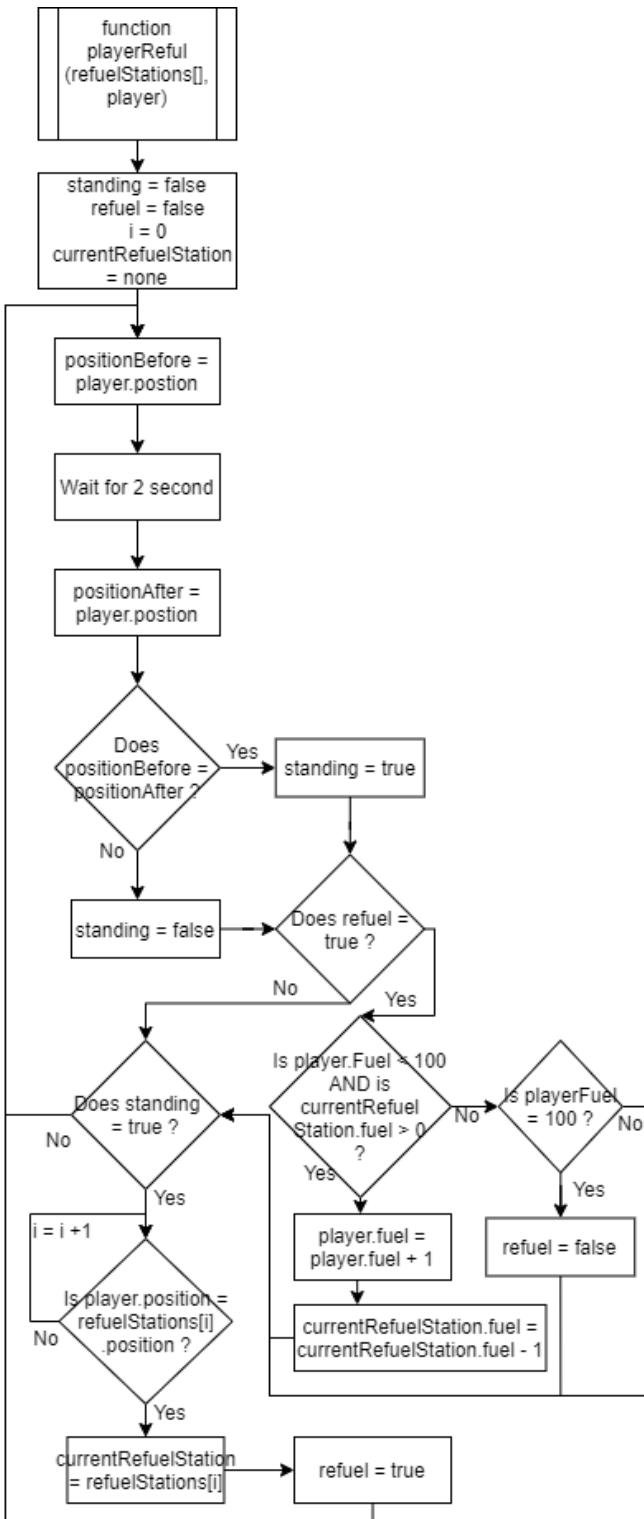
    while true:
        positionBefore = player.position
        waitForTime(2)
        positionAfter = player.position
        if positionBefore == positionAfter
            standing = true
        else
            standing = false

        if refuel
            if playerFuel < 100
                if currentRefuelStation.fuel > 0
                    player.fuel += 1
                    currentRefuelStation.fuel -= 1
            if player.fuel == 100
                refuel = false

        if standing
            for i in range (0, refuelStations.length)
                if player.position == refuelStations[i].position
                    currentRefuelStation = refuelStations[i]
                    refuel = true

// note: has parameter player however, using
// an object oriented paradigm, jump will extend
// the player class

```



Justification

This algorithm is used to allow the player to interact/use a refuel station. To use these, 2 conditions must be met; that is the player must be standing on the refuel station, and the player must be standing still. Since both these conditions need to be met, the order in which they are checked does not matter, and for the sake of efficiency we can choose to only check one if the other is first met. To check whether the player is standing on a refuel station, the algorithm has to loop over each of the stations and evaluate their position compared to the players. However, depending on the number of stations this can be quite costly in terms of performance, compared to checking if the player is standing still. So this is done first and only if this is met, the condition where the player and station's position must be the same is checked. To see if the player is standing still their position is simply evaluated before and after a 2 second gap, if it is the same, they are standing still. To check if the player is standing on a refuel station, each station is looped over and if its position is equal to that of the player, then refuel is set to true, since now both fundamental conditions are met. To refuel, it is checked if the player's fuel is below 100, i.e. they have space for more fuel and if it is, then the player's fuel is increased and the station's fuel store is decreased, both by 1 every 2 secs.

Link to other modules

Will extend the base player class but no direct links - this is a module that runs continuously during gameplay so is called when the player enters gameplay, checking if the player is trying to refuel.

Data structures & key variables

<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<i>standing</i>	Must be either true or false at any one time to dictate whether the player is standing or not.	Used to check whether the player is standing or not, if they are, then it can be further checked if they are standing on a station, but using this variable means that this is only checked if the player is first standing.	Boolean
<i>refuel</i>	Must be either true or false at any one time to dictate whether the player is trying to refuel or not.	Used to check whether the player can refuel or not, i.e. are they standing (still) on a refuel station. If they are, and they also have <100 fuel, then their fuel can be increased.	Boolean
<i>i</i>	Must be between 0 and the length of the list of refuel stations.	Used as a counter to loop through the list of refuel stations, so that their position can be evaluated to check if it's equal to the player's position.	Integer
<i>currentRefuelStation</i>	This variable takes the value of one of the refuel stations from the list of them, and must only represent one station at a time.	Stores the station gameobject/class that the player is standing on / using. This is stored so that the fuel value in the station can be reduced as it is used by the player.	Game object / refuel Station class

<i>positionBefore</i>	Must be a point in 3D space where the player gameobject exists.	Used to store the player's position before a 2 second wait, used to check if equal to the player's position after the 2 second wait, which would mean the player is standing still.	List (of coordinates) (floats)
<i>positionAfter</i>	Must be a point in 3D space where the player gameobject exists.	Used to store the player's position after a 2 second wait, used to check if equal to the player's position before the 2 second wait, which would mean the player is standing still.	List (of coordinates) (floats)

Iterative development module testing

<u>Input</u>	<u>Expected output</u>	<u>Justification</u>
Player stands still, not on a refuel station (Normal)	Player's fuel does not increase, all fuel stations retain fuel level.	Even though the player is standing still, they are not standing at a refuel station so both conditions are not met and so the player should not refuel.
Player, with less than 100 fuel, e.g. 50, stands still on a refuel station, with 50 fuel (Normal)	Player's fuel increases by 1 every 2 seconds and the station in use undergoes a decrease in fuel by the same amount.	The player stands still on a refuel station so they can refuel. Their fuel is also not at max (100), therefore there is space for more fuel and so more fuel is added, and this amount is reduced from the fuel store of the station that the player is using
Player, with 100 fuel, stands still on a refuel station with 50 fuel (Normal)	Player's fuel does not increase, all fuel stations retain fuel level.	The player stands still on a refuel station, i.e. both conditions are met so the player can refuel, however, the player's fuel is at max capacity (100) and there is no space for more fuel, so the player doesn't refuel and the station's fuel remains same.
Player, with less than 100 fuel, stands still on a refuel station with 0 fuel (Normal)	Player's fuel does not increase, all fuel stations retain fuel level.	The player stands still on a refuel station, i.e. both conditions are met so the player can refuel, however, there is no more fuel remaining in the refuel station so the player cannot increase their fuel, and the station's fuel capacity remains 0.

Environment generator

Pseudo code	Flowchart
--------------------	------------------

```

function environmentGenerator(room)

    if room = 0
        levelRooms = []

    currentScene = new Scene()
    levelRooms.append(currentScene)

    biomes = [
        town, city, forest,
        rainForest, mountainous, desert]
    spaceBetween = 15
    ground = green

    enemyCount = randomNum(10, 20)

    length = 200
    width = 200
    i = 0
    j = 0

    decideBiome()
    generateMesh()
    generateWalls()
    environmentFeatures()
    generateFuelStations()
    spawnEnemies()

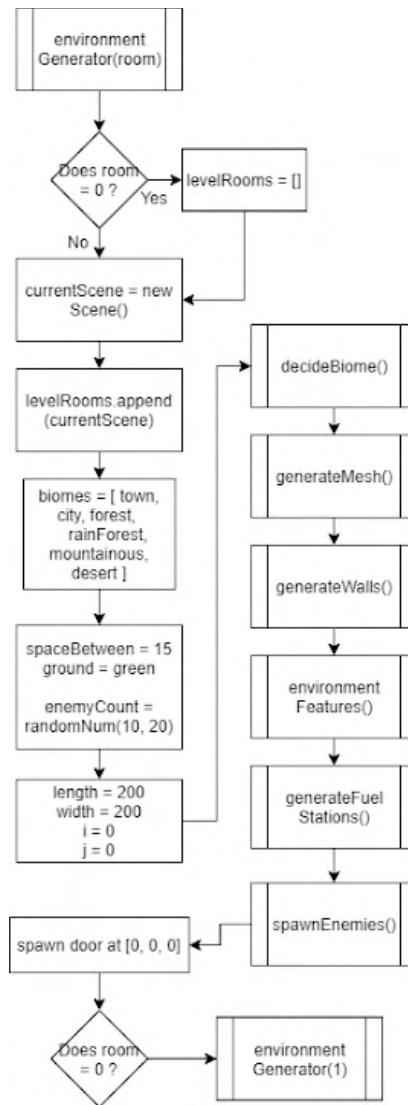
    instantiateGameObject(door, [0, 0, 0])

    if room = 0
        environmentGenerator(1)

    instantiateGameObject(player, [25, 10, 25])

// note: this module has been separated into
multiple functions for the sake of clarity

```



```

function decideBiome()

    biome = biomes[randomNum(biomes.length-1)]

    if biome = town
        envFeatures = [house, boulder]
        ground = brown

    else if biome = city
        envFeatures = [skyscraper, multiStory]
        ground = black

    else if biome = forest
        envFeatures = [tree, bush]

    else if biome = rainForest
        envFeatures = [tallTree, quickSand]

    else if biome = mountainous
        envFeatures = [mountain, tree]
        spaceBetween = 30
        ground = grey

    else if biome = desert
        envFeatures = [pyramid, dune, cacti]
        spaceBetween = 30
        ground = yellow

function generateMesh()

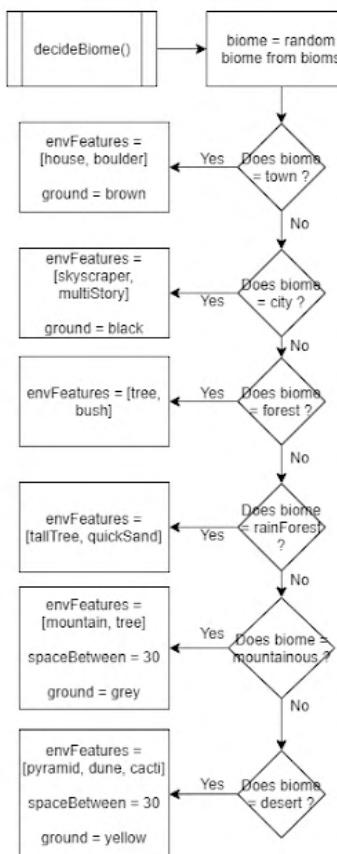
    mesh = new Mesh()
    mesh.color = ground
    vertices = []

    for z in range (0, width)
        for x in range (0, length)

            y = perlinNoise(x, z)
            vertices[i] = [x, y, z]
            i = i + 1

    for vert in vertices
        mesh.vertex = vertices[vert]
        mesh.drawTriangle(vert)

```



```

function generateWalls()

    instantiateGameObject(wall, [0, 0, 0])
    instantiateGameObject(wall, [length, 0, 0])
    instantiateGameObject(wall, [length, 0, width])
    instantiateGameObject(wall, [0, 0, width])

function environmentFeatures()

    for z in range (10, width-10, spaceBetween)
        for x in range (10, length-10, spaceBetween)
            index = randomNum(envFeatures.length-1)
            feature = envFeatures[index]
            instantiateGameObject(feature, [x, 0, z])
            feature.rotation.z = randomNum(360)

function generateFuelStations()

    fuelStations = randomNum(4)
    stationPos = none
    prevStationPos = none

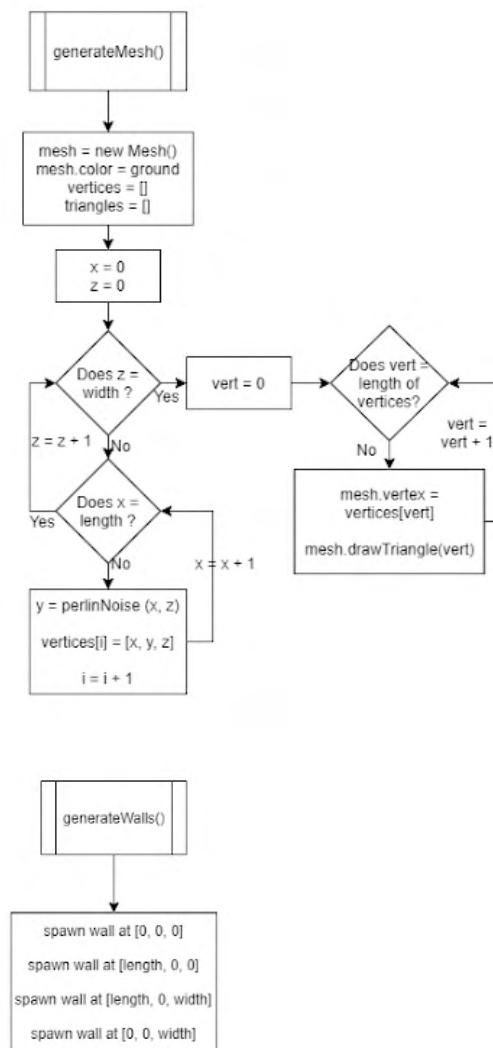
    for fs in range (0, fuelStations)
        while stationPos = prevStationPos
            stationPos = [randomNum(10), 0, randomNum(10)]
            prevStationPos = stationPos
            instantiateGameObject(fuelStation, stationPos)

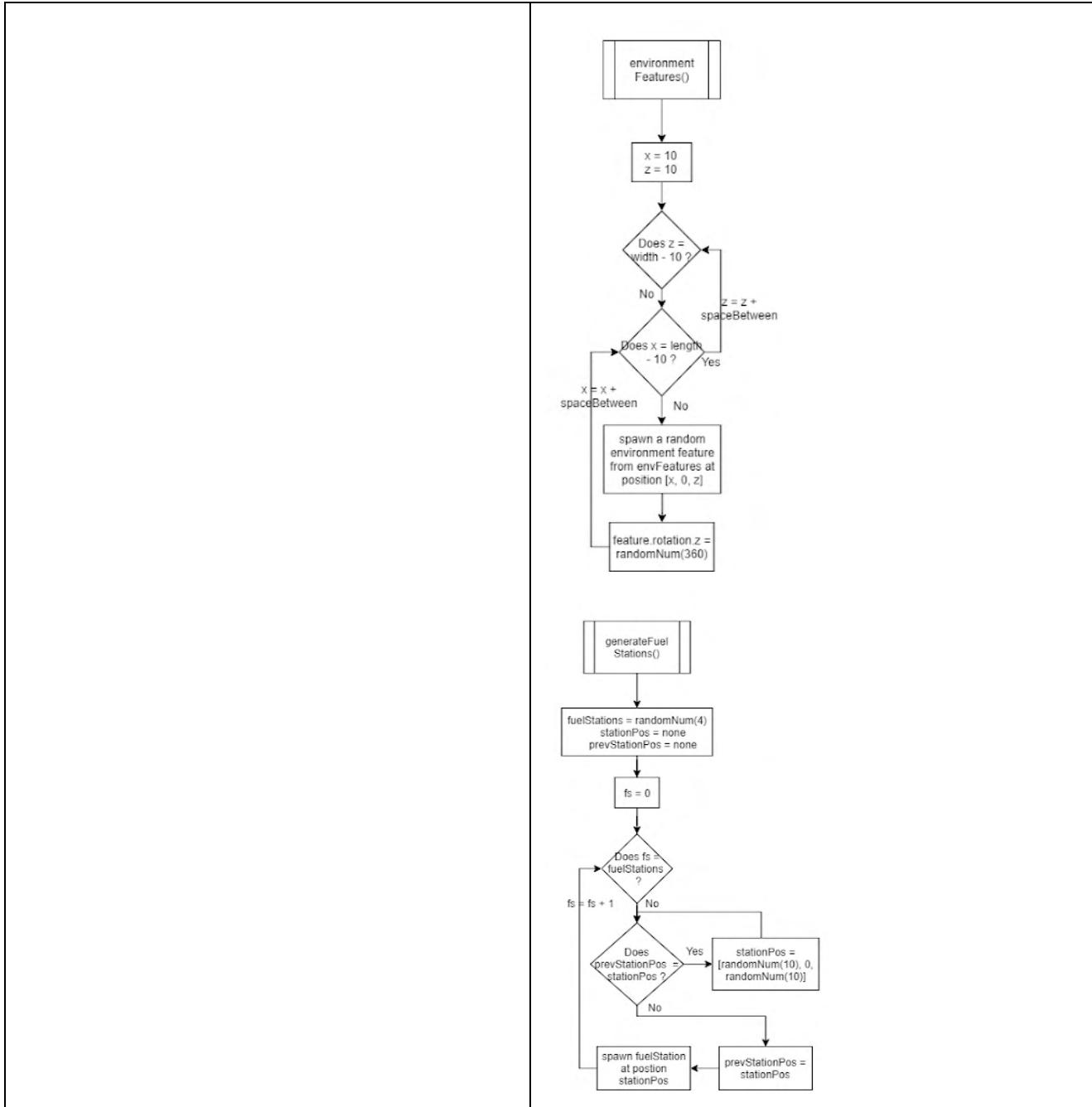
function spawnEnemies()

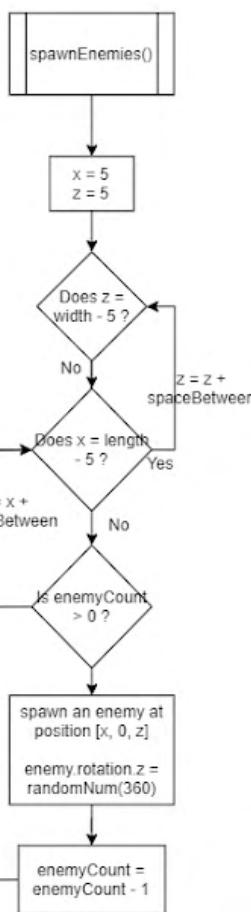
    for z in range (5, width-5, spaceBetween)
        for x in range (5, length-5, spaceBetween)

            if enemyCount > 0
                instantiateGameObject(enemy, [x, 0, z])
                enemy.rotation.z = randomNum(360)
                enemyCount = enemyCount - 1

```







Justification

This algorithm sets up / generates the map for the player to play on. This must be generated procedurally everytime to produce a different level every time for an infinite supply of levels. Therefore a variety of randomness is applied to many aspects of the level's generation. First of all, a random biome is chosen for the current room out of a list of 6 biomes each with specific, distinct environment features. Then the base / mesh is generated and this uses the PerlinNoise function to return a pseudo-random value which is assigned to the y-value, i.e. the height of each vertex of the mesh. Along with the vertices and the triangles generated to connect them, this generates random, realistic terrain for the level. Then the 4 walls are generated, however, there is no random aspect applied to this. Next, the biome specific environment features are randomly inserted into the level, this is done by randomly selecting a value from the list of environment features for the given biome. Once randomly chosen, each gameobject is added at specific intervals but leave a gap of 10 units around the border of the map as this is reserved for the door to the other room of the level as well as refuel stations. These refuel stations also include a random element, a random number of them is chosen to be in a room/level, up to 4. These stations are then randomly positioned somewhere along the border of the map, however, it is first checked that a station does not already exist in the place where a new one is to be spawned. Next the enemies are spawned, and again, to add an element of random, a random amount of enemies, between 10 and 20 is chosen and subsequently

spawned. Note, these spawn with an offset of 5 from the walls compared to the 10 of the environment features, however, they use the same space between each one. Therefore, it is ensured that no enemies spawn on top of or inside any environment gameobjects. Note that to generate 2 rooms, recursion is utilised, the same environmentGenerator is called after generating the first room to generate the second, however it is ensured that no more than 2 rooms are generated by passing the parameter 1 which terminates the function after 2 calls. The door, positioned in the corner of each room, is used to switch between rooms.

Link to other modules

This module is called when the player selects and enters gameplay; it is the first module to run to generate a level the player can actually play on.

Data structures & key variables

<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<i>rooms</i>	Must be either 1 or 0	This will be passed as a parameter to check whether 1 room has already been generated, in which case only one more room is generated, but if 2 rooms have already been generated then no more rooms are generated.	Integer
<i>levelRooms</i>	Must be a list of size less than or equal to 2, as this is the maximum number of rooms allowed in a level.	This is an array which stores the scenes in the current level so that when the player tries to enter the other room, the scene can change.	List of scenes in current level
<i>biomes</i>	A constant list defined which must have more than 1 element.	This is the list of biomes that could be selected to be applied to the current level. The algorithm checks which biome (string) is selected and alters/sets variables accordingly, such as ground colour and the environment features	List of biomes (strings)
<i>biome</i>	Must be a string value from the biomes array/list.	This stores the biome which has been randomly selected from the biomes list so the specific features of the biome can be applied.	String
<i>spaceBetween</i>	A preset constant, set to 15 as this works for most biomes, but is changed for others. But must be smaller than the length and width.	This is the distance between each environment feature to be spawned, preset to 15, but for some biomes with larger features it is increased if that biome is selected.	Float

<i>ground</i>	Must be a string which corresponds to a colour.	This is the colour the ground is set to depending on the biome chosen.	String
<i>enemyCount</i>	Must be a number between 10 and 20.	This is the number of enemies to be spawned into the room, randomly decided.	Integer
<i>length</i>	Preset value set to 200 but must be greater than 0.	This is the length of the room to be generated.	Float
<i>width</i>	Preset value set to 200 but must be greater than 0.	This is the width of the room to be generated.	Float
<i>i</i>	Must be greater than 0.	This is a counter used to incrementally add values to the list of vertices.	Integer
<i>envFeatures</i>	Must be a list of size greater than 0.	This is a list which holds the environment features to be spawned into the room depending on the biome. Random elements are selected to be spawned on each occasion.	List of features (strings)
<i>vertices</i>	Must be a list of size $(\text{length} + 1) \times (\text{width} + 1)$ As this is the required vertices for a plane of grid size length x width.	This is a list of points, i.e. vertices which are required to generate a base/mesh. Triangles are generated to join these vertices to each other and form the actual mesh.	List of points / 3D coordinates (floats)
<i>x</i>	Must be an integer greater than 0.	This is a counter which loops through the length of the base to generate vertex points equally across the length.	Integer
<i>y</i>	Must be a float between 0 and 1.	This is a float which is used to represent the height of each vertex using the perlinNoise function to get a pseudo-random value between 0 and 1.	Float
<i>z</i>	Must be an integer greater than 0.	This is a counter which loops through the width of the base to generate vertex points equally across the width.	Integer
<i>vert</i>	Must be an integer greater than 0.	Used as a counter to loop through each vertex to generate a triangle between the vertices.	Integer

<i>fuelStations</i>	Must be a positive integer less than or equal to 4.	This is the number of fuel stations to be spawned in the room, randomly decided.	Integer
<i>stationPos</i>	Must be a point in 3D which must have a randomly generated x-value between 0 and 10, a y-value of 0 and a z-value also between 0 and 10. This must also not be equal to the position of an already existing station, validated using the <i>prevStationPos</i> variable.	This is the position of a fuel station to be spawned, and this must be on the floor, along the border of the room, hence the and z values must be between 0 and 10.	List of coordinates (floats)
<i>prevStationPos</i>	Must be equal to the 3D position of a station last spawned.	Holds the position of the already spawned station and if the next station to be spawned has coincidentally randomly chosen the same position then the condition of <i>stationPos</i> being equal to <i>prevStation</i> is met and so a new position is generated, until a different position is generated.	List of coordinates (floats)

Iterative development module testing

Input	Expected output	Justification
Select play from menu (start gameplay)	<ul style="list-style-type: none"> - Base mesh generated successfully - Random biome applied successfully - Walls spawned correctly - Biome specific environment features spawned correctly - Fuel stations spawned correctly - Door/link to other room generated successfully - Enemies spawned successfully - Player spawned successfully - Process repeated to generate second room - Levels are different, i.e. randomness actually applied/works 	This algorithm has no real input as key factors such as the biome are all decided randomly to generate different levels each time. For this algorithm to be considered a success, it simply needs to generate everything correctly, in the right place and not interfere with anything else. This test will be carried out multiple times to test the randomness of the generation, i.e. to check that randomness is actually applied and that the level is not the same every time.

Enemies

Pseudo code	Flowchart		
<pre>function enemy() position = [] rotation = [] health = 75 speed = 5 dead = false damage = 20 range = 8 delay = 2 // this is really a class, which will be extended by other classes, designed as a module for reasons explained earlier e.g. testing.</pre>	<pre> graph TD A[enemy()] --> B["position = []\nrotation = []\nhealth = 75\nspeed = 5\ndamage = 20\ndeath = false\range = 8\ndelay = 2"] </pre>		
Justification			
<p>This module defines an enemy and its attributes. These attributes are the base attributes required to carry out all the enemies' requirements. It simply defines these attributes so that they can be altered by other modules, which extend this class, to allow the enemies to actually fulfil their role, i.e. movement, attacking.</p>			
Link to other modules			
<p>Links to all the enemy modules (classes) which include;</p> <p>The Enemy attack class which will decrease the player's health attribute by the enemy's damage attribute if the enemy lands a successful attack.</p> <p>The Enemy movement class which will move and rotate the enemy, changing the enemy's rotation and position by a factor of the enemy's speed attribute.</p> <p>The Enemy damage class which will reduce the enemy's health attribute by a damage value.</p>			
Data structures & key variables			
Variable / Data structure	Validation	Description & Justification	Type
<i>position</i>	Must be a point in 3D space. The x and y values must be on the map and therefore $0 < x < \text{length}$ and $0 < z < \text{width}$. The enemies must remain on the ground, and therefore the y value	This array stores 3 values each representing the x, y and z coordinates of the enemy's position.	List of coordinates (floats)

	must = 0.		
<i>rotation</i>	Must be a 3D angle, however the x value and z value must be 0 while the enemy is alive.	This is the angle of rotation the enemy is at, at any time. The x and z should only change when the enemy dies and falls over	List of 3 rotation values (x, y, z) (floats)
<i>health</i>	Must be a positive integer less than or equal to 75.	Used to denote dead or alive stat (alive if health>0), enemy loses health when hit by the player.	Integer
<i>speed</i>	Must be a constant to feel intuitive and predictable, as the enemies move around on foot.	This is the rate at which the enemy moves around, i.e. the rate of change of distance.	Float
<i>damage</i>	Must be a positive integer less than the player's health.	This is the amount of health lost by the player if they are successfully hit by an enemy.	Integer
<i>dead</i>	Must either be true or false as the enemy can only be dead or alive at any one time. At spawn, this is preset to false.	This is used to denote whether the enemy is dead or alive, dead is true if their health drops to, or below, 0.	Boolean
<i>range</i>	Must be a positive integer, preset to a value of 8.	This is how far ahead the enemy can attack, i.e. the enemy can attack the player if the player is within 8 units of the enemy.	Integer
<i>delay</i>	Must be a positive integer, preset to a value of 2	This is the time delay between the enemy's attacks, i.e. after an attack, the enemy must wait 2 seconds before being able to attack again.	Integer

Iterative development module testing

<u>Input</u>	<u>Expected output</u>	<u>Justification</u>
-		

Enemy Attack

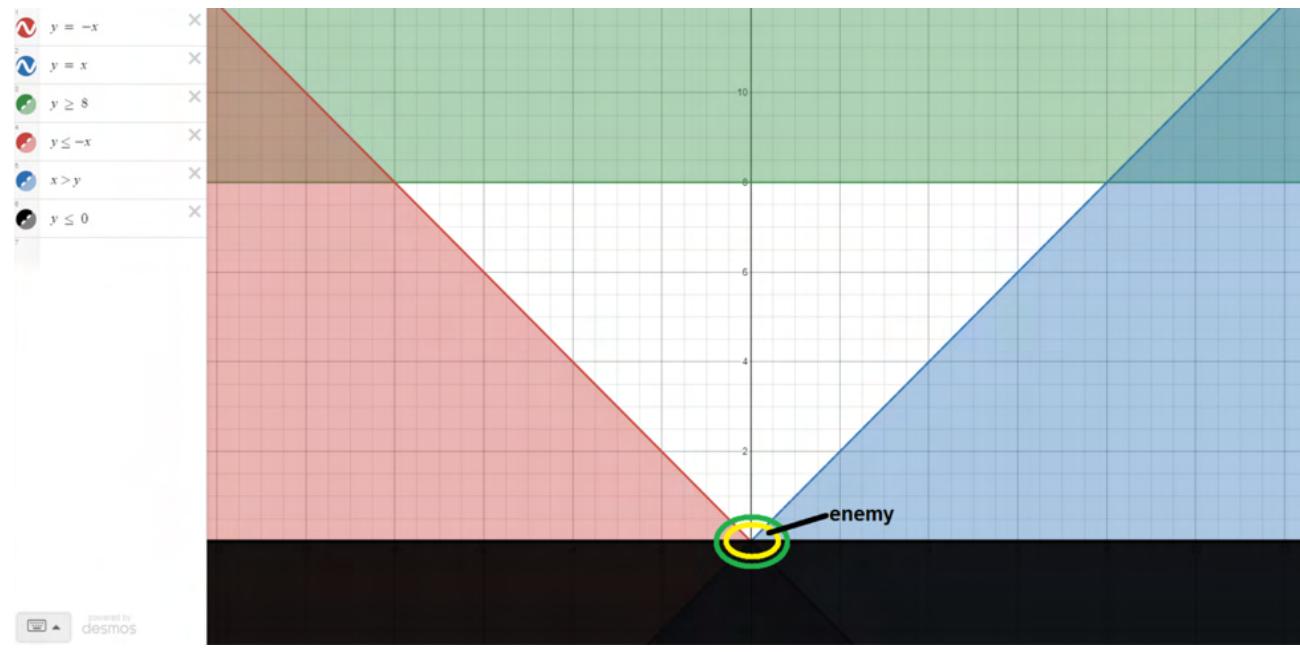
Pseudo code	Flowchart
<pre> functions enemyAttack(enemy, player) hit1 = false hit2 = false playerPos = player.position enemyPos = enemy.position while true if !hit1 if playerPos.y > enemy.range hit1 = false else if playerPos.z < enemy.range if playerPos.x < enemyPos.x if playerPos.z > -playerPos.x hit1 = true if playerPos.x > enemyPos.x if playerPos.z < playerPos.y hit1 = true if hit1 enemy.speed = enemy.speed / 1.5 playAnimation(enemyAttack) waitTime(3) if !hit2 stretchRange = enemy.range * 1.5 if playerPos-enemyPos < stretchRange hit2 = true if hit2 player.Damage(enemy.damage) hit2 = false waitTime(enemy.delay) hit1 = false //note: parameters are passed, however, when implemented with an object oriented paradigm, this will extend the base enemy class, i.e. be a subclass of the base enemy class. </pre>	<pre> graph TD Start[enemyAttack(enemy, player)] --> Init[hit1 = false hit2 = false] Init --> SetPos[playerPos = player.position enemyPos = enemy.position] SetPos --> Hit1{Does hit1 = true?} Hit1 -- No --> RangeY{Is playerPos.y > enemy.range?} RangeY -- Yes --> Hit1 RangeY -- No --> Hit2{Is playerPos.z < enemyPos.z?} Hit2 -- No --> Hit1 Hit2 -- Yes --> Hit3{Is playerPos.x < enemyPos.x?} Hit3 -- Yes --> Hit1 Hit3 -- No --> Hit4{Is playerPos.z > playerPos.x?} Hit4 -- Yes --> Hit1 Hit4 -- No --> Hit5{Is playerPos.z < playerPos.y?} Hit5 -- Yes --> Hit1 Hit5 -- No --> Hit6{Does hit2 = true?} Hit6 -- Yes --> Damage[player.Damage(enemy.damage)] Damage --> Hit7[hit2 = false] Hit7 --> Wait3[Wait for 3 seconds] Wait3 --> Hit8{Does hit2 = true?} Hit8 -- Yes --> Hit9[hit1 = true] Hit9 --> Speed[enemy.speed = enemy.speed / 1.5] Speed --> Animation[Play enemyAttack animation] Animation --> Wait3 Hit8 -- No --> Stretch[stretchRange = enemy.range * 1.5] Stretch --> Dist{Is the distance between player and enemy less than stretchRange?} Dist -- Yes --> Hit10[hit2 = true] Dist -- No --> Hit11[hit1 = false] Hit11 --> WaitDelay[Wait for (enemy.delay) seconds] WaitDelay --> Hit12[hit1 = false] </pre>

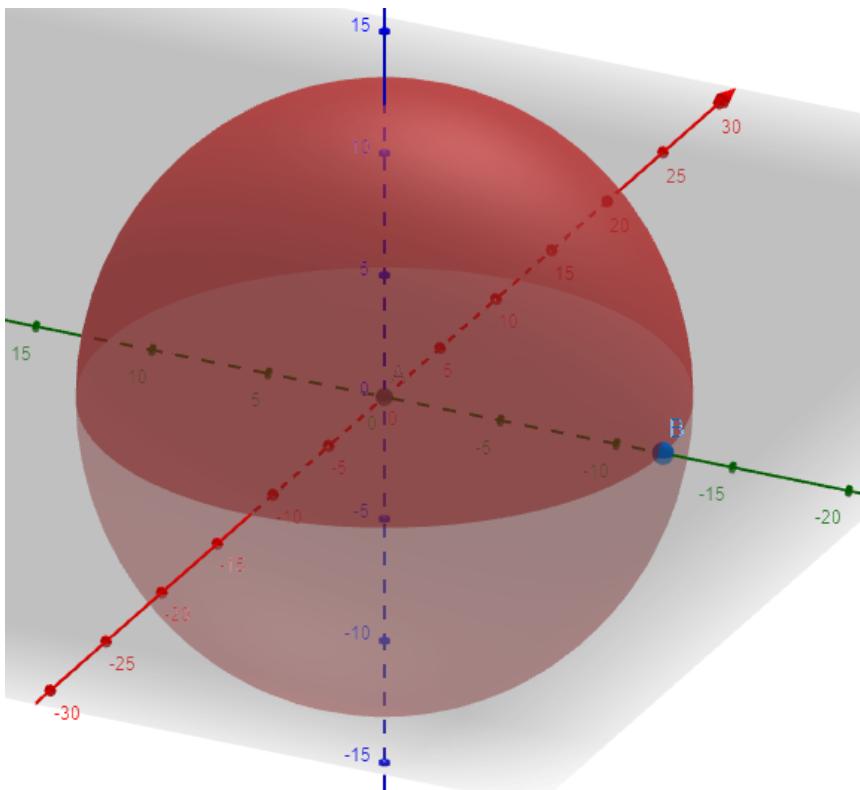
Justification

This algorithm is used to check if an enemy can attack the player and in which case to do so. The enemy can attack the player if;

The player is vertically in range of the enemy,
initially the player is in a triangular (prism) forward POV of the enemy,
the player is still in range of the enemy after a 3 second wait, however at this point the range is extended and changed to a sphere with the enemy at the center.

The algorithm first checks if the player's y position is greater than the enemy's range, if not, it continues to check if the other conditions are met. To check if the player is within the triangular POV, the program checks whether the player is within the maximum range (`enemy.range`), if so the program uses different conditions depending on whether the player is on the left or right of the enemy. It then compares the x and y values to see whether the player is located inside this triangular POV or not. These mathematical inequalities are best explained by the graphs below. (See bottom of this justification section.) If the player is within this triangular POV boundary, then `hit1` is set to true and the algorithm can progress to check the next conditions. So next, as long as `hit1` is true, the enemy attempts an attack, this involves reducing the speed, to allow the player to attack the enemy more easily, and then waiting 3 seconds before damaging the player if they are still in range. The range is now changed to a sphere with radius equal to the enemy range increased by a factor of 1.5, so if after the 3 seconds in which the player can escape, if they are still anywhere within this new range, checked by finding the modulus (i.e. magnitude) of the difference of the 2 positions (of the enemy and the player), `hit2` is set to true, and so then the player is damaged, `hit2` is now set to false. The algorithm then halts for seconds equal to the `enemy.delay`, i.e. the time the enemy has to wait between each attack. Then `hit1` is set to false so that the whole process has to now be repeated for another hit.





Link to other modules

Will extend the base enemy class but no direct links - this is a module that runs continuously during gameplay so is called when the player enters gameplay, checking for and executing attacks on the player if legal/viable.

Data structures & key variables

Variable / Data structure	Validation	Description & Justification	Type
<i>hit1</i>	Can only be true or false, the player can either meet the initial range conditions, or not.	This bool is used to check whether the player is initially in range for an attack, i.e. before the time wait in which the player has some time (3 seconds) to try and escape.	Boolean
<i>hit2</i>	Can only be true or false, the player can either meet the secondary range conditions, or not.	This bool is used to check whether the player is in range for an attack the second time, i.e. after the time wait when the range is bigger and a sphere instead of a triangle (triangular prism).	Boolean

Iterative development module testing		
<u>Input</u>	<u>Expected output</u>	<u>Justification</u>
As the player, stay out of the enemy range (Normal)	Enemy does not attack and the player does not take damage.	If the player is not in range of the enemy, even the initial range, the enemy shouldn't even attempt an attack so player health should remain the same.
As the player, be horizontally in range of the enemy (i.e. in the triangle) but be much higher than the enemy's range. (Normal)	Enemy does not attack and the player does not take damage.	Even though the player is horizontally close enough, they are too high up, therefore enemy should not even attempt an attack and player should not lose any health.
As the player, be horizontally and vertically in range of the enemy, but within the 3 second time period, move out of even the larger hit zone / range. (Normal)	Enemy attempts to attack the player, however, the player does not take damage.	Initially the player is in range, therefore the enemy will attempt to attack, however, during the 3 second time period, the player has moved far enough from the enemy such that the player is not within even the larger hit zone / range, and so the enemy's attack essentially 'misses' and the player does not take damage, i.e. health stays the same.
As the player be horizontally and vertically in range of the enemy, then during the 3 second time period, move out of the initial hit zone but stay within the second, larger zone. (Normal)	Enemy attempts to attack the player and succeeds, i.e. the player takes damage.	Initially the player is in range, therefore the enemy will attempt to attack the player, and will succeed in damaging the player, i.e. decreasing the player's health, as although the player moves out of the initial hit zone / range, they are still within the second, larger zone which is checked for the actual hit.

Enemy Damage

Pseudo code

Flowchart

```

function enemyDamage(enemy, damage)

    dead = false

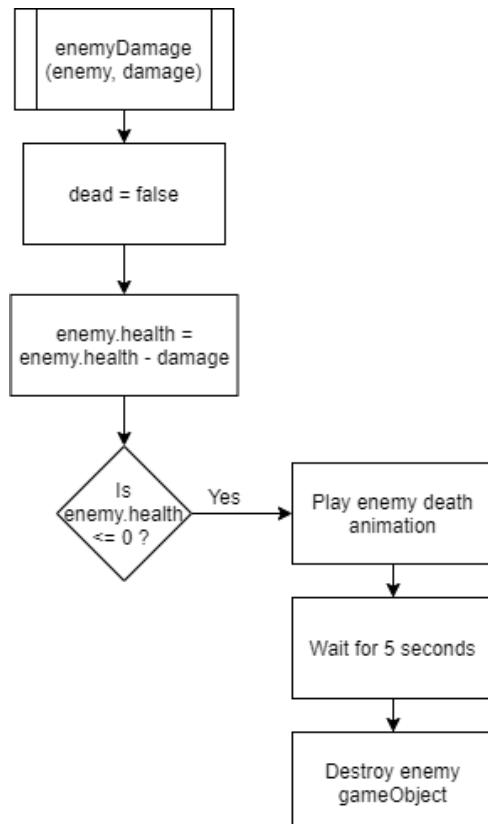
    enemy.health -= damage

    if enemy.health <= 0
        enemy.health = 0
        dead = true

    if dead
        playAnimation(enemyDeath)
        waitForTime(5)
        destroyGameObject(enemy)

```

//note: enemy parameter passed, however, when implemented with an object oriented paradigm, this will extend the base enemy class, i.e. be a subclass of the base enemy class.



Justification

This is a module which damages the enemy by a given damage. It also checks the health of the enemy and if this is less than or equal to 0, then the algorithm sets the boolean `dead` status to true and kills/despawns this enemy.

Link to other modules

This module extends the enemy class/module and links to the playerAttack module as it is called every time the player lands a successful strike on the enemy.

Data structures & key variables

Variable / Data structure	Validation	Description & Justification	Type
<code>dead</code>	Must be either true or false as the enemy can either be dead, or not dead (alive)	This is a boolean used to denote whether an enemy is alive. This is true when the enemy's health is less than or equal to 0. If this is true then the enemy dies.	Boolean

Iterative development module testing

<u>Input</u>	<u>Expected output</u>	<u>Justification</u>
Enemy starts with 50 health, attack with player for 30 damage. (Normal)	Enemy health is reduced to 20, enemy does not die.	Here the enemy has been damaged but as the health is not reduced to 0 or less, the enemy does not die.
Enemy starts with 20 health, attack with player for 40 damage (Normal)	Enemy health is reduced to 0 and enemy dies after 5 secs. (despawns)	Here the enemy takes more damage than the health it had, therefore, health is reduced to below 0 (then reset to 0) and so the enemy also dies, and despawns after 5 seconds of this.
Enemy starts with 10 health, attack with player for 10 damage (Boundary)	Enemy health is reduced to 0 and enemy dies after 5 secs. (despawns)	Here the enemy takes equal damage to the health it had, therefore, health is reduced to 0 and so, due to the \leq condition, the enemy also dies, and despawns after 5 seconds of this.

Enemy Movement

Pseudo code	Flowchart
<pre> function enemyMovement(enemy, player) moveTowards = false enemyPos = enemy.position playerPos = player.position distance = 0 while true if enemyPos-playerPos < 100 moveTowards = true while moveTowards enemy.rotation.y = playerPos for i in range (0, 50) while distance < 10 enemy.rotation.y += enemy.left * 0.1 lineSight = castRay(enemy, enemy.forward) point = lineSight.hitpoint distance = point - enemyPos enemyPos += enemy.forward * enemy.speed if enemyPos-playerPos > 100 moveTowards = false if j MOD 50 = 0 enemy.rotation.y = randomNum(0, 360) j = j + 1 while distance < 10 enemy.rotation.y += enemy.left * 0.1 lineSight = castRay(enemy, enemy.forward) point = lineSight.hitpoint distance = point - enemyPos enemyPos += enemy.forward * enemy.speed //note: parameters are passed, however, when implemented with an object oriented paradigm, this will extend the base enemy class, i.e. be a subclass of the base enemy class. </pre>	
Justification	
<p>This algorithm is responsible for enemy movement, of which there are 2 types. Random movement is where the enemy moves around randomly but avoids obstacles in its path. This type of motion occurs when the enemy is really far from the player, > 100 units away. The other type is where the enemy moves towards the player and avoids obstacles in its way. This type of motion occurs when the enemy is close enough to the player, i.e. within 100 units. The type of movement to occur is decided using the moveTowards boolean variable, if the enemy is within 100 units of the player then this is set to true. In this case, a new game loop is created in which the enemy initially rotates towards the player and attempts to move forwards, by a factor of it's speed, for 50 frames before </p>	

rotating to look towards the player again, and progress towards the player. However, during this, the algorithm checks that the distance between the enemy and any obstacle in front of it is large enough such that the enemy can move forwards without having to worry about the obstacle yet. This is done using a ray shot from the enemy and seeing how far it got before colliding with an obstacle. If the distance is not large enough, (< 10) i.e. an obstacle is in the way of the enemy, the enemy keeps rotating to the left along the y-axis, until the path in front of it is clear. At the end of each cycle of this game loop of the enemy moving towards the player, the algorithm checks that the enemy is still in range, i.e. within 50 units, of the player, if not, moveTowards is set to false, breaking out of this game loop and essentially triggering the random movement. If the enemy is too far from the player for it to move towards the player, the enemy moves essentially randomly. This involves the enemy rotating to a random (y) orientation every 50 frames and moving forwards by a factor of its speed, if it can, i.e. if there are no obstacles in front of it, and this is done using the same distance checking code as before.

Link to other modules

Will extend the base enemy class but no direct links - this is a module that runs continuously during gameplay so is called when the player enters gameplay, continuously moving the enemies.

Data structures & key variables

<u>Variable / Data structure</u>	<u>Validation</u>	<u>Description & Justification</u>	<u>Type</u>
<i>moveTowards</i>	Must be either true or false, this method of movement must either be in use or not.	This boolean is used to decide between the 2 movement types, i.e. whether to apply movement towards player or random motion.	Boolean
<i>distance</i>	Must be a positive real number (float)	This is the absolute/modulus value of the difference between the position of the enemy and the position of an obstacle in front of it. If this value is small enough, then action is taken to avoid the obstacle.	Float
<i>lineSight</i>	Must be of length > 1 .	This is a ray, essentially a line which stores data such as length to a certain point, intersection point, etc. Used to get information about the point/position of an obstacle.	List of lists of co-ordinates (many points)
<i>point</i>	Must be a singular point in 3D space.	This is the point at which the lineSight ray intersects with an obstacle, so this is the point to be avoided by the enemy.	List of coordinates (for 1 point)
<i>j</i>	Must be a positive integer.	A counter used to check if 5 frames	Integer

		have passed, to change the enemy's y-orientation for random motion, i.e. if j divided by 5 doesn't give a remainder, then rotate.	
--	--	---	--

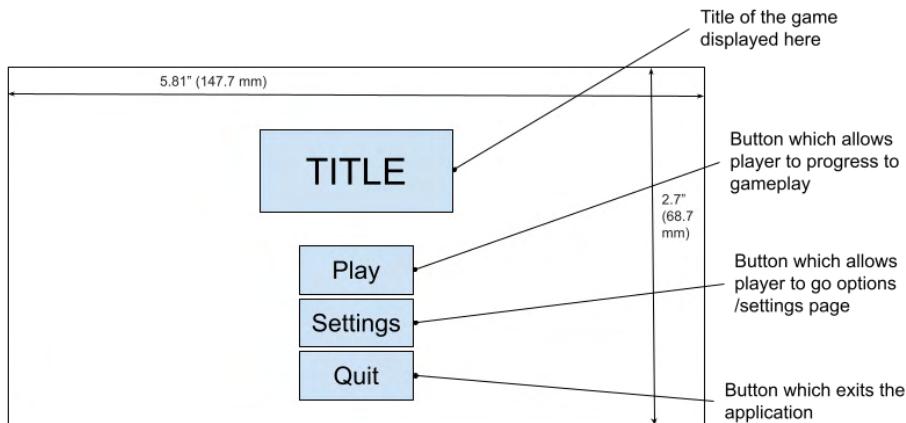
Iterative development module testing

<u>Input</u>	<u>Expected output</u>	<u>Justification</u>
As the player, be within 100 units of an enemy.	Enemy should move, generally towards the player, avoiding obstacles.	Here, the enemy is within range of the player to activate the move towards movement method instead of random motion. Therefore the enemy generally moves towards the player, avoiding any obstacles in the way.
As the player, be over 100 units away from an enemy.	Enemy should move in random directions, avoiding obstacles/	Here, the enemy is not within range of the player to activate the move towards movement method, so random motion occurs. The enemy moves randomly, changing direction randomly every 50 frames, and avoiding any obstacles in the way.

I presented all of these algorithms to the client, and after thoroughly reviewing the flowcharts designed specifically for their understanding, the clients were very happy with the modules and gave me the go-ahead for them all.

GUI & Usability features

Main Menu



Screen resolution chosen for my game is 5.81 by 2.7 inches, length and width respectively. This is because most mobile devices use this resolution to achieve roughly a 18:9 aspect ratio. By using this resolution, my game will be functional on almost all mobile devices.

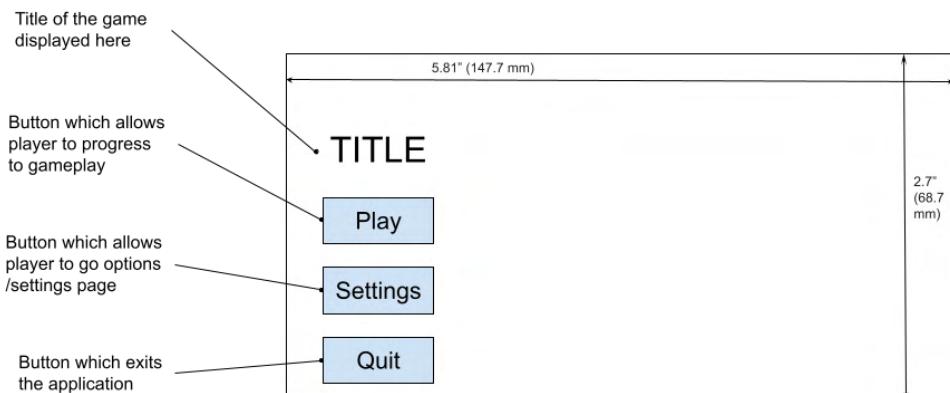
The menu is the first thing the player sees when they open the application. It includes the title of the game as well as 3 buttons; Play, Settings and Quit, all laid out in a clear manner, as a list respectively. The title is just some text, whereas the other 3 are buttons therefore are to be selected/pressed by the user as an input method.

I sent this design to my clients for review, and then received the following feedback:

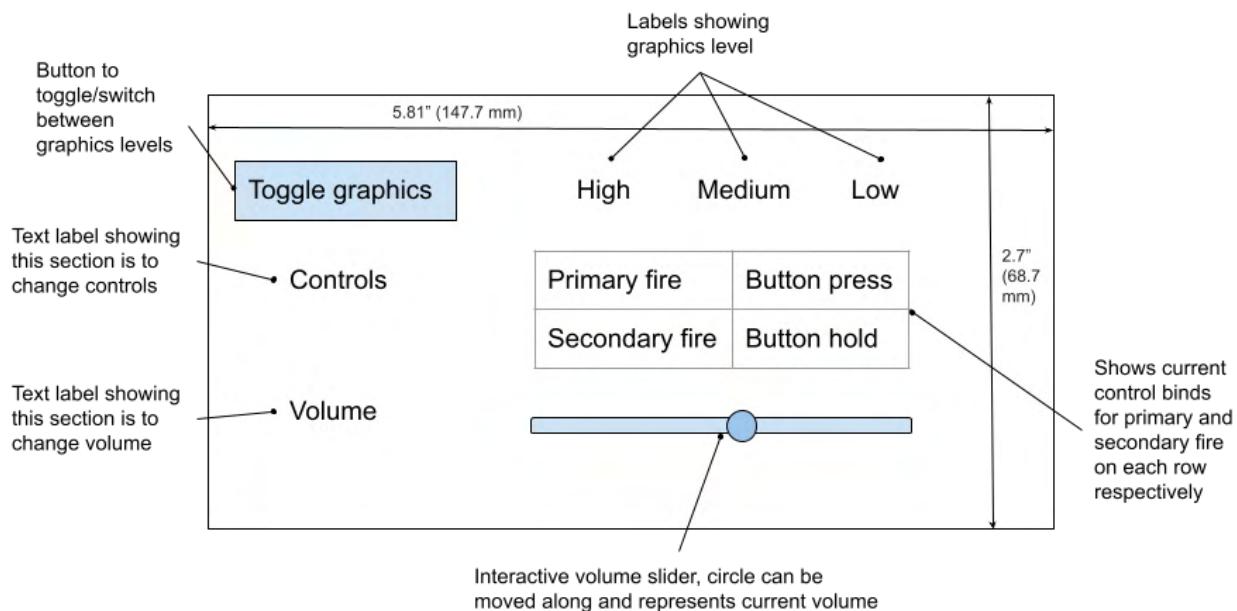
"Hi, we like your current design however, there are some changes we'd like to see;

- The title as just text, i.e. not in a textbox
- The title and buttons aligned to the left for more room to display background scenery
- The buttons spaced out more to avoid miss clicks and for clarity"

I then improved my design to meet these requirements:



Settings



The settings page allows the player to change 3 main settings; graphics level, controls and volume.

Graphics mode is changed by pressing a toggle button which switches between each one, having just one button makes the process easier and clearer and toggling between options won't take very long as there are only 3 graphics levels. The controls are shown as a table as you can easily see which control maps to which input by simply following each row.

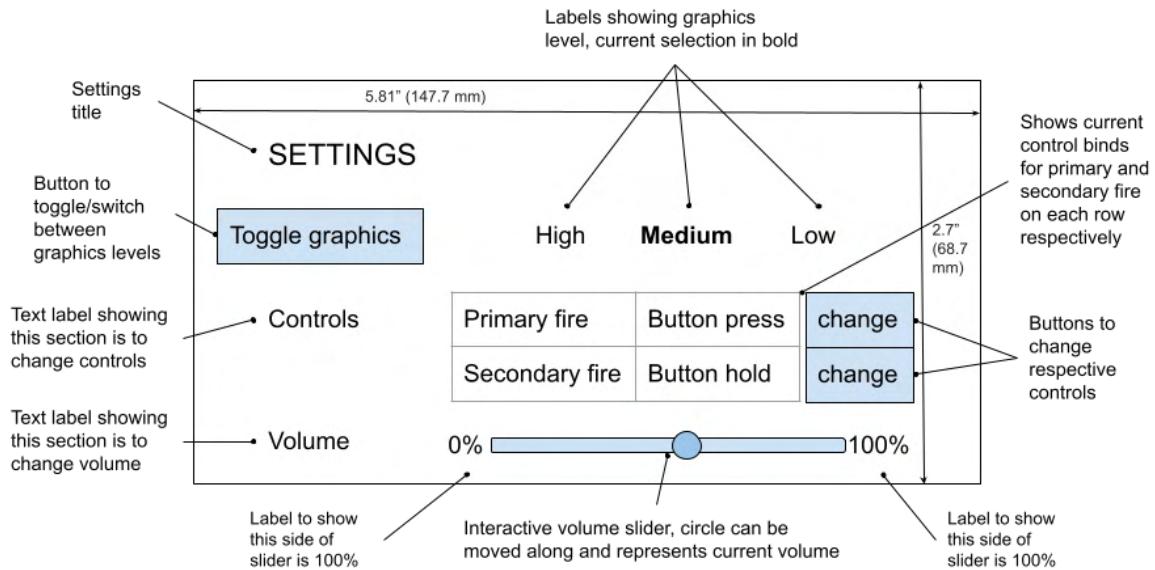
Furthermore, if new controls were to be added, extra rows can simply be added. The volume setting uses a slider, this is to automatically validate that the volume is between 0% and 100% and feels intuitive to the player to select a relative volume they are comfortable with.

I sent this design to my clients for review, and then received the following feedback:

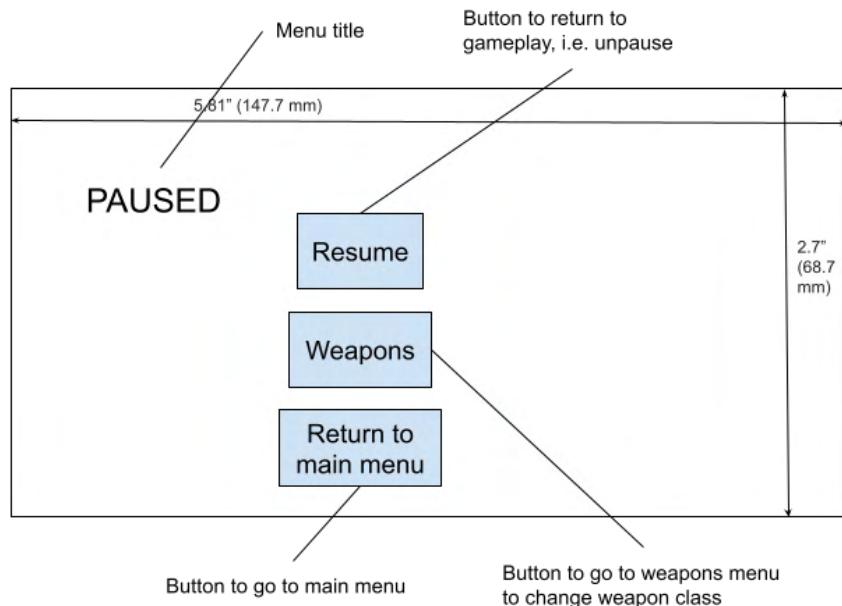
"Hi, thanks for sending us the design, here are some improvements we'd like:

- A settings title should be at the top of the page so that it's clear that this is a settings page, otherwise, just by looking at the page the player can't actually tell.
- How is the player meant to know which graphics level they are currently on? Possibly having the current graphics level in bold would help.
- How do you actually change the controls/bindings? There is no button for this.
- Can't tell which side of the volume slider is 0% and 100%, labels would clarify this."

I then improved my design to meet these requirements:



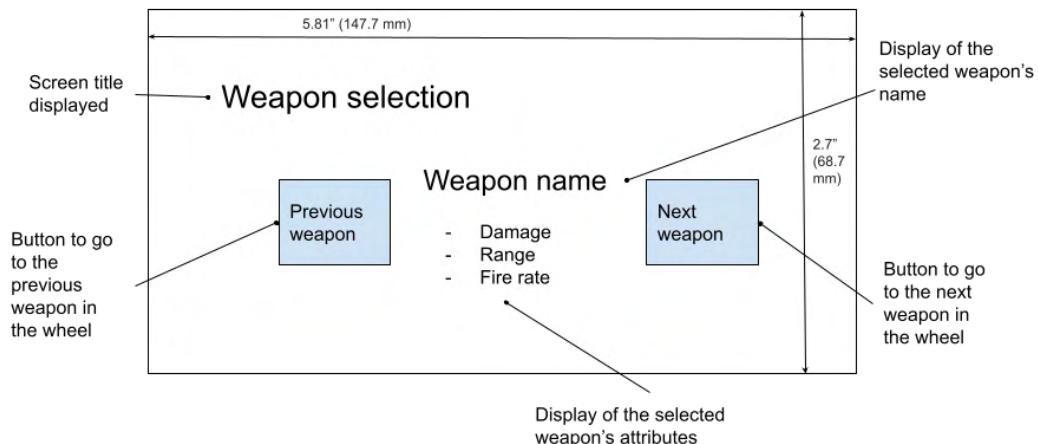
Pause menu



The pause menu is supposed to be simple as it is meant to be accessed quickly and easily. This is achieved using 3 simple, distinct buttons. Resume is the most commonly used button, therefore at the top, simply used to unpause. The weapons button is to take the player to the weapons menu, to change weapon class, if they press the button. The return to main menu button does what it says, further showing how clear and simple this menu is, it takes the player back to the main menu if they press it.

I sent this design to my clients for review and they said that they didn't have any improvements to suggest as they liked the simplicity.

Weapons menu



The weapons menu, used to select a weapon class to use, uses 2 buttons to flick through a wheel/loop of weapon classes, then each weapon's attributes are displayed below the weapon's

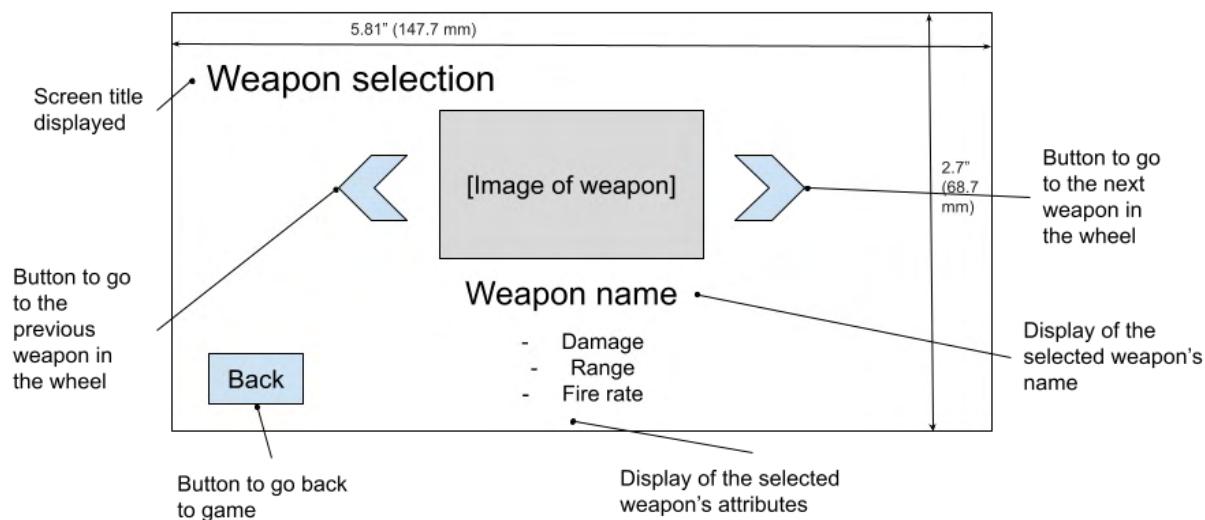
name for the player's information. Using a simple wheel means that there's only 2 buttons for the user to press rather than a list of grid of weapon classes of which the player has to click one. Furthermore, in this way, the attributes of each weapon can be displayed intuitively, instead of cramming all the information of all the weapon classes onto 1 screen.

I sent this design to my clients for review, and then received the following feedback:

"Hi, we like your design with the implementation of a wheel/loop, a few suggestions though;

- There should be an image of the weapon currently chosen so that the player gets more of a feel for what they're choosing.
- The previous/next weapon buttons should be changed to arrows instead of labelled buttons, for a simpler, intuitive look which is still easy to understand.
- How do you go back to the game or another screen? There should be a button for this."

I then improved my design to meet these requirements:



Website feedback form

We want your feedback...

Short written question

[textbox answer]

Multiple choice question

Option 1 Option 2 Option 3 ...

Multi-select question

Option 1 Option 2 Option 3 ...

Long written question

[textbox answer]

...

Note: As this is a HTML website, the resolution does not have to be considered.

The website feedback form will be a simple html form and feature a variety of question types as designed above. The short written question box will have a small textbox to subconsciously let the player know that this is meant to be a short response question. On the other hand a long written question will have a large text box to signal to the player that this is meant to be a longer response. Multiple choice questions and multi-select questions work in a similar way, the options are displayed and the user clicks on the input circle/box next to their answer. In the multiple choice questions, the user can only select 1 option whereas in the multi-select, the user can select multiple options. The questions will be provided by the clients, but these haven't been given as of yet.

I sent this design to my clients for review, and then received the following feedback:

"Hi Harsh, we like the structure/plan of the feedback form and are happy to keep it this way for now, however, we may request some changes in the future once the questions for the form have been finalised."

Usability designs note: The game will be implemented using the Unity game engine and so GUI and usability will be implemented through this. However, not all of the game screens have been designed as some, such as the gameplay, are to do with rendering rather than GUI design. Only the generally static screens in which the player interacts / enters any inputs, such as the menu screen, have been designed as graphical user interfaces.

Post development test plan

All tests done in the iterative testing ensure that the solution is functional and robust, the following tests further ensure this and instead test the program module's together as one program. Furthermore, testing is carried out to instead focus on the stakeholders; the players and the clients, to make sure that all of their requirements are met and that they are happy with the program as a whole. The graphical user interface also requires testing and so that is carried out.

Overall program user testing (Beta testing)

I will gather a group of avid gamers to test the overall functionality of the game, to test that all modules work together as they should and that there are no logical errors or efficiency/robustness issues. Any issues that any of the players come across will be reported to me and written up in a report. This will then be analysed and the required modules will be reviewed, any possible improvements will be made.

Testing requirements

Test no.	Test subject	Justification (testing to ensure what?)	Test data to be used	Expected output
1	GUI	Ensures the game has a functional GUI	- (Playing the game)	All GUI tests (see below) are passed.
2	Environment generator	Ensures that the environment successfully generates every time a new level is started, and ensures that distinct levels never run out.	Enter gameplay via the menu play button (click).	Environment generates successfully, i.e. no rendering/spawning errors with objects interfering with each other and sufficient objects spawned. Also the environment generated must be different every time.
3	Free play mode	Ensures that there is a mode in which the enemy spawn functionality is removed so that the player can practise without enemies.	Enter freeplay through the pause menu.	All gameplay requirements are met, such as environment generation, player modules etc. But, no enemies.

4	Player	Ensures that the player functions as required during gameplay, this includes movement, attacking, etc.	- (Playing the game)	Player functions as required, all modules work allowing the player to move, attack, take damage, etc.
5	UI	Ensures that there is sufficient UI for the player.	- (Playing the game)	All UI tests (see below) are passed.
6	Weapon classes	Ensures that the player can choose and use different weapon classes, in ways that they should be used.	Changing and using weapon class in weapons menu, accessed through pause menu.	Weapon classes should be able to be changed easily and successfully, these changes should carry over to the actual gameplay. All different weapon classes should work as they are described with respective attributes.
7	Enemies	Ensures that the enemies function as required during gameplay, this includes movement, attacking, etc	- (Playing the game)	Enemy functions as required, all modules work allowing the enemy to move, attack, take damage, etc.
8	Menu (Play/Settings/Quit)	Ensures that the menu and respective buttons	Load up game. Press play button. Press settings button. Press Quit button.	Should display all menu options with functional buttons; Play button should enter the player into gameplay. Settings button should take the player to the settings screen. Quit button should close the game application.
8.2	Settings page	Ensures the settings page functions as required, allowing the player to change graphics level, controls and volume.	Press toggle graphics button. Change control binds. Slide vol. slider.	Graphics level should toggle between levels. Controls should change to the button the player presses. Volume should change to the percentage of slider selected.
9	Website	Ensures website works as required.	Use website	Information should be viewable, game should be downloadable, feedback form should accept user inputs.

Graphical user interface testing

Test no.	Test subject	Justification (testing to ensure what?)	Test data to be used	Expected output
1	Main Menu	Ensures that the main menu screen is displayed to the player as designed and required. Ensures that the menu allows inputs, i.e. ensures the player can choose any of the 3 options	Start the program / load the game (Menu is first screen to be displayed) Click the play button Click the settings button Click the exit button	The Main menu screen should be displayed to the player as designed and required. All 3 buttons should work, i.e. allowing inputs. Play button should take player to game. Settings button should take the player to the settings/options page. Exit button should close the application.
2	Settings	Ensures that the settings screen is displayed to the player as designed and required. Ensures that the player can successfully input data in the form of their preferences for; graphics level, controls, volume.	Enter the Settings page via the settings button on the main menu. Click the toggle graphics button. Click the change button for both primary and secondary fire, then enter input of choice. Click / Drag along points on the volume slider.	The Settings screen should be displayed to the player as designed and required. The player should be able to change any of the 3 options to their preference. Graphics toggle button - Graphics levels should switch between each other. Change controls button - Primary/secondary fire controls should change the player's inputs. Volume slider - Volume should change to the percentage of the slider player has clicked on.
3	Pause menu	Ensures that the Pause menu is displayed to the player as designed and required. Ensures the pause menu allows inputs so that the player can select whether to unpause, go to the weapons menu or go to the main menu.	Enter the Pause menu via the pause button during gameplay. Click the unpause button. Click the weapons menu button. Click the back to main menu button.	The pause menu screen should be displayed to the player as designed and required. All 3 buttons should work, i.e. allowing inputs. Unpause button should return the player to gameplay. Weapons menu button should take the player to the weapons menu screen. Return to main menu button should take the player to the main menu.

4	Weapons menu	<p>Ensures that the Weapons menu is displayed to the player as designed and required.</p> <p>Ensures the player can interact with the menu, i.e. allows inputs</p> <p>Ensures the player can select / switch between the weapons using the previous/next weapon buttons, and that back button works as intended.</p>	<p>Enter the Weapons menu via the weapons menu button during gameplay.</p> <p>Click the previous weapon button.</p> <p>Click the next weapon button.</p> <p>Click the back button.</p>	<p>The Weapons menu screen should be displayed to the player as designed and required. All 3 buttons should work, i.e. allowing inputs.</p> <p>The previous/last weapon buttons should switch between the weapons and change the corresponding information such as image, name and attributes.</p> <p>The back button should take the player back to the game in a paused state, i.e. to the pause menu.</p>
5	Website feedback form	<p>Ensures the website feedback form allows the players to enter data and submit through the use of buttons and test inputs.</p>	<p>Go to the website's feedback form page.</p> <p>Enter text data into the text data questions as input.</p> <p>Click the lozenges for multiple choice questions as input.</p> <p>Click the boxes for multi-select questions as input.</p> <p>Click the submit button.</p>	<p>The Website feedback form should be displayed to the player as designed and required.</p> <p>Text data entered should appear in the textbox and be stored when the player clicks submit.</p> <p>Clicked lozenges should display as filled and the corresponding value (option) should be stored when the player clicks submit.</p> <p>Clicked boxes should display as filled and the corresponding value (option) should be stored when the player clicks submit.</p>

Client acceptance testing

To ensure user acceptance, the client will carry out a run through of all the game's features in a logical order as a test to make sure that they are completely happy with all areas of the application / product.

1. Start the program.
2. Through the menu's settings button, go to settings/options page.
3. Change graphics level to whichever you desire using the toggle graphics button.
4. Change the primary and secondary fire controls to whichever you desire, using the change controls button.
5. Change the volume using the volume slider to whichever you desire.
6. Click the back button to go back to the main menu.
7. Click the play button to enter gameplay.
8. Play the game by defeating all enemies to progress to the next level, for at least 3 levels to see how different each level is.
9. Press the pause button to enter the pause menu.
10. Click the unpause button to go back to gameplay.
11. Switch to free play to play with no enemies.
12. Go to the weapons menu via the button on pause menu, to change weapon class using the next/previous buttons.
13. Click the back button to go back to the pause menu.
14. Click the back to main menu button to go back to the main menu.
15. Click the exit button to close the game/application.
16. Go to Website feedback form.
17. Fill out the form and click submit.

After this, the client will answer the following questions which will then be evaluated to make any improvements to the game to increase the user acceptance of the program. The client should also bring up any other questions, suggestions or concerns that they may have about the product which will also be evaluated.

- What did you like and not like about the program?
- What are the key changes you would like to see in the program to better meet the requirements we had set?
- What are the changes you would like to see in the program for overall better user experience.
- What additional features would you like to see in the program? (I.e. features not in the requirements set)
- Are you happy with the program?

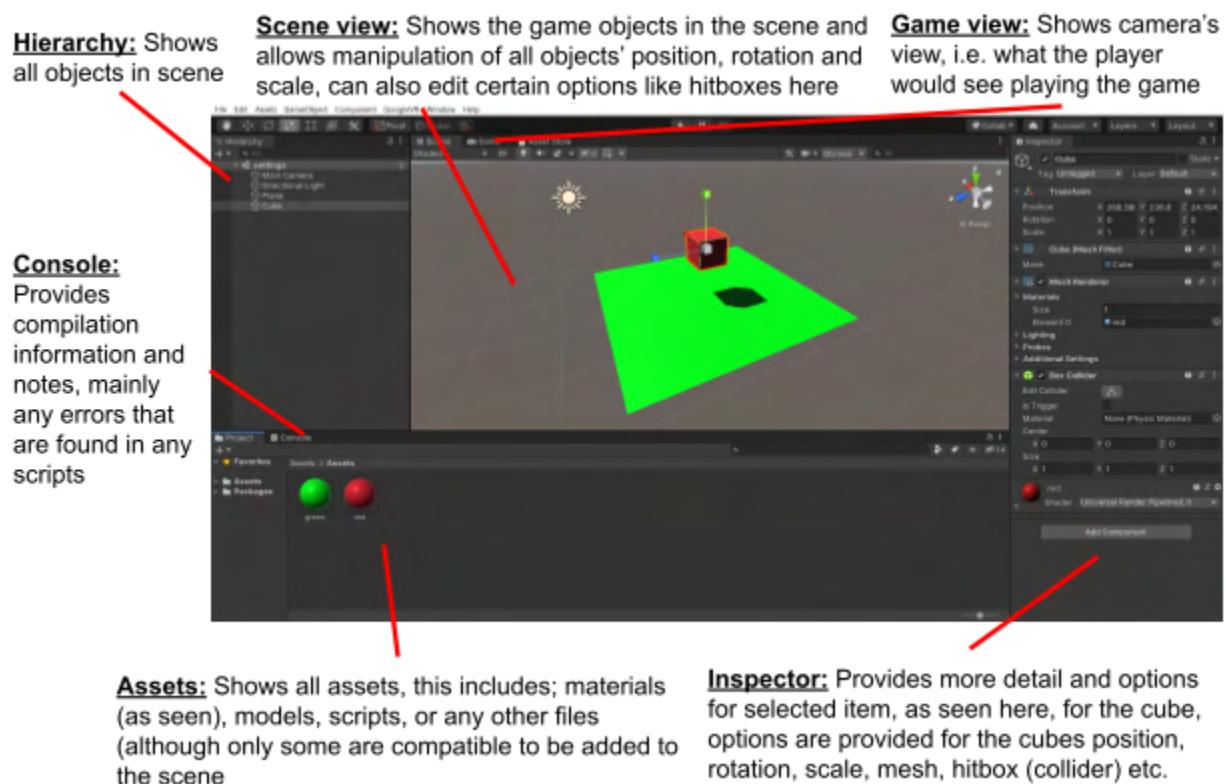
These questions, being quite open and seeking of negative feedback, should allow the client to be as open and upfront as possible so that any important questions or concerns can be raised for evaluation and reiteration of the program.

IMPLEMENTATION

Introduction

The Unity game engine is an object-oriented engine that will be used to develop and produce the product as the client is onboard with the idea of using an engine and in fact, encourages it due to the many benefits. The benefits of using the Unity engine include being able to quickly add and alter 2D and 3D objects within the game, compilation is done by the engine, a high level language (C#) is used and many more etc. Overall, these many benefits enable me to produce a high quality, complex game without having to worry about low level graphics problems which are incurred when making a 3D game from scratch, as these problems are not yet solvable by me due to lack of experience.

Below is a simple introduction into the engine:



These key terms are important as they may be referenced during development.

Development Plan

Iterative Development

My approach to development is an iterative one, for each module I aim to complete a single prototype, before testing it and developing a second prototype if the first one fails any tests. This will be repeated until a prototype is created such that all tests are passed successfully.

Order of module development (in relation to design section)

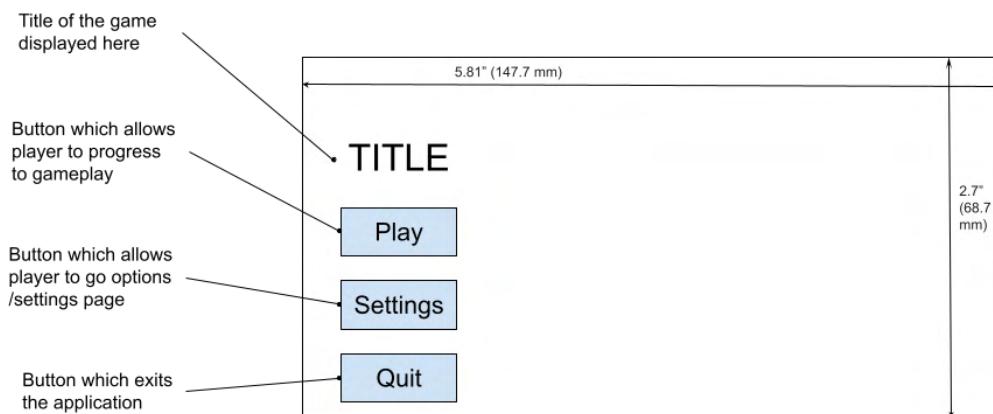
See below for the order in which I plan to develop the solution. Note, the module names used refer to those in the design section problem decomposition.

1. Main Menu
2. Player
3. Environment generation
4. Enemies
5. Weapons
6. UI
7. Website

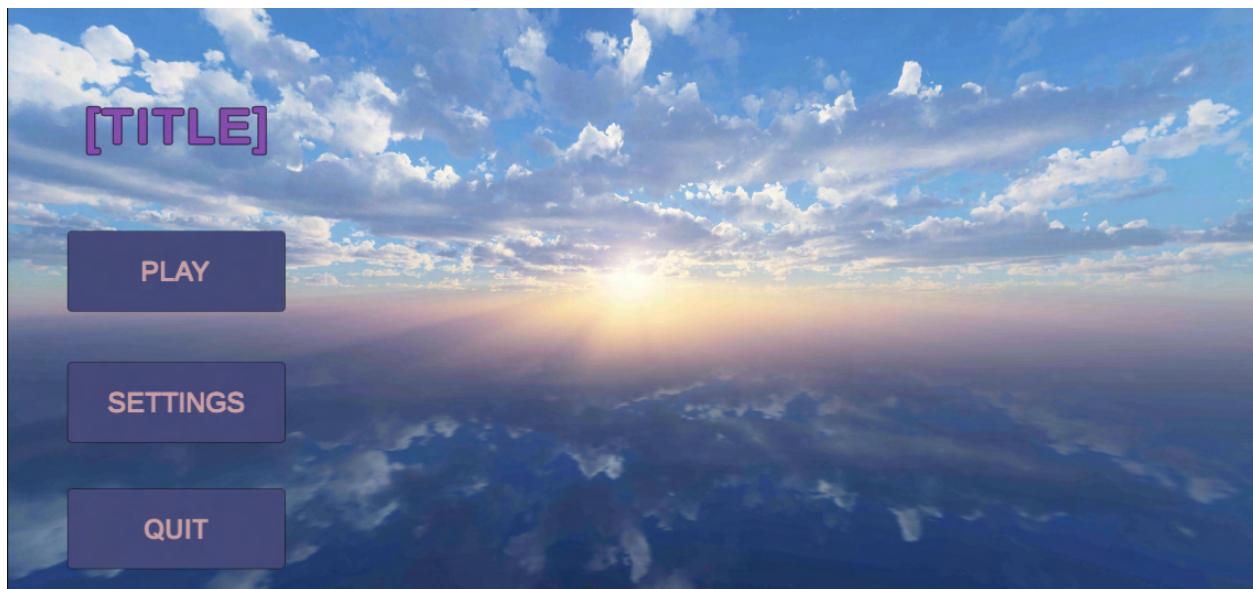
Main menu

This section meets requirement(s): 8.1, 8.2, 8.5 and relates to ‘main menu’ and ‘quit’ from the design section.

For the menu, it is important to reference the GUI designed for this:



I tried to best replicate this design in Unity:



As seen, I added some background scenery to improve user experience and chose a suitable colour palette. (This acknowledges limitations discussed during analysis and aims to reduce them). The title and buttons are positioned just as designed. An important point to note is that Unity takes care of the resolution for all scenes depending on the build settings, where ‘mobile’

has already been selected. Therefore, Unity will automatically size the scene to the right resolution for mobile devices.

To make the buttons work, I developed the following script:

```
public class menuButtons : MonoBehaviour
// inherits MonoBehaviour to use Unity specific methods
{
    4 references
    private string sceneName;
    // scene loaded using this sceneName string,
    // set to different value depending on button pressed

    0 references
    public void playButtonClick()
    // module for when play button is pressed
    {
        sceneName = "game0";
        loadScene(sceneName);
    }
    // sceneName set to required name for gameplay,
    // LoadScene module called using this sceneName

    0 references
    public void settingsButtonClick()
    // module for when settings button is pressed
    {
        sceneName = "settings";
        loadScene(sceneName);
    }
    // sceneName set to required name for settings page,
    // LoadScene module called using this sceneName

    2 references
    public void loadScene(string sceneName)
    // modules used to Load a scene using sceneName
    {
        Scene newScene = SceneManager.GetSceneByName(sceneName);
        SceneManager.LoadScene(sceneName, LoadSceneMode.Single);
        SceneManager.SetActiveScene(newScene);
    }
    // creates a new scene, loads it and sets it as active,
    // required so player can play on the scene

    0 references
    public void quitButtonClick()
    // module for when quit button is pressed
    {
        Debug.Log("Application now exiting");
        Application.Quit();
    }
    // sends a console message and then exits application
}
```

The class involves 4 functions, 1 for each button on the menu and an additional function for loading a scene by scene name. This was done with the aim to reuse code and the same code was being used in 2 of the modules.

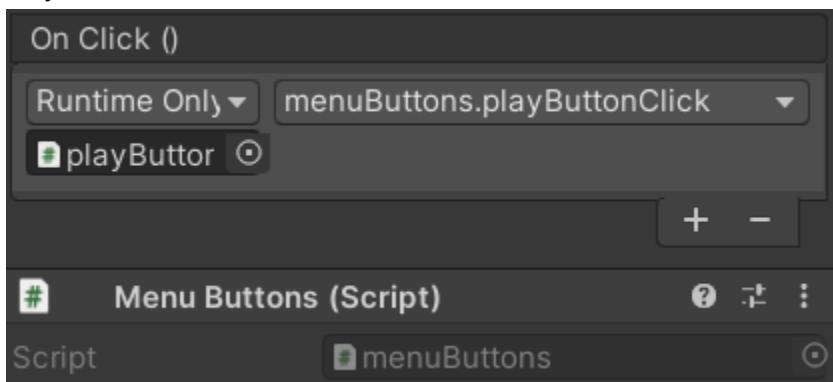
playButtonClick takes you to the play scene.

settingsButtonClick takes you to the settings scene.

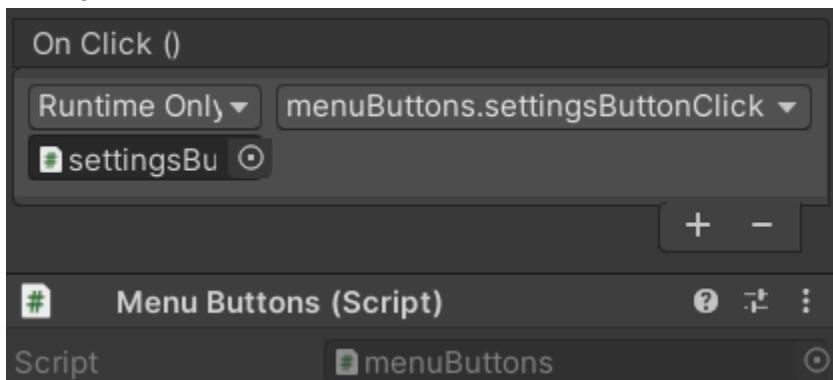
quitButtonClick exits the application.

Then, using Unity's GUI system, I enabled the correct modules to be called when specific buttons are pressed:

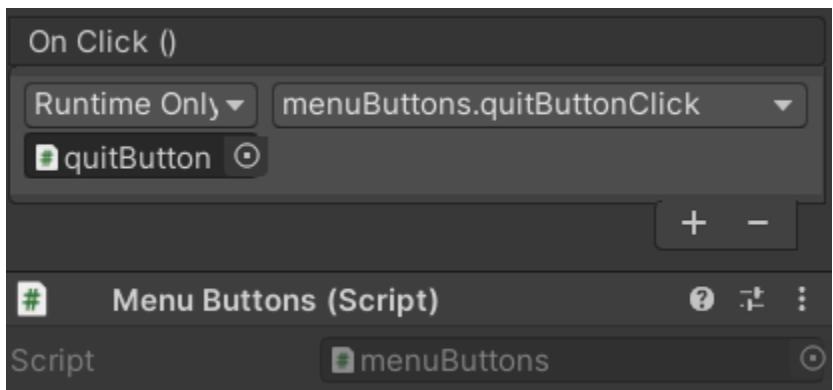
Play button:



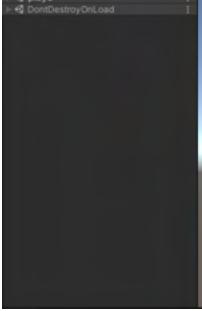
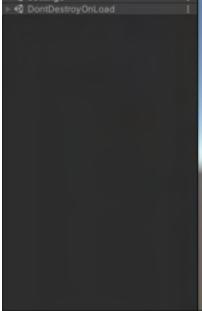
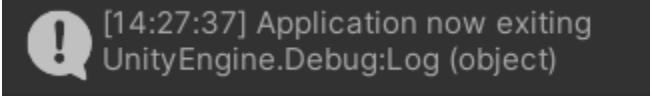
Settings button:



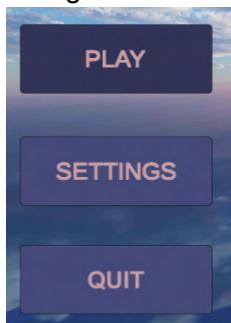
Quit button:



Testing

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/Fail</u>
'Play' button click (acceptance data)	'Play' subroutine is called and runs		Pass
'Settings' button click (acceptance data)	'Settings' subroutine is called and runs		Pass
'Quit' button click (acceptance data)	'Quit' subroutine is called and runs	<p>The application cannot be exited in the Unity editor, however, we can prove that the correct function is called and runs via the console log:</p> 	Pass

After some time I realised that it can be hard to tell which menu button is currently selected when using the main menu. Visibility issues are something that was discussed in the limitations section during analysis, so to further reduce this as a limitation, I decided to make the buttons change to a darker colour when selected to make it clearer. The change is shown here:



Here, play is selected.

Gvr modules

To make a Google VR game some of the relevant Gvr modules need to be used. Most of the time, this includes; GvrEditorEmulator, GvrReticlePointer & GvrEventSystem.

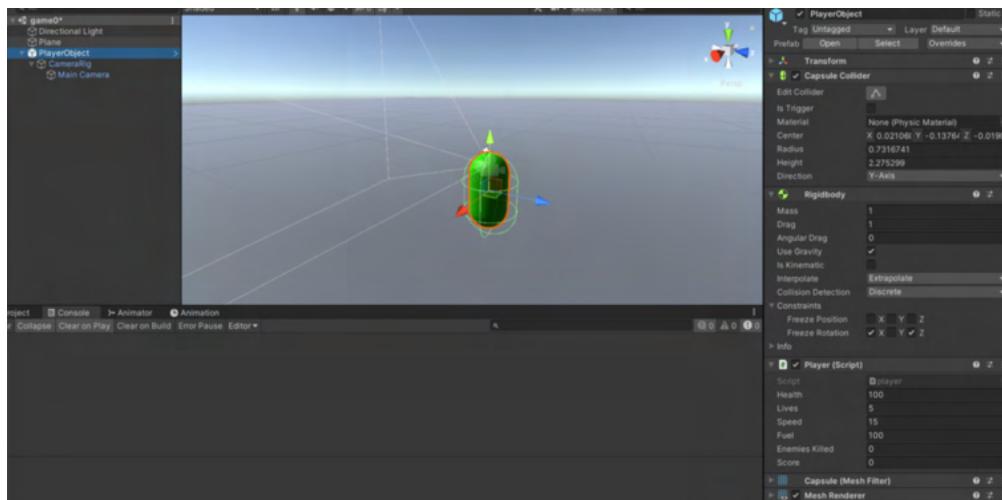
GvrEditorEmulator allows me to play test the game in the Unity editor as if it was being played on a mobile device as a Cardboard VR game. It adds controls such as holding down the 'alt' key allows you to look around using the mouse, etc. GvrReticlePointer is a reticle pointer, i.e. a crosshair which, when paired with the GvrEventSystem, can easily register any events with other game objects. (<https://github.com/googlevr/gvr-unity-sdk>)

For simplicity's sake I will be developing the game as a normal desktop game, then adding these modules at the end, adapting the game into a Gvr game.

Player

This section meets requirement(s): 4.1, 4.3, 4.4 and relates to 'player', 'player movement', 'player jump', 'player attack', 'player damage' and 'player refuel' from the design section.

I started by creating an object to represent the player and decided a simple capsule was best, as this is to be a first-person game, the player won't actually see themselves so a realistic model is not required.



The capsule has a pre-built capsule collider, slightly bigger than the capsule model. This means that this player capsule can stand on the ground without falling through due to the interaction between the 2 colliders of the capsule and that of the floor plane.

I also added a Rigidbody component, and as the name suggests, it turns the player capsule into a rigid body, i.e. allowing it to obey the laws of physics, e.g. falling if not colliding with another rigid body. Then I froze the x and z axis rotation as the player is not supposed to tilt/trip over. Next, I placed the main camera as a child of the capsule, around where the eyes of the player would be, this means that when the player capsule moves, the camera also follows, i.e. the relative positions of the capsule and camera are always constant. Any other objects I give the

player will now be children of the camera, as things such as weapons should rotate as the camera is rotated by the player's view.

Link to design section:



Finally, I added the base player script with the attributes that the player will have, such as health, lives, speed, etc:

```
using UnityEngine;

3 references
public class player : MonoBehaviour
// inherits MonoBehaviour to use Unity specific methods
// in this class and child classes
{
    0 references
    protected int health = 100;
    // player's health attribute, set to 100

    0 references
    protected int lives = 5;
    // player's lives attribute, starts with 5 lives

    6 references
    protected float speed = 15f;
    // player's speed attribute, set to 15

    0 references
    protected Weapon weapon = chosenWeapon;

    0 references
    protected int fuel = 100;
    // player's fuel attribute used by mobility gear

    0 references
    protected int enemiesKilled = 0;
    // number of enemies the player has killed

    1 reference
    protected int score = 0;
    // player's score attribute
}
```

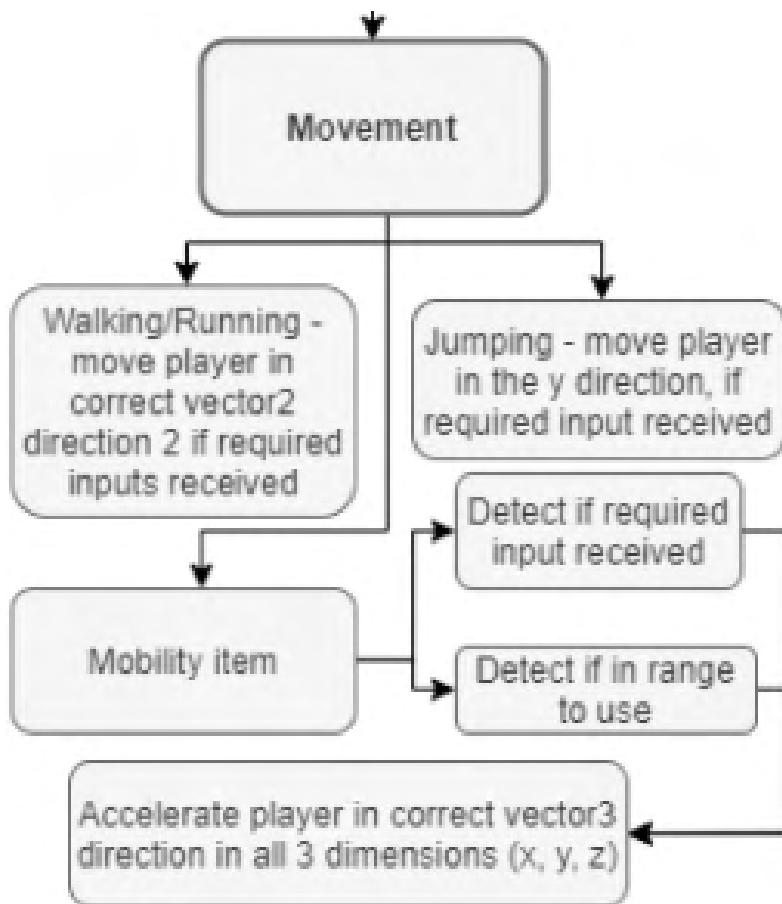
It is important to remember that this class does not have any methods and therefore does not do anything, this is because it is a base class which is to be inherited by other classes which will implement the required functionalities.

As seen the player has 7 attributes including; health - used to allow the player to die, lives - used for gameover, speed - to determine how fast the player should move, weapon - which weapon the player is using, fuel - if or not the player can use mobility gear, enemies killed - count of how many enemies the player has killed and score - a general representation of the player's progress during the game. These are protected variables so that they can be accessed by child classes, but not anything else. Note, the weapon class has not yet been created, hence the error. Note also, there are no tests for this class.

Player Movement

This class relates to the player movement class from the design section.

Link to design section:



As a general description, the player movement system is an automatic movement system in which the player moves in the direction they are facing by a factor of their speed. However, the player may not always want to move, therefore, by looking between certain angles of the horizontal, the player can stop moving. For this to seem smooth, a multiplier is applied slow or speed the player depending on the angle they are looking at.

The initial script was developed as shown below:

```
public class playerMovement : player
{
    // VARIABLES

    // camera's transform to be accessed via editor
    3 references
    public Transform cam;

    // min and max angle in which player can look to move
    1 reference
    private float minAngle = 30.0f;
    1 reference
    private float maxAngle = 330.0f;

    // multiplier used to increase speed depending on angle
    3 references
    private float speedMultiplier;

    // flag to define whether player should move
    3 references
    private bool toMove = false;

    // mouse sensitivity
    2 references
    private float sensitivity = 120f;

    // amount of rotation to be applied around x-axis
    4 references
    private float verticleRotation = 0f;
```

Here, the attributes are defined at the start of the class before any methods are developed as they will be dependent on these attributes.

```
// Start function called once, when program first runs
0 references
void Start()
{
    // makes sure player cursor is locked to center crosshair
    Cursor.lockState = CursorLockMode.Locked;
    // makes sure player cursor is not seen, only crosshair
    Cursor.visible = false;
}
```

```

// FixedUpdate() runs at small, regular intervals, but slower than Update()
0 references
void FixedUpdate()
{
    // move called to move player in direction player Looks
    move();
}
// FixedUpdate() used as movement is automatic, smoother to move at regular intervals,
// move() has no inputs so player won't feel input delay due to less frequent calls

// Update() runs once per frame, therefore dependant on performance
0 references
void Update()
{
    // Look() function called rotation
    look();
}
// Look() involves input, and as Update() is more frequent than FixedUpdate(),
// player wont feel input delay

```

The main game loop is created via the use of Update() and FixedUpdate() as these run repeatedly, forever. Start runs only once, when the program first runs.

Player movement requires the player to move around as well as look around, therefore these run in the game loop.

```

private void look()
{
    // depending on how much the mouse is moved, a value between 1 and -1 is returned
    // these floats can then be applied to rotate the player accordingly
    float mouseX = Input.GetAxis("Mouse X") * sensitivity * Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * sensitivity * Time.deltaTime;

    // player rotated in y-axis depending on how much mouse is moved horizontally
    this.transform.Rotate(Vector3.up * mouseX);

    // use this intermediate variable so that it can be clamped
    // rotation clamped between 90 and -90 so that the player can't look behind
    verticleRotation -= mouseY;
    verticleRotation = Mathf.Clamp(verticleRotation, -90f, 90f);

    // camera rotated by verticleRotation in the x-axis and 0 in the y and z
    cam.localRotation = Quaternion.Euler(verticleRotation, 0f, 0f);
    // camera is rotated instead of player so that entire player body does not
}

```

The look() method takes input from the player's mouse to make them look in that direction by rotating the player body and/or player camera. The use of the verticleRotation variable is essential as it allows me to clamp this value between -90 and 90 degrees. This ensures that the

player can only look in front of themselves and not behind as this is not possible for a normal human.

```
private void move()
{
    // the angle the camera is looking at
    float angle = cam.eulerAngles.x;

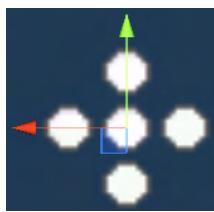
    // if the camera angle is within min and max range,
    // player can move
    if (angle < minAngle || angle > maxAngle)
    {
        toMove = true;
    }
    // otherwise the player should not move
    else
    {
        toMove = false;
    }

    // relevant calculations only carried out when player needs to move
    if (toMove)
    {
        // multiplier set to 1,
        // if player/camera is looking completely horizontally
        if (angle == 0)
        {
            speedMultiplier = 1f;
        }
        // otherwise, multiplier is calculated as reciprocal of the angle
        // this gives a multiplier between 1 and 0 (but never 0)
        else
        {
            speedMultiplier = 1/angle;
        }

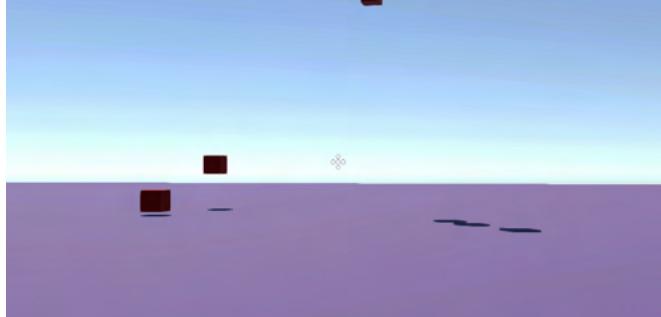
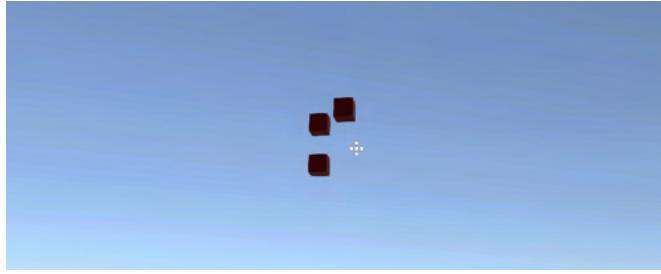
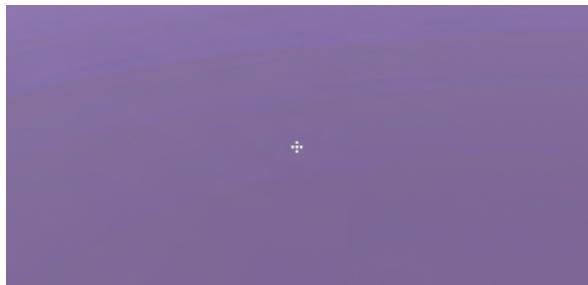
        // multiplier applied to the player's speed
        base.speed *= speedMultiplier;
        // player moved in the direction they are looking by a factor of their speed
        this.transform.position += cam.forward * base.speed * Time.fixedDeltaTime;
    }
}
```

The move() function calculates and applies a multiplier for the player's speed depending on the angle they are looking at. The player is then moved forwards, i.e. the direction they are looking in, by this new, altered speed.

I also added the following cross hair as a child of the camera:



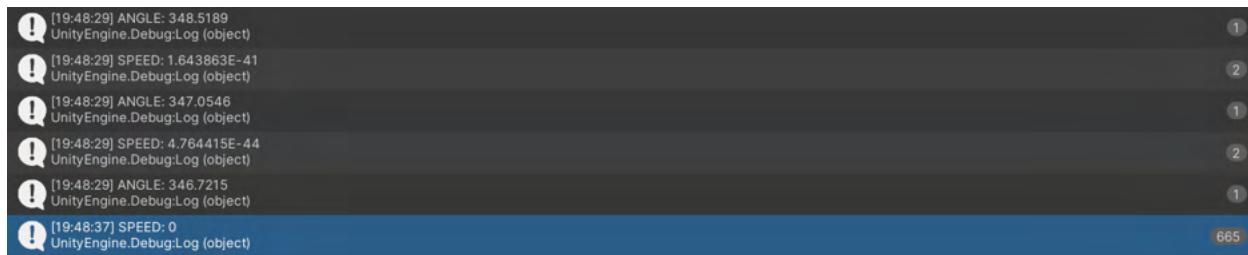
Testing 1

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Looking horizontally (0 degrees) (Normal)	Player moves in the direction they are looking with maximum speed	 Player does not move	Fail
Looking at 15 degrees, above horizontal (Normal)	Player moves with speed 8, in the direction they are looking	 Player does not move	Fail
Looking at 50 degrees, above horizontal (Normal)	Player does not move	 Player does not move	Pass
Looking at 330 degrees, i.e. 30 degrees below horizontal (Boundary)	Player does not move	 Player does not move	Pass

Iteration 2

So overall, the main problem was found to be the fact that the player does not move at all. Using some print / console debug statements in the Update() method, I was able to analyse how the angle and the speed changed during gameplay, as these are the main variables required to move the player.

The following console screenshot shows this:



```
[19:48:29] ANGLE: 348.5189  
UnityEngine.DebugLog (object)  
[19:48:29] SPEED: 1.643863E-41  
UnityEngine.DebugLog (object)  
[19:48:29] ANGLE: 347.0546  
UnityEngine.DebugLog (object)  
[19:48:29] SPEED: 4.764415E-44  
UnityEngine.DebugLog (object)  
[19:48:29] ANGLE: 346.7215  
UnityEngine.DebugLog (object)  
[19:48:37] SPEED: 0  
UnityEngine.DebugLog (object)
```

As seen, the speed quickly becomes extremely small until it becomes 0 and does not change after this. I realised that this is because the speed is affected by the multiplier but does not reset every time, i.e. the multiplier makes the speed smaller and smaller until it becomes 0.

```
// multiplier applied to the player's speed  
base.speed *= speedMultiplier;
```

This line repeats and the speed eventually maxes out at 0. This is because the multiplier is always a value below 1. I considered adding 1 to the multiplier so that the speed never reaches 0, however, this will just shift the problem by making the speed larger and larger, exponentially.

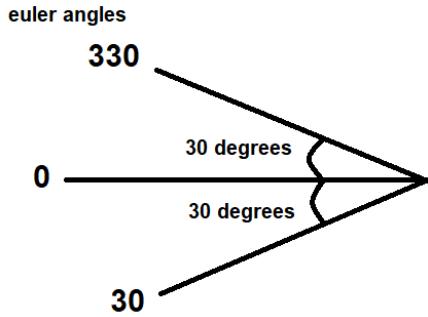
Instead, I introduced a new constant, halfMaxSpeed, defined as a float, for now 7.5. Since the speed changes, we need a constant to refer to. We can apply the multiplier to this.

However, since the multiplier has a domain of $0 < x \leq 1$ and so we can add 1 to change that to $1 < x \leq 2$. Therefore, the maximum multiplier that can be applied is 2 and if this is applied to half the max speed, it will result in the max speed. This creates the function for speed below:

```
// multiplier+1 applied to half the player's max speed  
base.speed = halfMaxSpeed * (speedMultiplier+1);
```

The speed can now only range from between halfMaxSpeed (7.5) and double this, 15.

Another problem I noticed from the console logs was that the camera euler angles were offset differently than I had assumed. Below 0, the angles started to increase and above the horizontal they decreased from 360. In other words, they work as shown in the diagram below:



This meant that the inverse function to calculate the multiplier does not work as intended, I wanted the multiplier to be 1 at 0 degrees and then reduce equally, regardless of whether the player looks up or down. However, this is not the case, for example, the multiplier should be the same when the player looks at 20 degrees and 340 degrees, but $1/20$ and $1/340$ give different values.

Therefore, I introduced a new method that takes the camera euler angle and returns the acute version of that angle:

```
private float calculateAcuteAngle(float cameraAngle)
{
    if (cameraAngle > maxAngle)
    {
        angle = 360f - cameraAngle;
    }
    // this is the range in which angle is between 360 and 330
    // changes it to a value between 0 and 30
    else if (cameraAngle < minAngle)
    {
        angle = cameraAngle;
    }
    // Will return a number between (0 - 30.0)

    return angle;
}
```

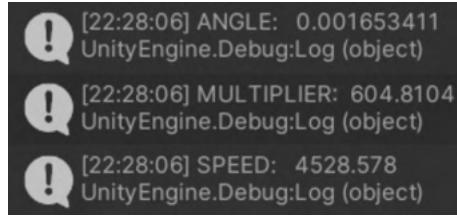
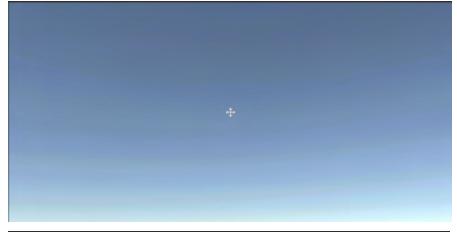
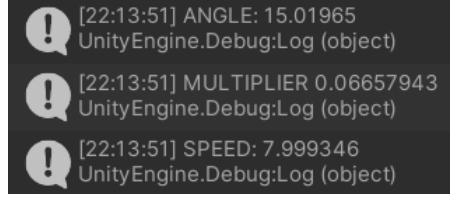
If required, i.e. when looking above the horizontal, the method converts the angle to a value between 0 and 30, to match the values below the horizontal 0.

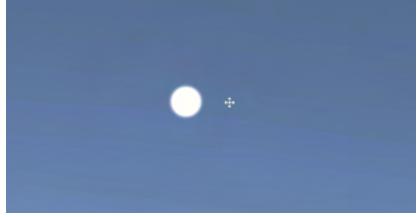
This function is called when the player is to move as the multiplier is only calculated in this case.

```
// relevant calculations only carried out when player needs to move
if (toMove)
{
    angle = calculateAcuteAngle(angle);
```

I also broke up other parts of the move() function into smaller sub functions such as moveCheck() and calculateMultiplier(). These can be seen in the appendix.

Testing 2

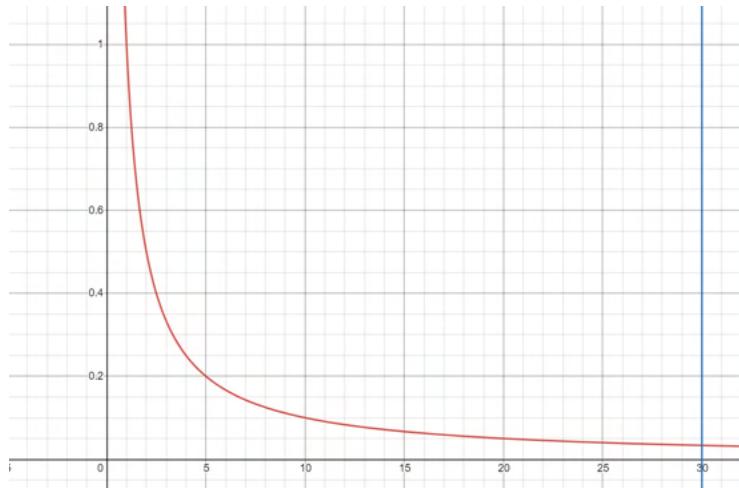
<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Looking horizontally (0 degrees) (Normal)	Player moves in the direction they are looking with maximum speed	 	Fail
Looking at 15 degrees, above horizontal (Normal)	Player moves with speed 8, in the direction they are looking	 	Pass

Looking at 50 degrees, above horizontal (Normal)	Player does not move		Pass
Looking at 330 degrees, i.e. 30 degrees below horizontal (Boundary)	Player does not move		Pass

Iteration 3

Although at exactly 0 degrees (horizontal) the multiplier is 1 as required due to the if statement that calculates the multiplier exclusively for an angle of 0 (to avoid an undefined math error). However, even for angles between 0 and 1, the multiplier is calculated using $1/\text{angle}$. This is a problem as it results in extremely large multipliers and therefore extremely large speeds.

I also realised that the multiplier calculation should be done differently.



As seen, the exponential function only really provides a high speed for angles 0-5 degrees, then provides a relatively quite low multiplier for 5-30, i.e. it is not spread out well. And of course, the multiplier needs to max out at 1 and not go beyond this.

To make 1 the maximum multiplier we can add 1 to the angle (x) which translates the function 1 to the left:



However, this is still not ideal as from 10-30 degrees the multiplier is consistently low. The only way to change this is to implement a new variable k , with its own function, into the multiplier function. I.e. $y = k / (x+1)$

In essence, this can be solved by trying to create a function which matches ideal results consistently across the range. I decided that;

1. at $x = 0$, y must = 1 (as discussed before).
2. at $x = 15$, y must = 0.5 (about half way)
3. at $x = 30$, y must = 0.1.

Using the function $y = k / (x+1)$, to satisfy the first condition, when $x = 0$, for $y = 1$, k must = 1.

For the second condition, when $x = 15$, for $y = 0.5$, k must = 8.

For the third condition, when $x = 30$, for $y = 0.1$, k must = 3.1.

The function of k must satisfy the points; $(0, 1)$, $(15, 8)$, $(30, 3.1)$. Therefore, this will be a quadratic (parabola) function, which can be worked out using these points:

k is a parabola
 $\therefore k = ax^2 + bx + c$

$(0, 1)$
 $(15, 8)$
 $(30, 3.1)$

$l = a(0)^2 + b(0) + c$
 $\therefore c = 1$

$8 = (15)^2 a + 15b + 1$
 $3.1 = (30)^2 a + 30b + 1$

$225a + 15b = 7 \quad (1)$
 $900a + 30b = 2.1 \quad (2)$

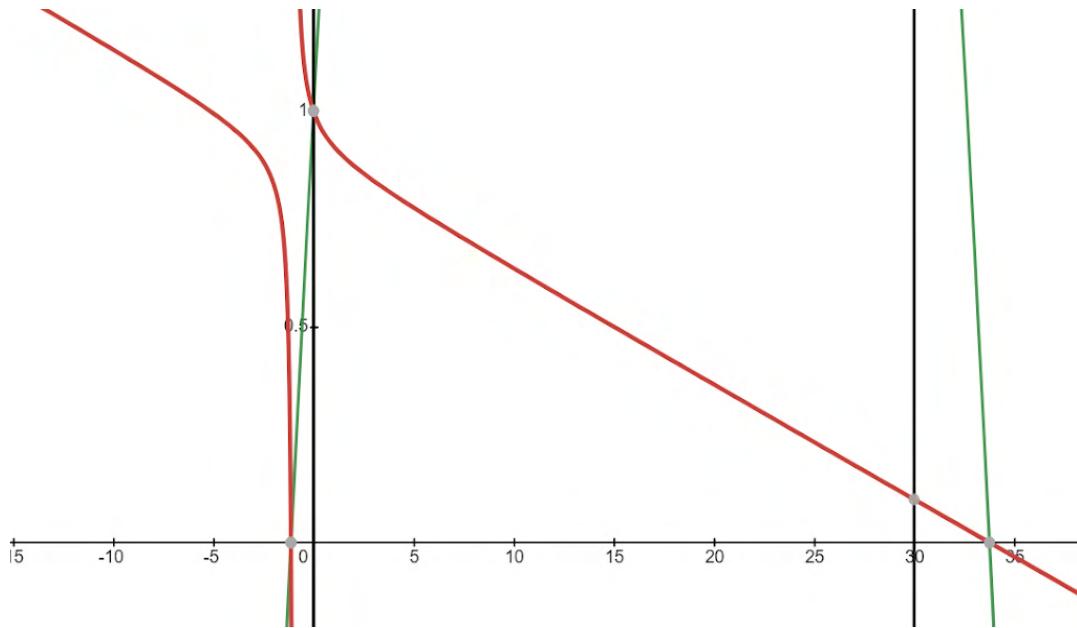
$450a = -11.9$
 $\therefore a = -\frac{11.9}{4500}$

$\text{SUB } a = -\frac{11.9}{4500} \text{ in } (1)$
 $225\left(-\frac{11.9}{4500}\right) + 15b = 7$
 $15b = \frac{259}{20}$
 $\therefore b = \frac{259}{300}$

$\therefore k = -\left(\frac{11.9}{4500}\right)x^2 + \left(\frac{259}{300}\right)x + 1$

Therefore, $k = -(119/4500)x^2 + (259/300)x + 1$

As seen, this provides a much smoother curve (red):



The green represents the function for k and the black vertical lines represent the boundaries 0 and 30.

This was implemented in the player movement, calculateMultiplier() function as the following:

```
float k = ( (-0.026444f)*(Mathf.Pow(angle, 2f)) + (0.86333f)*angle + (1f) );
speedMultiplier = (k/(angle+1f));
```

Values are rounded as this will improve performance and have no noticeable gameplay effects. The float, k is also declared only in this separate calculateMultiplier() function which is useful in terms of space complexity as it means that the complex variable k is only stored while the function runs.

Testing 3

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Looking horizontally (0 degrees) (Normal)	Player moves in the direction they are looking with maximum speed		Pass

Looking at 15 degrees, above horizontal (Normal)	Player moves with speed 8, in the direction they are looking	 	Pass
Looking at 50 degrees, above horizontal (Normal)	Player does not move	 <p>Player does not move</p>	Pass
Looking at 330 degrees, i.e. 30 degrees below horizontal (Boundary)	Player does not move	 <p>Player does not move</p>	Pass

Player Jump

This class relates to the ‘player jump’ class from the design section.

The player jump system is another automatic system in which the player has to quickly look upwards to jump. This can be determined by looking at the difference between the angle of the camera before and after a short time wait. If between this time wait, the player has looked up quickly, the difference will be large and the player can be allowed to jump.

I initially developed the following script:

```
public class playerJump : player
// inherites player class to access attributes such as speed
{
    // VARIABLES

    // refrence to the camera
    3 references
    public Transform cam;

    // refrence to the player's rigid body component
    2 references
    private Rigidbody playerBody;

    // bool for either jumping or not jumping
    4 references
    private bool jump = false;

    0 references
    void Start()
    {
        // player rigid body component accessed
        playerBody = GetComponent<Rigidbody>();
        // coroutine for jumping started
        // coroutine used as time waits are required
        StartCoroutine(Jump());
    }
}
```

```

private IEnumerator Jump()
{
    // while true and wait for fixed update mimic the fixed update method
    // this means that this loop will run at consistent intervals
    // roughly once per frame
    while (true)
    {
        // if not jumping, checks carried out to check for jump
        if (!jump)
        {
            // angle sample taken before half a second wait
            float startAngle = calculateLinearAngle(cam.eulerAngles.x);
            // program waits for 0.5 seconds
            yield return new WaitForSeconds(0.5f);
            // angle sample taken before half a second wait
            float endAngle = calculateLinearAngle(cam.eulerAngles.x);

            // difference calculated between 2 angle samples
            float angleChange = startAngle - endAngle;

            // if this difference is greater than 20 (degrees)
            // then jump is set to true
            if (angleChange >= 20f)
            {
                jump = true;
            }
        }

        // if jump is true, the player is to jump
        if (jump)
        {
            // jump set to false so that the player only jumps once,
            // until jump criteria are met once again
            jump = false;

            // velocity and position are altered to mimic the art of jumping
            playerBody.velocity += Vector3.up * 10;
            transform.position += cam.forward * -2f * base.speed * Time.fixedDeltaTime;

            // program waits for 10 seconds
            // so that the player can fall back to the ground again before another jump
            yield return new WaitForSeconds(10f);
        }

        // waits for next frame before looping
        yield return new WaitForSeconds();
    }
}

```

```

private float calculateLinearAngle(float cameraAngle)
{
    // if the camera is looking below the horizontal,
    // the following is applied to alter the angle
    if (cameraAngle >= 0 && cameraAngle <= 150f)
    {
        // 30 degrees is added to the camera angle,
        // this means it is now between 30 and 60
        cameraAngle += 30f;
    }

    // if the camera is looking above the horizontal,
    // the following is applied to alter the angle
    else if (cameraAngle > 180f)
    {
        // 330 degreed are taken away from the angle
        // this means that it is now between 0 and 30
        cameraAngle -= 330f;
    }

    // overall, this function returns a value between 0 and 60
    // this is a linear scale so the difference can be calculated
    return cameraAngle;
}

```

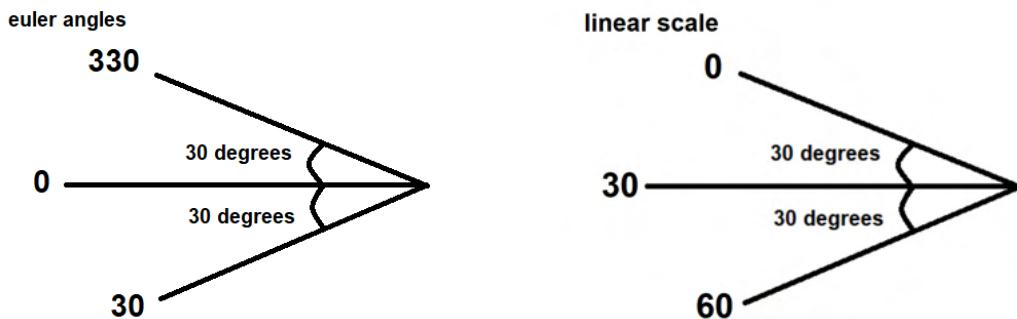
This class uses a coroutine which is a function which allows me to implement time waits. This is essential in calculating if the player has quickly looked upwards. I can take the current camera x angle and then repeat this after a short time wait of 0.5 seconds. Then the difference between these can be found, and if above 20 degrees (which seemed to be reasonable) then the player can jump. However, as we found out with the player movement, the euler angles of the camera increase as you go anti-clockwise. This was a problem as you could be looking 20 degrees below the horizontal and then look 20 degrees above, which is a change of 40 degrees, valid for a jump. However, with euler angles this calculation wouldn't work as the 2 values would be 20 and 340. Therefore, I implemented the calculateLinearAngle() function to convert the angle into a value within the linear range 0-60. This then allowed me to easily calculate the angle change by finding the difference between the 2 angles.

If the criteria for jumping was met (as discussed above) then the player's velocity and position was simply adjusted to mimic the action of a jump. I then added another time wait of 10 seconds so that the player would already have fallen down to the floor before being able to jump again.

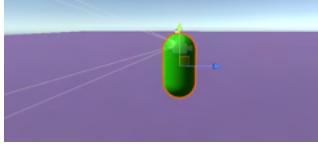
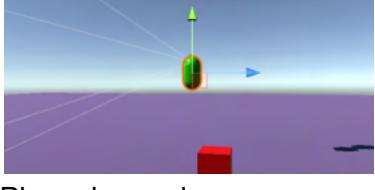
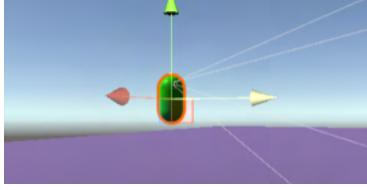
This whole feature was implemented in a while loop which included a `waitForFixedUpdate()`. This meant that the while loop would essentially mimic the `FixedUpdate()` method, by running roughly once per frame, but at regular intervals, therefore not performance dependent.

The diagrams below show the euler angles and the linear scale they are converted to:

0 **and** below has 30 added to it to get a value between 30-60 (or greater), above 0, 330 is taken away to get a value between 0-30 (or less) (euler angle increases from 330 to horizontal).



Testing 1

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Quickly change looking angle by a few degrees (5-10 degrees). (Normal)	Player does not jump (no upwards movement)	 Player did not jump	Pass
Quickly change looking angle by 35 degrees. (Normal)	Player jumps (moves upwards)	 Player jumped	Pass
Quickly change looking angle by 20 degrees. (Boundary)	Player jumps (moves upwards)	 Player jumped	Pass

Quickly change looking angle by over 25 degrees, during a jump, i.e. while the player is mid-air. (Erroneous)	Player does not jump (no upwards movement)		Pass
		Player did not jump again (only once)	

Iteration 2

Although all tests were passed, I was not happy with how the jump would reset. Currently, after jumping the program does not allow the player to jump again for 10 seconds. This ensures that when the player tries to jump again, they are already grounded. However, the player may have fallen back to the ground again before 10 seconds, and via testing I found this was often the case.

I could shorten the time to less than 10 seconds, however, the time to fall down will always be different due to, depending on the varying player speed. Using a time wait isn't a very good simulation of real life jumping and therefore takes away from the immersive experience of the game.

Instead, I used a ray cast to check if the player was grounded and developed the following:

```

// ray shot downwards to check how close ground is
Ray ray = new Ray(transform.position, -transform.up);
RaycastHit hitInfo;

// if the length to the intersection point of the ray on the ground
// is less than or equal to 3 units,
// the player can be considered as on the ground
if (Physics.Raycast(ray, out hitInfo, 3))
{
    grounded = true;
}
// otherwise, player can't be considered as on the ground
else
{
    grounded = false;
}

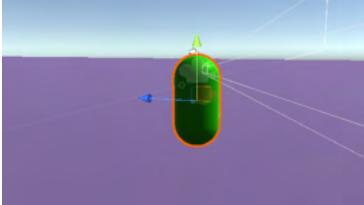
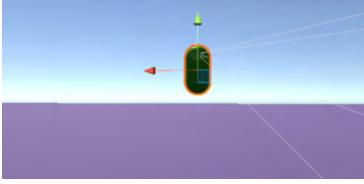
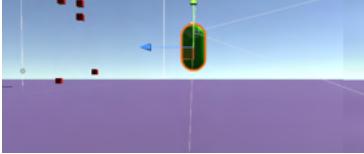
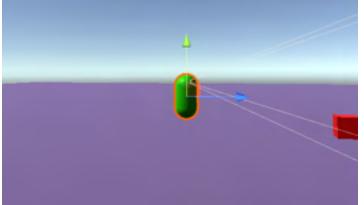
// if this difference is greater than 20 (degrees),
// and player is on the ground,
// then jump is set to true
if (endAngle <= 0 && angleChange >= 20f && grounded)
{
    jump = true;
}

```

A new variable, the bool `grounded`, was introduced. This checks whether the player is close enough to the floor to be considered as grounded, allowing the player to jump. A ray is shot downwards, and if the point at which the ray hits the floor is less than or equal to 3 units away from the player, then `grounded` is set to true. Then, when checking for a jump, the `grounded` variable must also be true for the statement to be true and therefore, for the player to jump. Other than that, the jump condition also included a new condition in which the end angle must be less than 0, this essentially means the player must be looking above 30 degrees above the horizontal for the player to jump. This was added after a bug was discovered that the player could jump by looking quickly downwards as well, which is not how the feature was intended. Also, the time wait after jumping was removed.

I repeated the tests to make sure that the additions/changes still allow the jump class to function as intended.

Testing 2

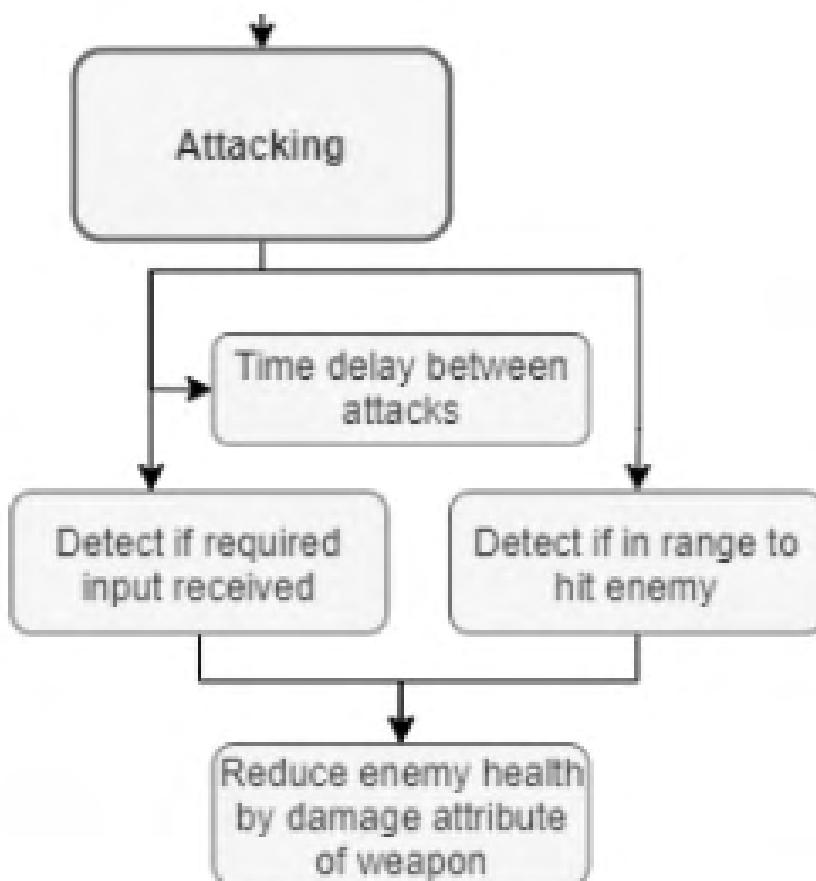
<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Quickly change looking angle by a few degrees (5-10 degrees). (Normal)	Player does not jump (no upwards movement)	 Player did not jump	Pass
Quickly change looking angle by 35 degrees. (Normal)	Player jumps (moves upwards)	 Player jumped	Pass
Quickly change looking angle by 20 degrees. (Boundary)	Player jumps (moves upwards)	 Player jumped	Pass
Quickly change looking angle by over 25 degrees, during a jump, i.e. while the player is mid-air. (Erroneous)	Player does not jump (no upwards movement)	 Player did not jump again (only once)	Pass

As seen, all tests were still passed.

Player Attack

This class relates to the 'player attack' class from the design section.

Link to design section:



The player attack class consists of 2 main functionalities, this includes the player being able to attack enemies using their chosen weapon as well as the player being able to use the grapple hook.

I developed the following script to begin with:

```
public class playerAttack : player
{
    // VARIABLES

    // refrence to the camera
    3 references
    public Transform cam;
    // refrence to the enemy mask
    1 reference
    public LayerMask enemyMask;

    // flag for grappling
    4 references
    private bool canGrapple = false;

    // refrence to player's line renderer component
    5 references
    private LineRenderer lr;
    // refrence to spring joint component
    10 references
    private SpringJoint joint;

    // vector 3 coordinates for grapple point
    3 references
    private Vector3 grapplePoint;
    // float value for distance to grapple point
    5 references
    private float grappleDistance;

    // min grapple distance
    1 reference
    private float minGrapple = 8f;
    // max grapple distance
    1 reference
    private float maxGrapple = 45f;
```

```

void Start()
{
    // accesses the line render component of the player
    lr = GetComponent<LineRenderer>();
}

0 references
void Update()
{
    // if the player left clicks, they should attack/fire
    if (Input.GetMouseButtonDown(0))
    {
        hit();
    }
    // if the player holds left click, should should grapple
    if (Input.GetMouseButton(0))
    {
        Grapple();
    }
    // if player lifts left click, i.e. stops holding
    // they should stop grappling
    else if (Input.GetMouseUp(0))
    {
        StopGrapple();
    }
}

```

```

private void hit()
{
    // shoot a ray from player camera, forwards
    Ray hitRay = new Ray (cam.position, transform.forward);
    RaycastHit attackPoint;

    // calls the fire method of weapon,
    // to play animations or cast bullets
    base.weapon.fire();

    // if the ray hits an object with the enemy mask,
    // and the distance to this object is within the weapon range,
    // the hit should land and player should inflict damage on enemy
    if (Physics.Raycast(hitRay, out attackPoint, base.weapon.range, enemyMask))
    {
        // accesses the base enemy script
        Enemy enemyHit = attackPoint.collider.GetComponent<Enemy>();
        // calls the damage method of the enemy,
        // and uses the weapons damage attribute to reduce enemy health
        enemyHit.Damage(base.weapon.damage);
        // player gains 5 score for a hit
        base.addScore(5);
    }
}

private void Grapple()
{
    // if currently cant grapple, should check for grapple
    if (!canGrapple)
    {
        // shoot a ray from player camera, forwards
        Ray grappleRay = new Ray (cam.position, cam.forward);
        RaycastHit hitPoint;

        // if the ray hits an object, then we can check further to grapple
        if (Physics.Raycast(grappleRay, out hitPoint) && hitPoint.point != null)
        {
            grapplePoint = hitPoint.point;
            grappleDistance = hitPoint.distance;

            // if the distance to where we are grappling is between the limits,
            // then we can grapple
            if (grappleDistance > minGrapple && grappleDistance < maxGrapple)
            {
                canGrapple = true;
            }
        }
    }
}

```

```

// if we are to grapple,
if (canGrapple)
{
    // adds a spring joint to the player gameobject
    joint = this.gameObject.AddComponent<SpringJoint>();

    // attaches the spring joint to the grapple point
    joint.autoConfigureConnectedAnchor = false;
    joint.connectedAnchor = grapplePoint;

    // configures joint to Liking
    // i.e. responsible for bounciness, length, etc.
    joint.maxDistance = grappleDistance * 0.3f;
    joint.minDistance = grappleDistance * 0.1f;
    joint.spring = 6f;
    joint.damper = 2f;
    joint.massScale = 4.5f;

    // sets the line render to have 2 points
    lr.positionCount = 2;

    // only render grapple line if a joint exists
    if (joint)
    {
        // sets the first point of the line to the player's position
        lr.SetPosition(0, this.transform.position);
        // sets the second point of the line to the grapple point
        lr.SetPosition(1, grapplePoint);
    }

    // decreases fuel by 0.2 for every frame grappled
    base.fuel -= 0.02f;
}
}

private void StopGrapple()
{
    // player should not grapple any longer
    canGrapple = false;
    // line should not be rendered any longer
    // so line should have no points
    lr.positionCount = 0;
    // joint should no longer exist
    Destroy(joint);
}

```

This class has 3 main functions each called in the update method:

- hit() - called when the player left clicks (or presses on screen)
- Grapple() - called when the player left clicks (or presses on screen)
- StopGrapple() - called when the player lets go of left click (or screen)

The hit function uses a ray cast to try and detect hits landed on the enemy. This is done by

taking the first gameobject that the ray collides with and using a mask to check if this is an enemy. The ray also has a length restriction applied to it which is equal to the range of the weapon the player is currently using. If these requirements are met, then the respective enemy gameobject is accessed so that the base enemy class' take damage method can be called to damage the enemy. Score is also added to the player's current score. Note that the weapon fire method is called regardless of if an attack hits an enemy. This method fires the weapon, i.e. spawns any bullets if required as well as playing relevant animations.

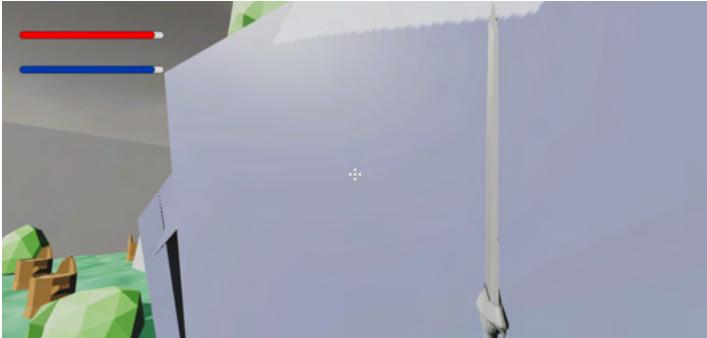
The grapple function runs every frame that the player has pressed left click and essentially does 2 things; if not grappling, checks grapple conditions, but if we are to grapple, then applies forces and physics to enable grappling.

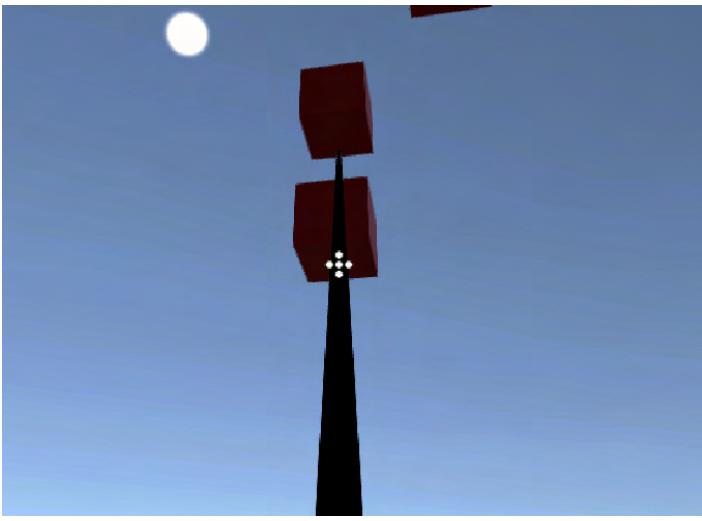
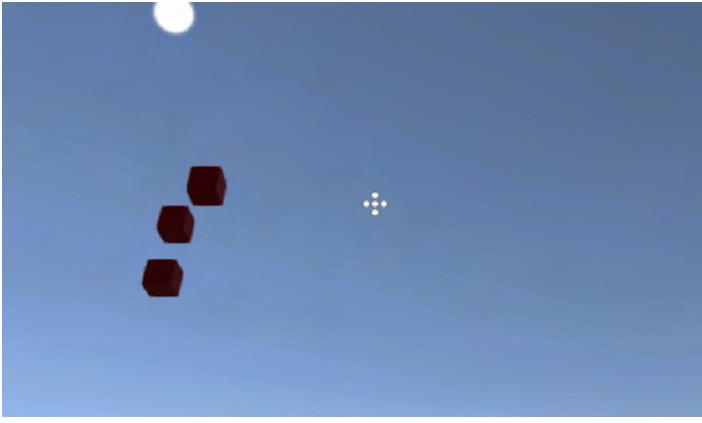
To check if we can grapple, a ray is cast out forwards, to the centre of the screen in the direction the player is looking. I.e. to the player's crosshair. The player cannot grapple onto nothing, therefore we check if the player has actually hit a gameobject, with the ray. If the ray has indeed hit an object to grapple, the distance between the player and this point they want to grapple to is checked to ensure that the player is grappling to an allowable distance. Currently, minGrapple is set to 8 units and maxGrapple to 45. If this final condition is met, canGrapple is set to true to allow the player to grapple.

Once we know we can grapple, relevant physics is applied to simulate the use of a grapple hook. This is mainly done using a spring joint which connects from the player to the point they are trying to grapple to. This point is set to the anchor which means that the grapple line can rotate around this point but remains attached to it. The joint is further configured to simulate a grapple hook by adjusting the spring, damper etc. settings to make the joint seem more springy or stiff etc. Although the spring joint simulates all the physics of the grapple hook, there needs to be a grapple line and this is done by the line renderer component. To draw a line from the player to the grapple point we set the line renderer position count to 2 points and set these points as the player's position and the grapple point position. Finally, fuel is also deducted for every frame grappled and currently this is set to 0.02.

When the player lets go of the mouse, this indicates that they wish to stop grappling. This requires any spring joints to be destroyed so that the player can move by walking as normal, and the line renderer position count must also be set to 0 so that no line is drawn.

Testing 1

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Press button, attack enemy a distance greater than the weapon range (Normal)	The enemy is not damaged.		Pass
Press button, attack enemy a distance less than the weapon range (Normal)	The enemy is damaged.		Pass
Press button, attack enemy a distance equal to the weapon range (Boundary)	The enemy is not damaged.	 (this test was very difficult to do accurately)	Pass

Hold down button, use grapple hook towards any enemy or environment object, at a distance greater than 45. E.g. 50 (Normal)	Grapple hook/line does not render and the player does not move towards grapple point.		Pass
Hold down button, use grapple hook towards any enemy or environment object, at a distance less than 45 E.g 20 (Normal)	Grapple hook/line renders from player to grapple point. Player moves towards this point (until they are out of boundary)	 <p>Player did move towards the grapple point, the movement was a little bit unpredictable, and the line did render but the render seemed a bit offset.</p>	Fail
Hold down button, use grapple hook towards an object which is not an enemy or environment object (Erroneous)	Grapple hook/line does not render and the player does not move towards grapple point.		Pass

Note, some tests were conducted at a later point as some are dependent on other classes being developed first.

Iteration 2

The main 3 problems I found from the testing was:

- the grapple line rendering a bit offset to the player and lagging
- the movement while grappling was not as intended
- the player would not fall back down after grappling

The solution to the grapple line turned out to be that the line should really be rendered after the calculations of where we are grappling to are carried out, instead of the same time, as this causes the line to essentially delay/lag behind our movement. To fix this, the line rendering code can be moved to the LateUpdate() method, which runs after the Update() method has and should solve our problem.

For the movement in the air while grappling, I realised that the player movement script was applied even while in the air grappling, as nothing was in place to pause the normal movement. We do not need automatic forwards movement, as the movement while grappling is taken care of the physics attributes of the spring joint as well as the damping settings.

To enable the player to fall back down again, the spring joint is destroyed when the player stops grappling, this should mean the player is no longer connected to the grapple point so that they fall. I checked the StopGrapple() method however, this was running as intended. The problem seemed to be that more than 1 spring joints were being created and then later only 1 destroyed, meaning that the player would remain in the state of grappling even though no grapple line was drawn on the screen.

To fix the grapple line not being in sync, I added the following code:

```
void LateUpdate()
// render the grapple line in LateUpdate() as this runs after update
// this fixes the problem of the the line not being in sync
{
    // only render grapple line if a joint exists
    if (joint)
    {
        // sets the first point of the line to the player's position
        lr.SetPosition(0, this.transform.position);
        // sets the second point of the line to the grapple point
        lr.SetPosition(1, grapplePoint);
    }
}
```

The line rendering code is now just run in the LateUpdate() instead of within the grapple function. This will ensure that the grapple line only draws after the grapple point is decided and so the line should render in sync with the player and camera.

Although, this is not completely ideal as before, this code was running only when we wanted to grapple, but now if (joint) is checked every frame. However, just 1 if statement should not cause any performance issues at all

To solve the movement issue, I first reference the player movement script and access it in the Start() method:

```
private playerMovement movement;  
movement = GetComponent<playerMovement>();
```

This will be used to disable the toMove bool when grappling, but then re-enable it when we stop grappling. So I created 2 methods in the playerMovement class to allow this:

```
1 reference  
public void disableAutoMove()  
// disables the toMove bool  
{  
|   toMove = false;  
}  
  
1 reference  
public void enableAutoMove()  
// enables the toMove bool  
{  
|   toMove = true;  
}
```

I called disableAutoMove() when we should grapple. Similarly, the enableAutoMove() method is called when we stop grappling. i.e. These are called when the canGrapple bool is altered.

```
movement.disableAutoMove();  
// the automatic movement is disabled  
// since the physics of the spring joint takes care of movement  
  
// player should move automatically again  
movement.enableAutoMove();
```

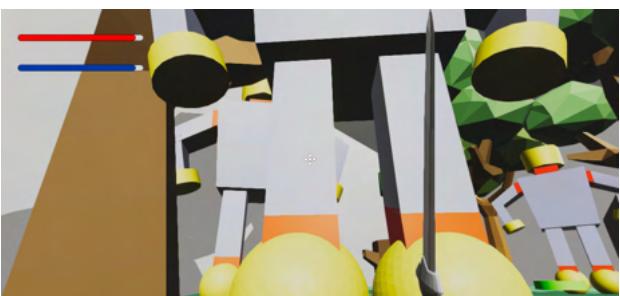
And finally, to fix the error of multiple spring joints being created for the player, I added an if statement to check if a spring joint already exists, before creating one:

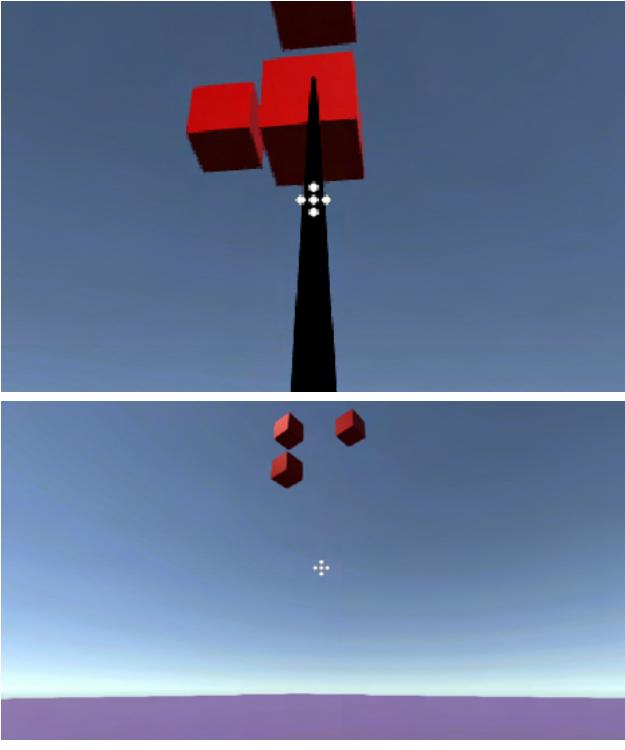
```

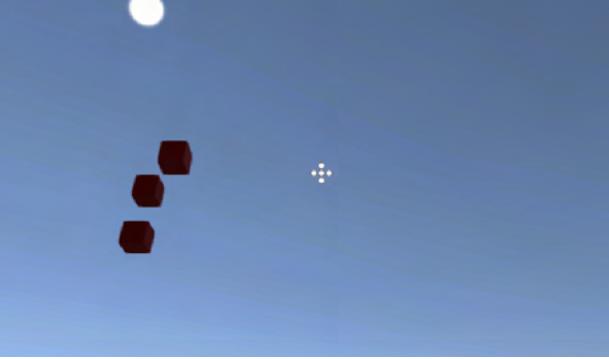
// adds a spring joint to the player gameobject
// only if there isn't one already
if (!joint)
{
    joint = this.gameObject.AddComponent<SpringJoint>();
}

```

Testing 2

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Press button, attack enemy a distance greater than the weapon range (Normal)	The enemy is not damaged.		Pass
Press button, attack enemy a distance less than the weapon range (Normal)	The enemy is damaged.		Pass
Press button, attack enemy a distance equal to the weapon range (Boundary)	The enemy is not damaged.	 (this test was very difficult to do accurately)	Pass

<p>Hold down button, use grapple hook towards any enemy or environment object, at a distance greater than 45. E.g. 50 (Normal)</p>	<p>Grapple hook/line does not render and the player does not move towards grapple point.</p>		<p>Pass</p>
<p>Hold down button, use grapple hook towards any enemy or environment object, at a distance less than 45 E.g 20 (Normal)</p>	<p>Grapple hook/line renders from player to grapple point. Player moves towards this point (until they are out of boundary)</p>	 <p>Player did now move correctly towards the grapple point, and the grapple line rendered in sync. Also, after I stopped grappling, the player did fall back down</p>	<p>Pass</p>

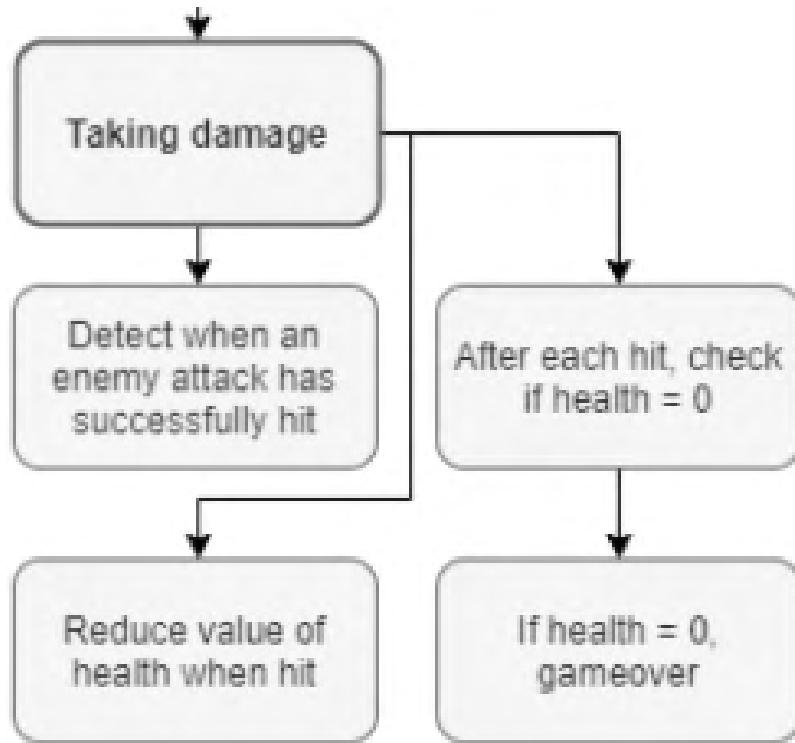
Hold down button, use grapple hook towards an object which is not an enemy or environment object (Erroneous)	Grapple hook/line does not render and the player does not move towards grapple point.		Pass
--	---	--	------

Note, some tests were conducted at a later point as some are dependent on other classes being developed first.

Player Take Damage

This class relates to the 'player damage' class from the design section.

Link to design section:



The player needs to be able to take damage, but also die if they have no health left. In this case they also need to be able to respawn, but only if they have lives remaining. This module aims to cover these rules and functionalities.

I extended the base player class with the following methods:

```
0 references
public void takeDamage(int _damage)
{
    // removes from health, given damage amount
    health -= _damage;
    // only checks death if damage taken
    checkDeath();
}
```

This method takes in an integer representing the amount of damage the player should take. When this module is called, which will be in the enemy attack class, the player will lose this much health when this module is called.

Then, the checkDeath() method is also called to check if the player is now dead after taking damage. This method is shown below:

```
1 reference
private void checkDeath()
{
    // when health is reduced to 0,
    // player should die and lose a life
    if (health <= 0)
    {
        lives -= 1;

        // if the player runs out of lives
        // then game over, otherwise respawn
        if (lives == 0)
        {
            gameOver();
        }
        else
        {
            respawn();
        }
        // only checks for game over,
        // each time player dies
    }
}
```

This method checks if the player's new health is below or equal to 0, in which case they are considered dead. In this case, they should either respawn or get a game over screen depending on how many lives they have left. So if the player has 0 lives left then they are shown the game over screen, otherwise they respawn. These methods have not yet been developed as this will be done later in the implementation process due to dependencies such as level generation.

For testing, I set up a system that calls the damage function when I left click, essentially dealing damage. This allows me to test the damage functionalities without yet implementing enemies.

```
void Start()
{
    Debug.Log("Starting Health = " + health);
    Debug.Log("Starting Lives = " + lives);
}

void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        takeDamage(10);
    }
}
```

I also added the following print statements to keep track of health, lives and function calls:

```
Debug.Log(_damage + " damage taken");

Debug.Log("Current Health = " + health);

Debug.Log("Current Lives remaining = " + lives);

1 reference
private void gameOver()
{
    Debug.Log("GAMEOVER");
    // display game over screen with stats, etc
    // 2 options; play again, or go to main menu
    // if play again, reset score and other stats
}

1 reference
private void respawn()
{
    Debug.Log("RESPAWN");
    // on same level, reset enemies, position, etc.
    // keep score and other stats
}
```

Testing

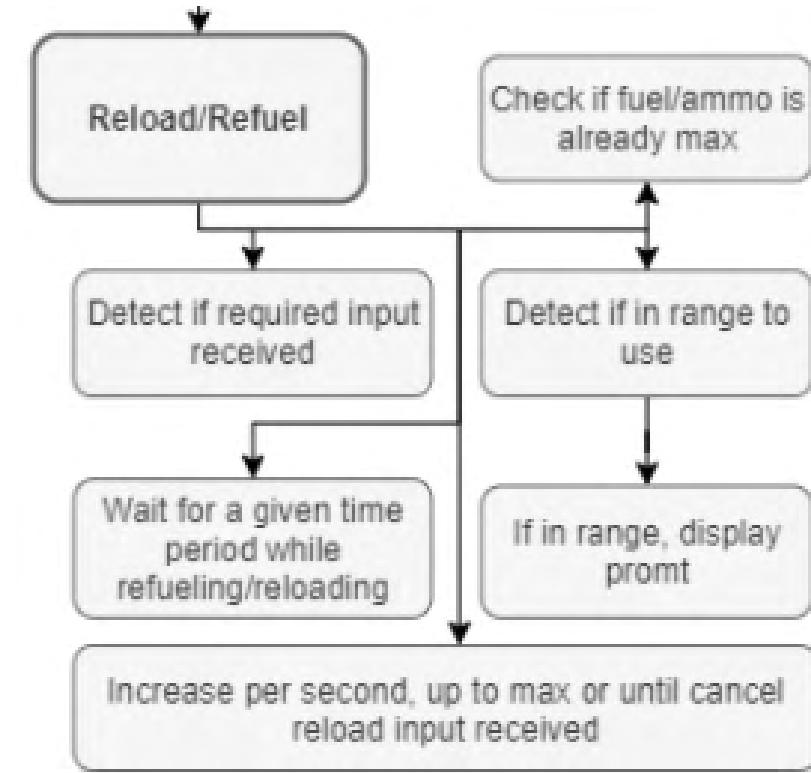
<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Player starts with 3 lives, 80 health and takes 30 damage (Normal)	Player health reduced to 50.	<p>[23:37:49] Starting Health = 80 UnityEngine.Debug:Log (object)</p> <p>[23:37:49] Starting Lives = 3 UnityEngine.Debug:Log (object)</p> <p>[23:37:50] 10 damage taken UnityEngine.Debug:Log (object)</p> <p>[23:37:50] Current Health = 70 UnityEngine.Debug:Log (object)</p> <p>[23:37:51] 10 damage taken UnityEngine.Debug:Log (object)</p> <p>[23:37:51] Current Health = 60 UnityEngine.Debug:Log (object)</p> <p>[23:37:52] 10 damage taken UnityEngine.Debug:Log (object)</p> <p>[23:37:52] Current Health = 50 UnityEngine.Debug:Log (object)</p>	Pass
Player starts with 3 lives, 20 health and takes 20 damage (Normal)	Player health is reduced to 0 and player loses a life and so respawns with 2 lives remaining.	<p>[23:41:04] Starting Health = 20 UnityEngine.Debug:Log (object)</p> <p>[23:41:04] Starting Lives = 3 UnityEngine.Debug:Log (object)</p> <p>[23:41:09] 10 damage taken UnityEngine.Debug:Log (object)</p> <p>[23:41:09] Current Health = 10 UnityEngine.Debug:Log (object)</p> <p>[23:41:10] 10 damage taken UnityEngine.Debug:Log (object)</p> <p>[23:41:10] Current Health = 0 UnityEngine.Debug:Log (object)</p> <p>[23:41:11] Current Lives remaining = 2 UnityEngine.Debug:Log (object)</p> <p>[23:41:11] RESPAWN UnityEngine.Debug:Log (object)</p>	Pass

Player starts with 1 lives, 10 health and takes 20 damage (Normal)	Player health is reduced to 0 and player loses a life. gameOver() is called as the player now has 0 lives remaining.	<pre>! [23:47:10] Starting Health = 10 UnityEngine.Debug.Log (object) ! [23:47:10] Starting Lives = 1 UnityEngine.Debug.Log (object) ! [23:47:14] 20 damage taken UnityEngine.Debug.Log (object) ! [23:47:14] Current Health = -10 UnityEngine.Debug.Log (object) ! [23:47:14] Current Lives remaining = 1 UnityEngine.Debug.Log (object) ! [23:47:14] GAMEOVER UnityEngine.Debug.Log (object)</pre>	Pass
--	--	--	------

Player Refuel

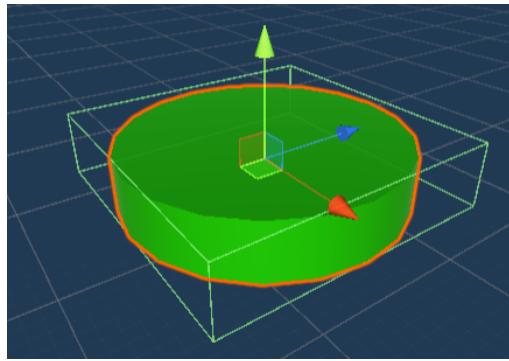
This class relates to the 'player refuel' class from the design section.

Link to design section:



The aim of this module is to allow the player to increase their fuel level if they are standing on top of a fuel station. More specifically, this requires the player to be standing still as well as standing on top of a fuel station. Further to this, regarding the game as a whole, the player refuelling should also decrease the fuel level of the fuel station in use.

Firstly, I created a refuel station gameobject:



This cylindrical station has a renderer to draw the cylindrical shape, and I decided to use a basic 3D box collider for collisions.

I then developed the script for its methods and attributes:

```
public class fuelStation : MonoBehaviour
{
    // the station has a fuel store,
    // i.e. how much fuel the station has
    4 references
    private float fuelStore = 100;

    // method to set the fuel level of the station
    0 references
    private void setStationFuel(float _fuel)
    {
        fuelStore = _fuel;
    }

    // returns the current fuel store
    2 references
    public float getFuelStore()
    {
        return fuelStore;
    }

    // method to decrease the fuel store,
    // by a given amount of fuel used
    1 reference
    public void useFuel(float _fuelUsed)
    {
        fuelStore -= _fuelUsed;
    }

    // method to check if fuel exists/
    // /is left in the fuel station
    2 references
    public bool checkEmpty()
    {
        if (fuelStore > 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
}
```

The station currently only has one attribute, fuelStore, a float value of how much fuel the station holds at any one time. Methods include a getter and setter for fuelStore as well as a method to check if the station is out of fuel or not.

Next, I developed the player script to allow the player to refuel:

```

public class playerRefuel : player
{
    // reference to the player class
    4 references
    player player;

    // flag to determine standing or not
    3 references
    private bool standing;

    // flag to determine if the player is allowed to refuel
    3 references
    private bool canRefuel;

    // reference to the fuel station to be used
    3 references
    private fuelStation currentRefuelStation;

    // mask to filter out everything except fuel stations
    1 reference
    public LayerMask stationMask;

    0 references
    void Start()
    {
        // accesses the player script
        player = GetComponent<player>();
        // starts the checkStanding coroutine/procedure
        StartCoroutine(CheckStanding());
    }
}

```

The player refuel class requires; a reference to the player class (accessed in the start method), a boolean flag for standing and canRefuel to denote between standing and being able to refuel, reference to the fuelStation script for the fuel station that the player uses at any time, and a mask to filter out ray collisions with colliders of that other than the colliders of fuel stations.

Note, the reference to the player script is done using the variable instead of relying just on the inheritance. I found this to be much better and so have updated all the other player scripts to do the same. This meant that the base player script needs many more getter/setter methods, which can all be seen in the appendix.

```

public IEnumerator CheckStanding()
{
    // while true loop to keep running while the game plays
    while (true)
    {
        // get player position (before time wait)
        Vector3 posBefore = this.transform.position;
        // wait for 2 seconds
        yield return new WaitForSeconds(1f);
        // get player position (after time wait)
        Vector3 posAfter = this.transform.position;

        // if the player is in the same position,
        // before and after the time wait,
        // then they are standing and so its set to true,
        // otherwise it is set to false
        if (posBefore == posAfter)
        {
            standing = true;
        }
        else
        {
            standing = false;
        }
    }
}

```

This coroutine procedure determines if the player is standing at any time or not. This is done by recording the player's position, waiting for 1 second, and then recording the new position of the player. If these 2 position values are the same, then the player is standing still. Therefore, standing is set to true and then this can be used to further determine if the player should refuel in the update() method, including checks to see where the player is standing still, i.e. if it is at a refuel station.

```

void Update()
{
    // if the player is standing,
    // which is checked in the checkStanding coroutine,
    // then we check if the player is on a refuel station
    if (standing)
    {
        // a ray is cast from the player, downwards
        Ray StationRay = new Ray(transform.position, -transform.up);
        RaycastHit hitInfo;

        // if the ray hits an object with the fuel station mask,
        // and the collision is <= 5 units from the player,
        // then the player can refuel
        if (Physics.Raycast(StationRay, out hitInfo, 5f, stationMask))
        {
            // the refuel station that the player is standing on
            currentRefuelStation = hitInfo.transform.gameObject.GetComponent<fuelStation>();
            // canRefuel set to true so that the player now refuels
            canRefuel = true;
        }
    }
    // otherwise, if the player is not standing on a station,
    // the player should not refuel
    else
    {
        canRefuel = false;
    }

    // if the player is to refuel,
    if (canRefuel)
    {
        // only if the player fuel is not full,
        // and the fuel station is not empty, should the player refuel
        if (player.getFuel() < 100 && !currentRefuelStation.checkEmpty())
        {
            // player fuel level is increased
            player.addFuel(0.2f);
            // same amount of fuel is decreased from the fuel station
            currentRefuelStation.useFuel(0.2f);

            Debug.Log("Player fuel level increased to: " + player.getFuel());
        }
    }
}

```

The update() method uses the standing boolean flag altered in the checkStanding() coroutine to check if the player is standing on a refuel station. This is done using a raycast downwards from the player. If this ray collides with a refuel station, then the refuel station script which includes the fuelStore attribute is accessed. Further checks are carried out, ensuring that the player is not already at max fuel and also that the fuel station in use has fuel left to be used. If so, the player's fuel is increased by 0.2 and the fuel station in use's fuel store decreases by the same amount.

My initial thoughts were that 0.2 is a very small value but since the update() method runs every frame, this means that at a 60 fps standard frame rate, the player would gain 12 fuel per second. However, I realised this meant that this was frame rate dependent, which is not ideal,

therefore I switched to using the `fixedUpdate()` method rather than `update()`. To more easily decipher if a fuel station has fuel left or not, I wrote a script to change the colour of the station to red when out of fuel, as this will make it clearer during testing.

```
public class fuelStationModes : fuelStation
{
    0 references
    void Update()
    {
        // if the fuel station has no fuel left,
        // the fuel station should turn red
        if (checkEmpty())
        {
            GetComponent<Renderer>().material.color = Color.red;
        }

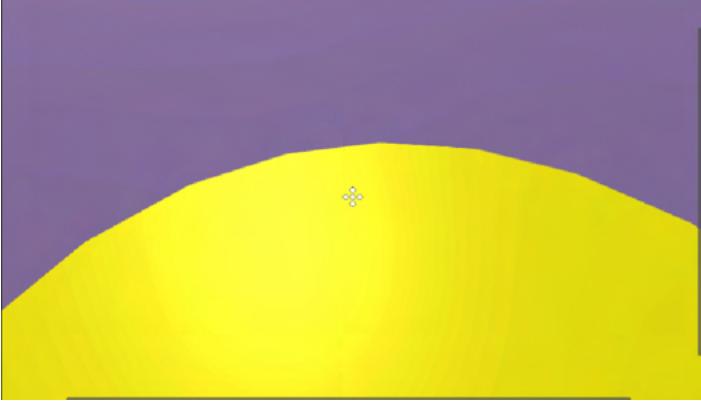
        // if the fuel level is between 0 and 50,
        // the fuel station should turn yellow
        if (getFuelStore() > 0 && getFuelStore() < 50f)
        {
            GetComponent<Renderer>().material.color = Color.yellow;
        }

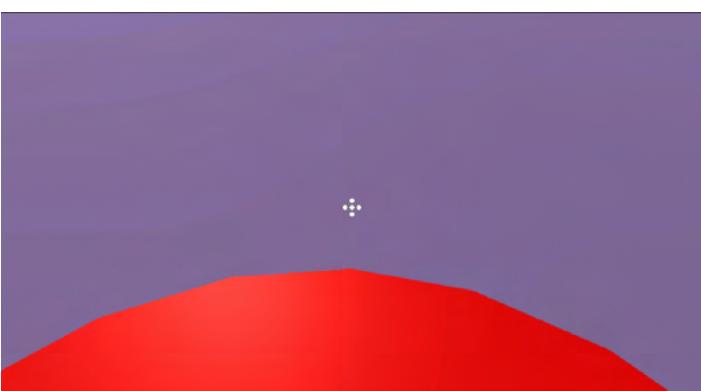
        // otherwise, if the fuel level is over 50,
        // the fuel station should be green
        else
        {
            GetComponent<Renderer>().material.color = Color.green;
        }
    }
}
```

This script simply changes the colour of the material of the fuel station depending on its fuel level.

- 50 - 100 means the station will appear green
- 0 - 50 means the station will appear yellow
- 0 means the station will appear red

Testing

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Player stands still, not on a refuel station (Normal)	Player's fuel does not increase, fuel station retains fuel level.	 Player stood still, although no fuel levels were altered.	Pass
Player, with less than 100 fuel, e.g. 50, stands still on a refuel station, with 50 fuel (Normal)	Player's fuel increases and the station in use undergoes a decrease in fuel by the same amount.	 <div style="background-color: black; color: white; padding: 5px;"> ! [00:13:27] Player fuel level increased to: 86.19971 UnityEngine.Debug:Log (object) ! [00:13:27] Fuel station fuel level decreased to: 63.80055 UnityEngine.Debug:Log (object) </div> <div style="background-color: black; color: white; padding: 5px;"> ! [00:13:27] Player fuel level increased to: 87.59969 UnityEngine.Debug:Log (object) ! [00:13:27] Fuel station fuel level decreased to: 62.40054 UnityEngine.Debug:Log (object) </div> <div style="background-color: black; color: white; padding: 5px;"> ! [00:13:27] Player fuel level increased to: 89.59966 UnityEngine.Debug:Log (object) ! [00:13:27] Fuel station fuel level decreased to: 60.40054 UnityEngine.Debug:Log (object) </div>	Pass

Player, with 100 fuel, stands still on a refuel station with 50 fuel (Normal)	Player's fuel does not increase, all fuel stations retain fuel level.	 Player did not refuel	Pass
Player, with less than 100 fuel, stands still on a refuel station with 0 fuel (Normal)	Player's fuel does not increase, all fuel stations retain fuel level.	 Player did not refuel	Pass

Progress Review 1

At this point in the development process, the main menu and the player has been developed completely. **This fulfills requirements 4.1, 4.3, 4.4, 8.1, 8.2, and 8.5.**

The main menu is completed, although the play button needs to be hooked up to the algorithm which will generate levels (this has not yet been developed) and the settings page exists, although needs to have options to change certain settings in this page. The quit button does already close the application. Regarding the player, all functionalities are complete including moving, jumpung, using the grappling gear, refueling, attacking and taking damage. The only potential exception is that there are no weapons currently for the player to use, although once the weapons system is implemented, this will instantly start working as the player attack module is completed such that it will work with any implemented weapons.

This roughly represents 35% of the overall development completed. However, with a tight deadline, this is not currently an ideal rate of progress. Arguably, I have been learning Unity more and more while developing and will continue to do so, however I feel much more confident now to accelerate the development process. I expect to make less logical errors within the program and this should mean that I can complete more modules in a shorter amount of time. Overall, I still hope to complete all the requirements that have been set out, however, the more desirable features are unlikely to all be completed. The user accounts feature would take quite a lot of time and so may have to be left out but I may possibly still include saving player stats via databases in some way or another.

I presented the current prototype to my clients for feedback, they really liked what they saw and their main feedback and suggestions were:

1. Part of the grappling line disappears at times, a part of the line closest to the player
2. It is hard to know when the player has run out of fuel, it would be better to have some kind of indicator, just as the stations do when they run out of fuel

I then addressed these suggestions

1. This can be improved by changing the near clipping plane of the player's camera. Currently set to 0.1, the part of the line 0.1 units or closer to the (player) camera did not render. Therefore, I have changed the near clipping plane to 0.01.
2. On screen UI will help the player keep track of their health and fuel visually, this has not yet been developed. Explaining this to the client, they were on board with having GUI implemented to solve this issue in a later stage of the development.

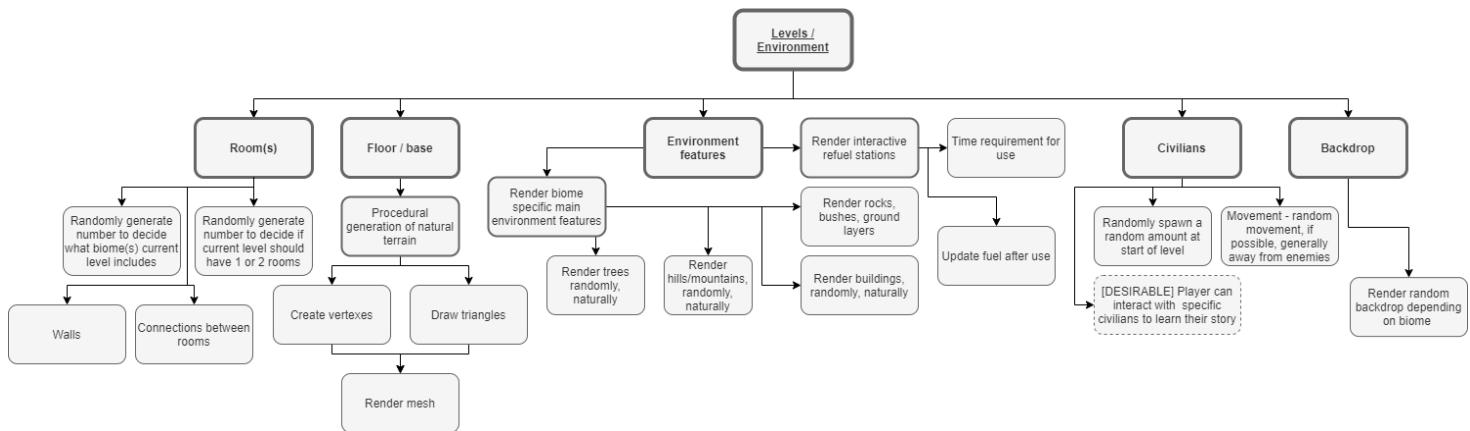
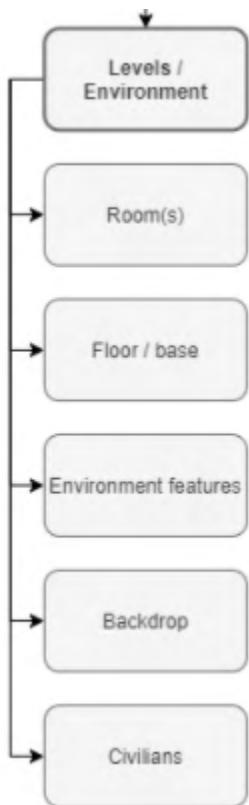
In the end, the client was happy for me to move forwards with the development.

Overall, this review has clearly indicated that I need to speed up the development process and avoid creating so many errors within the program.

Environment generation

This section meets requirement(s): 2.1, 2.2, 2.3, 2.4, 2.6, 2.7 and relates to the 'environment generator' class from the design section.

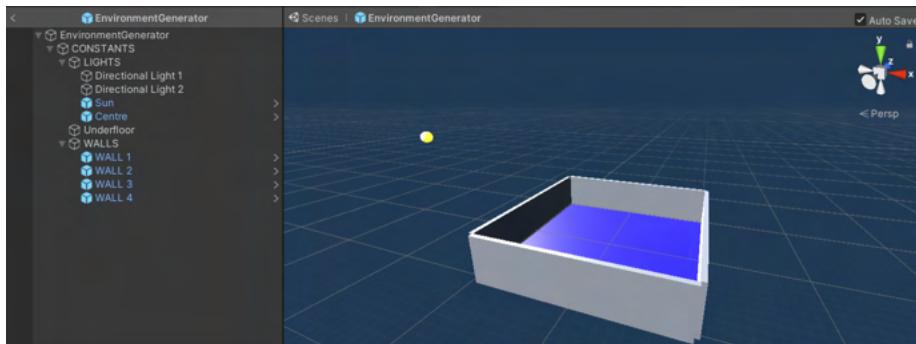
Link to design section:



Environment constants

Each level is to be randomly generated in terms of the terrain, biome, environment features, enemy spawns, etc. However, fundamentally, each level will have some features which are always consistent. For example, the walls of the level, the lighting, etc.

Therefore, I constructed a fundamental level prefab which can be built upon to form a randomly generated level each time this is required:



As seen, the overall environment generator prefab contains children including the constants which are required in every level regardless of the randomization. This involves the walls, underfloor base and the lighting. These can be seen in the preview on the right. The sun has been created such that it has an emission material so it shines like a sun and rotates around the centre object, which is located in the middle of the underfloor.

The script for the sun to rotate around the map is as seen below:

```
public class Sun : MonoBehaviour
{
    // the target which you rotate around
    1 reference
    public GameObject target;

    // the speed at which you rotate, around the target
    1 reference
    public int speed = 1;

    // the x,y,z values of the direction that this object rotates around
    1 reference
    public float x = 1f;
    1 reference
    public float y = 1f;
    1 reference
    public float z = 0f;

    0 references
    void Update()
    {
        // rotates around target in the direction x,y,z with speed, speed
        transform.RotateAround(target.transform.position, new Vector3(x, y, z), speed * Time.deltaTime);
    }
}
```

Terrain mesh generation

To generate the actual terrain of the level, this must be done by generating a custom mesh, and then randomising y-values (the height of each vertex) using noise. The variables are shown below:

```
#region MESH GENERATION VARS
// the mesh that we will configure
8 references
private Mesh mesh;

// array which will contain generated vertices
3 references
private Vector3[] vertices;

// array which will contain generated triangles
8 references
private int[] triangles;

// Length of terrain mesh
7 references
private int xSize = 200;

// width of terrain mesh
4 references
private int zSize = 200;

// scale for perlin noise
2 references
private float noiseScale = 0.1f;
#endregion
```

I wrote the following script to generate this terrain mesh:

```
private void GenerateMesh()
{
    // instantiates a new mesh
    mesh = new Mesh();
    GetComponent<MeshFilter>().mesh = mesh;

    // instantiates a new array storing vertices
    vertices = new Vector3[(xSize+1)*(zSize+1)];

    // sets vertices to x, z values with random y value
    for (int i = 0, z = 0; z <= zSize; z++)
    {
        for (int x = 0; x <= xSize; x++)
        {
            float y = Mathf.PerlinNoise(x * noiseScale, z * noiseScale) * 2f;
            vertices[i] = new Vector3(x, y, z);
            i++;
        }
    }
}
```

Initially, we instantiate a new mesh and generate vertices with x, y, and z values, y values generated with parameters x and z so that the terrain is smooth since 2 different height values will be similar if the relative x and z values are similar. The noise scale is applied since perlin

noise actually repeats every integer, so this scale is applied to ensure the same values aren't generated and we don't get a completely flat plane.

```
// instantiates a new array of triangles for mesh
triangles = new int[xSize*zSize*6];

// sets triangles using vertexes
int vert = 0;
int tris = 0;
for (int z = 0; z < zSize; z++)
{
    for (int x = 0; x < xSize; x++)
    {
        triangles[0 + tris] = vert + 0;
        triangles[1 + tris] = vert + xSize + 1;
        triangles[2 + tris] = vert + 1;
        triangles[3 + tris] = vert + 1;
        triangles[4 + tris] = vert + xSize + 1;
        triangles[5 + tris] = vert + xSize + 2;

        vert++;
        tris += 6;
    }
    vert++;
}
```

Next, we generate the triangles of the mesh using this array of vertices for every x and z vertex. This uses a new array for these triangles with size $x * z * 6$, since there will be 6 triangles per vertex. Essentially, this 'connects' the vertices together with triangles which overall, generates a mesh.

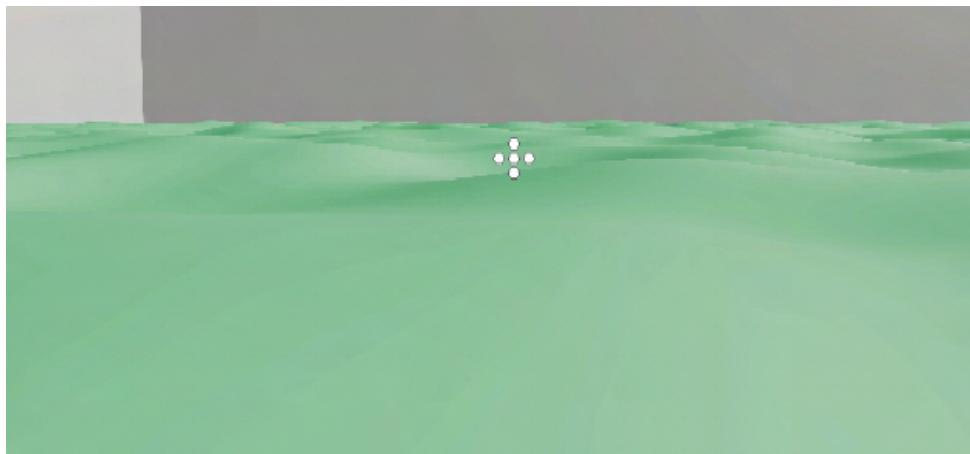
```
// clears any previous mesh
mesh.Clear();

// sets the vertexes of the mesh to the ones we generated
mesh.vertices = vertices;
// sets the vertexes of the mesh to the ones we generated
mesh.triangles = triangles;

// updates mesh attributes for various functionalities,
// such as showing textures, adding a collider, etc
mesh.RecalculateNormals();
mesh.RecalculateBounds();
MeshCollider meshCollider = gameObject.GetComponent<MeshCollider>();
meshCollider.sharedMesh = mesh;
```

Then we simply set the mesh's vertices and triangles to the generated arrays and configure some of the mesh settings so that the mesh and any materials applied works well with the lighting, and enables a collider too, so that the player can actually walk/run on the terrain.

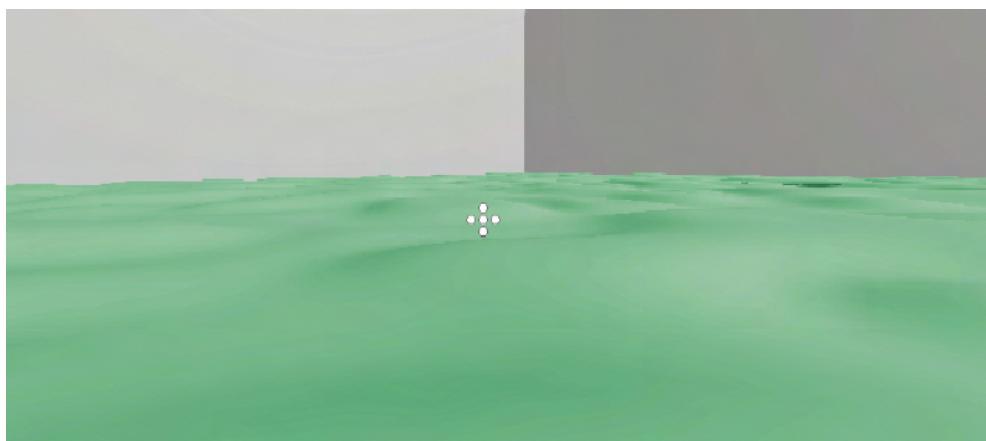
However, the terrain generated seemed a bit repetitive (as seen below) due to the perlin noise alone. Instead I tried layering different noise and other mathematical functions (such as trigonometric ones).



I decided to generate the y-values in the following way:

```
// y value calculated by layering perlin noise multiple times,  
// on top of itself and then layering it with the sin  
float y1 = Mathf.PerlinNoise(x * noiseScale, z * noiseScale) * 2f;  
float y2 = Mathf.PerlinNoise(x * noiseScale, z * noiseScale) * 2f;  
float y = Mathf.Sin(Mathf.PerlinNoise(y1, y2));
```

And this proved to add more variety:



Random Biome selection

Each level needs to be designated a biome so that relevant environment features can be spawned onto the level that follow that theme. This should be done randomly. The relevant variables used are shown below:

```
#region ENVIRONMENT VARS

// game object List of all spawned environment objects
// used to destroy these when a new level is generated
3 references
private List<Object> allEnvObjects = new List<Object>();

// string array of length 5 storing the different biome names
2 references
private string[] biomes = new string[5] {"town", "city", "forest", "snow", "desert"};

// string for the chosen biome
7 references
private string chosenBiome;

// array of gameobjects to be randomly spawned into the Level
9 references
private GameObject[] envFeatures;

// colour for the mesh to be set to depending on biome
7 references
private Color groundColor;

// amount of space between each environment feature spawned
8 references
private int spaceBetween;

// amount of space above the ground each environment feature should be spawned
7 references
private int groundOffset;
```

In this function, the biomes array will be used to randomly select a biome. Then the environment features array will be populated accordingly as well configuring some of the other variables such as the ground colour, space between and the ground offset.

```

private void DecideBiome()
{
    // random number between 0 and the amount of biomes
    int index = Random.Range(0, biomes.Length);
    // this number used to choose random biome via index
    chosenBiome = biomes[index];

    // then depending on the biome,
    // envFeatures is populated with the relevant gameobjects
    // groundColour, spaceBetween and groundOffset are configured

    if (chosenBiome == "town")
    {
        envFeatures = new GameObject[7] {house_1, house_2, house_3, house_4, house_5, townhall, tower};
       groundColor = Color.grey;
        spaceBetween = 50;
        groundOffset = 0;
    }

    else if (chosenBiome == "city")
    {
        envFeatures = new GameObject[5] {skyscraper_1, skyscraper_2, multistory_1, multistory_2, office};
       groundColor = Color.black;
        spaceBetween = 20;
        groundOffset = 0;
    }

    else if (chosenBiome == "forest")
    {
        envFeatures = new GameObject[8] {tree_1, tree_2, tree_3, bush_1, bush_2, branch, boulder, stump};
       groundColor = new Color32(63, 120, 84, 251);
        spaceBetween = 25;
        groundOffset = 0;
    }

    else if (chosenBiome == "rainforest")
    {
        envFeatures = new GameObject[4] {talltree_1, talltree_2, bigbush, mudpatch};
       groundColor = Color.green;
        spaceBetween = 20;
        groundOffset = 0;
    }
}

```

```

    else if (chosenBiome == "snow")
    {
        envFeatures = new GameObject[3] {snowtree_1, snowtree_2, snowtree_3};
       groundColor = Color.white;
        spaceBetween = 50;
        groundOffset = 0;
    }

    else if (chosenBiome == "desert")
    {
        envFeatures = new GameObject[4] {giantcacti_1, giantcacti_2, giantcacti_3, giantcacti_4};
       groundColor = Color.yellow;
        spaceBetween = 50;
        groundOffset = 20;
    }
    else
    {
        // this instantiates the list,
        // so that from a logical perspective,
        // it is always defined,
        // before it is referenced in the generate environment function
        envFeatures = new GameObject[0] {};
    }
}

```

So overall, this function randomly chooses a biome for the level being generated and sets the relevant environment feature game objects as the environment features array so that this can be used later to spawn the environment game objects. Note, the variables which hold the environment variables are explained in the next function.

Environment features generation

The environment features array, containing the environment game objects to be used for a given level, needs to now be used to spawn these environment game objects into the level. Variables used for the random biome selection will be used here as well as the following:

```

#region town
1 reference
public GameObject house_1;
1 reference
public GameObject house_2;
1 reference
public GameObject house_3;
1 reference
public GameObject house_4;
1 reference
public GameObject house_5;
1 reference
public GameObject townhall;
1 reference
public GameObject tower;
#endregion

#region city
1 reference
public GameObject skyscraper_1;
1 reference
public GameObject skyscraper_2;
1 reference
public GameObject multistory_1;
1 reference
public GameObject multistory_2;
1 reference
public GameObject office;
#endregion

#region forest
1 reference
public GameObject tree_1;
1 reference
public GameObject tree_2;
1 reference
public GameObject tree_3;
1 reference
public GameObject bush_1;
1 reference
public GameObject bush_2;
1 reference
public GameObject branch;
1 reference
public GameObject boulder;
1 reference
public GameObject stump;
#endregion

#region snow
1 reference
public GameObject snowtree_1;
1 reference
public GameObject snowtree_2;
1 reference
public GameObject snowtree_3;
#endregion

#region desert
1 reference
public GameObject giantcacti_1;
1 reference
public GameObject giantcacti_2;
1 reference
public GameObject giantcacti_3;
1 reference
public GameObject giantcacti_4;
#endregion

#region rainforest
1 reference
public GameObject talltree_1;
1 reference
public GameObject talltree_2;
1 reference
public GameObject bigbush;
1 reference
public GameObject mudpatch;
#endregion

```

These are public game object variables, this means that in the Unity inspector I can set these variables to be the relevant prefabs models that I want.

Find screen shots of the free models I've downloaded to use as these environment features in the appendix, with reference to where I found them.

Then I developed the function to spawn these environment features:

```

private void GenerateEnvironment()
{
    // sets the render's colour attribute to the required ground colour
    GetComponent<Renderer>().material.color = groundColor;

    // Loops through the width of the level,
    // incrementing with the required space between objects
    for (int x = 20; x <= 180; x += spaceBetween)
    {
        // Loops through the length of the level,
        // incrementing with the required space between objects
        for (int z = 20; z <= 180; z += spaceBetween)
        {
            // random number between 0 and the amount of environment features
            int randomInt = Random.Range(0, envFeatures.Length);
            // this number used to choose random environment feature via index
            GameObject currentFeature = envFeatures[randomInt];
            // the position of this environment feature at this x, z position,
            // with height equal to the required ground offset
            Vector3 envPos = new Vector3(x, 1 + groundOffset, z);
            // the rotation of this environment feature with 0 x, z rotation,
            // with a random y-axis rotation between 0 and 360 degrees
            Quaternion envRotation = Quaternion.Euler(new Vector3(0, Random.Range(0, 360), 0));
            // spawns the environment feature at this position and rotation
            // and adds this to the list of all environment objects
            allEnvObjects.Add(Instantiate(currentFeature, envPos, envRotation));
        }
    }
}

```

The function does 2 main things; sets the colour of the generated terrain mesh to the relevant colour for the current biome, and spawns randomly selected environment features. The second requirement is met using 2 for loops, 1 nested inside the other. This loops through each x and z position, i.e. the length and width of the level, and increments using the space between variable, spawning the environment object at each position with random rotation.

Fuel stations generation

Each level needs to have some refuel stations so that the player can refuel when they run out of fuel, to keep playing. 5 fuel stations per level would be adequate. These need to be spawned onto the level at random positions but such that they do not spawn inside of other already existing environment features.

Here are the relevant variables that will be used:

```

#region FUEL STATION VARS
// reference to the fuel station model so that it can be spawned
1 reference
public GameObject fuelStation;

// layer mask to only detect the environment
1 reference
public LayerMask environmentMask;

// layer mask to detect only other fuel stations
1 reference
public LayerMask stationMask;

// flag to check if there is an object directly below a point
4 references
private bool objectBelow;

// current position of fuel station before spawning
4 references
private Vector3 stationPos;
#endregion

```

```

private void GenerateFuelStations()
{
    Vector3 randomPos;
    Ray groundCheck;

    // Loops 5 times to spawn 5 fuel stations
    for (int i = 0; i < 5; i++)
    {
        // while the current position where the fuel station is to be placed
        // has an object below it, choose a new random position
        while (objectBelow)
        {
            // generates new x-z random position
            int randx = Random.Range(20, 180);
            int randz = Random.Range(20, 180);
            randomPos = new Vector3 (randx, 2, randz);

            // cast ray down to check for objects below
            groundCheck = new Ray(randomPos, -transform.up);
            RaycastHit groundHit;

            // check if there is an object below and set the bool flag accordingly
            if (Physics.Raycast(groundCheck, out groundHit, environmentMask))
            {
                objectBelow = true;
            }
            else if (Physics.Raycast(groundCheck, out groundHit, stationMask))
            {
                objectBelow = true;
            }
            else
            {
                objectBelow = false;
                stationPos = randomPos;
            }
        }

        // fuel station is instantiated at this random position
        Instantiate(fuelStation, stationPos, Quaternion.identity);
    }
}

```

This function repeatedly generates random positions for the 5 fuel stations; if a position has an environment object, or another fuel station already below it, then it randomly selects another position. Once a valid position is chosen, a fuel station is spawned there. This repeats until all 5 stations are spawned onto the level.

Spawning Enemies

Enemies must be spawned consistently throughout the level; if there are too many enemies concentrated in one place, it may be too difficult for the player to defeat them. The enemy prefab (developed in the Enemies development section) can simply be instantiated to spawn an enemy, therefore we just need to find a suitable position for each enemy to be spawned.

Therefore, I developed the following function:

```
private void SpawnEnemies()
{
    // spawns enemies across a number of rows and columns
    for (int x = 20; x <= 180; x += 80)
    {
        for (int z = 20; z <= 180; z += 80)
        {
            // Instantiates enemy at this position
            Instantiate(enemy, new Vector3(x, 10, z), Quaternion.identity);
            enemyCount++;
        }
    }
}
```

The function loops through the length and width of the map, evenly spawning enemies throughout, essentially in rows and columns. The enemy count is incremented every time, as this will need to be kept track of so that when it reaches 0, the program knows that the level has been completed.

Spawning the Player

The player can be randomly spawned within the level, to keep with the random nature of this level generation:

```
private void SpawnPlayer()
{
    // chooses a random x and z value for where the player should spawn
    int randx = Random.Range(20, 180);
    int randz = Random.Range(20, 180);
    Vector3 randomPos = new Vector3 (randx, 20, randz);
    // spawns player by instantiating the player prefab
    Instantiate(player, randomPos, Quaternion.identity);
}
```

A random position is chosen within the length and width of the level and the player prefab is spawned.

New level

The player should move on to the next level when all the enemies are defeated. This occurs when there are no enemies remaining. At this point, a new level should be generated for the player.

```
private void Update()
{
    Debug.Log("Enemies remaining: " + enemyCount);

    // if there are no enemies left,
    // and if a new level is not currently being created,
    // then a new level should start being created
    if (enemyCount == 0 && newLevelAllowed)
    {
        NewLevel();
    }
}
```

In the update method, the program constantly checks if there are any enemies left and if a new level is allowed, which is always true unless another level is already being generated.

```
private void NewLevel()
{
    // ensures that a new level generation,
    // is not start until this one is finished
    newLevelAllowed = false;
    // destroys all previous level objects
    DestroyPrev();
    // generates the floor
    GenerateMesh();
    // randomly decides which biome will be used
    DecideBiome();
    // randomly spawns relevant environment features
    GenerateEnvironment();
    // randomly spawns fuel station across the map
    GenerateFuelStations();
    // randomly spawns enemies across the map
    SpawnEnemies();
    // spawns the player at a random position
    SpawnPlayer();
    // this level has now been generated,
    // so a new level is now allowed
    newLevelAllowed = true;
}
```

Generating a new level requires the previous one to be destroyed first, then the same initial generation functions can be called to generate a new level. While this happens, new level allowed is set to false but afterwards, it is set back to true.

```
private void DestroyPrev()
{
    // destroy the current terrain mesh
    Destroy(mesh);

    // destroy all current environment features
    GameObject[] envObjects = GameObject.FindGameObjectsWithTag("Environment");
    for(int i = 0; i < envObjects.Length; i++)
    {
        Destroy(envObjects[i]);
    }

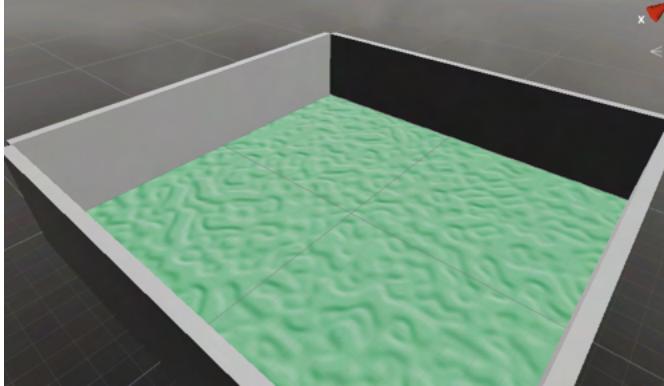
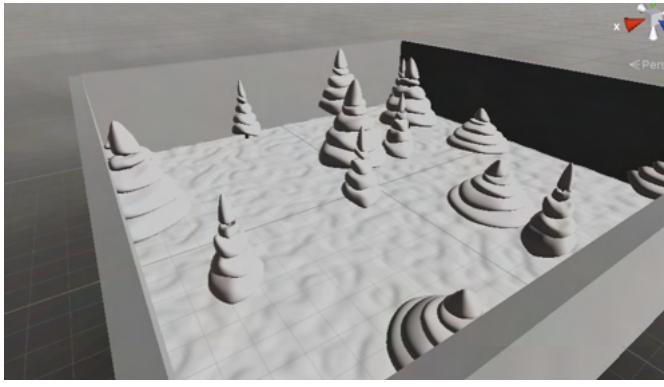
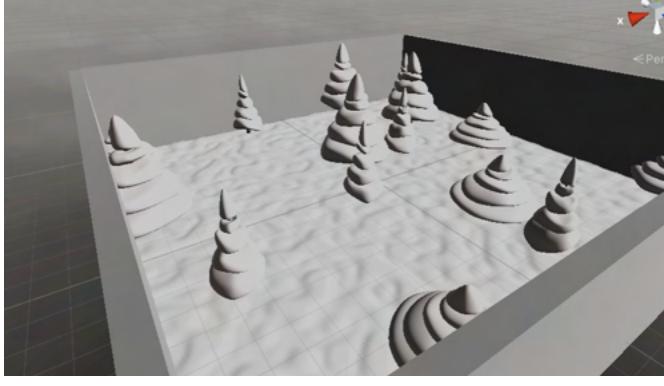
    // destroy all current fuel stations
    GameObject[] fuelStations = GameObject.FindGameObjectsWithTag("FuelStation");
    for(int i = 0; i < fuelStations.Length; i++)
    {
        Destroy(fuelStations[i]);
    }

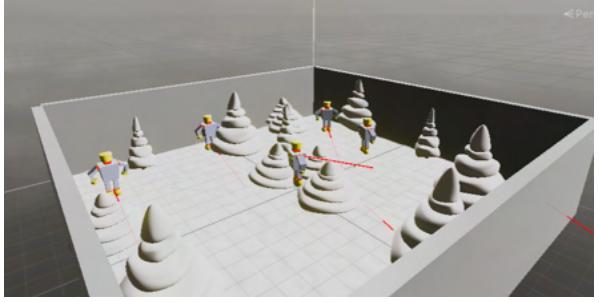
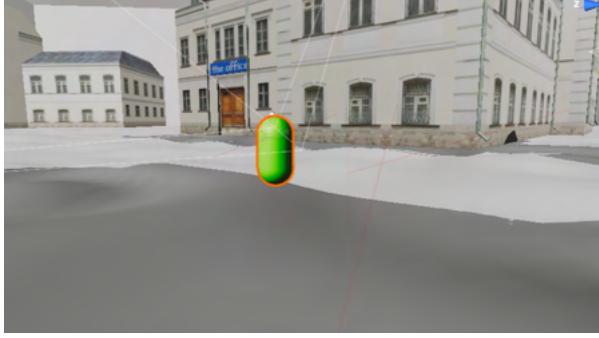
    // destroy all current remaining enemies
    GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
    for(int i = 0; i < enemies.Length; i++)
    {
        Destroy(enemies[i]);
    }

    // destroy player as new one will be spawned
    Destroy(player.gameObject);
}
```

This function removes the previous level by destroying the mesh, all environment objects, all fuel stations, all enemies (in case there is a mistake and some are left - this avoids the program failing and allows the player to keep playing) and ultimately the player is also destroyed. This is because new instances of all these things will be generated.

Testing 1

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Select play from menu (start gameplay)	-	-	-
↳	Base mesh generated successfully		Pass
↳	Random biome applied successfully		Pass
↳	Biome specific environment features spawned correctly		Pass

↳	Fuel stations spawned correctly	 Fuel stations all spawn in one place, at the corner of the level	Fail
↳	Door/link and other room generated successfully	- No second room feature currently implemented	Fail
↳	Enemies spawned successfully		Pass
↳	Player spawned successfully		Pass

↳	Process repeated to generate new level		Fail
---	--	--	------

Iteration 2

3 tests here were failed; the fuel stations would not generate properly, there is no second room to each level and new levels don't generate properly when one is completed.

I decided that the best decision here would be not to implement the second room feature on each level for a few reasons. Firstly, after actually developing the environment generation, it feels as if this wouldn't bring much benefit to the game - with only 4 biomes, having 2 rooms per level would just feel like 2 levels but you are able to travel between them. Instead, having separate levels means that gameplay is essentially the same, but since 2 rooms don't have to be playable at the same time, the performance of the program will be much better, as levels can be deleted after they are completed.

For the fuel station generation, the problem was in the use of the boolean variable `objectBelow`. This bool was initially undefined, and then altered whether there was an object under the random chosen position for the fuel station. However, between fuel station generations, this variable was not reset. Therefore I added code to reset this variable after a fuel station had been spawned:

```
// fuel station is instantiated at this random position,
// and is appended to the list of all fuel stations
objectBelow = true;
Instantiate(fuelStation, stationPos, Quaternion.identity);
```

After a level is completed, new levels would not spawn properly due to one main reason, being that game objects such as environment features were not correctly being destroyed before the new ones are added. This means that the level is oversaturated with environment objects, as well as fuel stations, player instances etc.

This means that the DestroyPrev method was not working properly. It relies on finding all these objects via tag and then adding them to a list. This clearly doesn't work, and I realised it was due to the lists being gameobject lists. To add all the gameobjects, all the environment gameobjects for example, and be able to destroy them, the list must be an Object type list. Also, finding these objects via their tag is not possible as this doesn't work with object lists. Instead, I can ensure that all objects that I want to remove are in the list by adding them to the list as they are spawned.

The below code adds environment features, fuel stations and enemies to their respective lists as they are instantiated:

```
// spawns the environment feature at this position and rotation  
// and adds this to the list of all environment objects  
allEnvObjects.Add(Instantiate(currentFeature, envPos, envRotation));  
  
// fuel station is instantiated at this random position,  
// and is appended to the list of all fuel stations  
allFuelStations.Add(Instantiate(fuelStation, randomPos, Quaternion.identity));  
Debug.Log("Station at: " + stationPos);  
  
// Instanciates enemy at this position  
// and adds it to list of all enemies  
allEnemies.Add(Instantiate(enemy, new Vector3(x, 15, z), Quaternion.identity));  
enemyCount++;
```

Then these lists can be looped through when all the objects are to be destroyed:

```

// destroy all current environment features
for (int i = 0; i < allEnvObjects.Count; i++)
{
    Destroy(allEnvObjects[i]);
}

// destroy all current fuel stations
for (int i = 0; i < allFuelStations.Count; i++)
{
    Destroy(allFuelStations[i]);
}

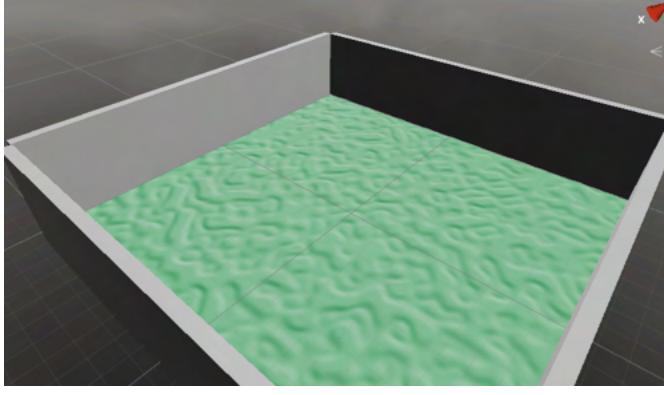
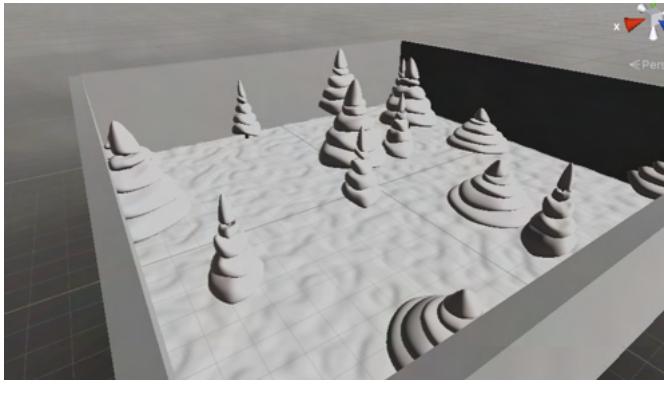
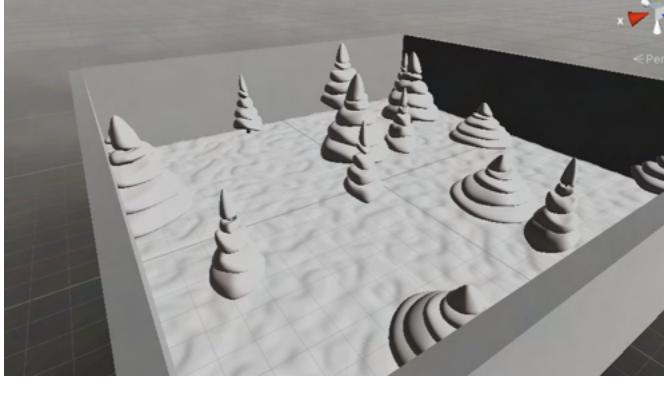
// destroy all current remaining enemies
for (int i = 0; i < allEnemies.Count; i++)
{
    Destroy(allEnemies[i]);
}

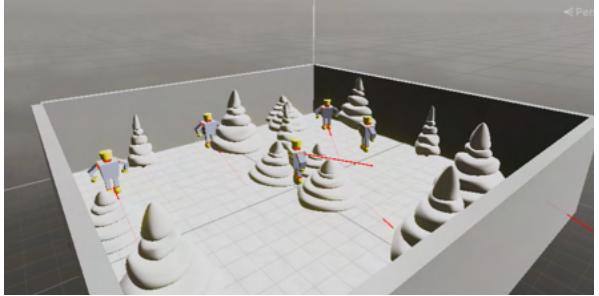
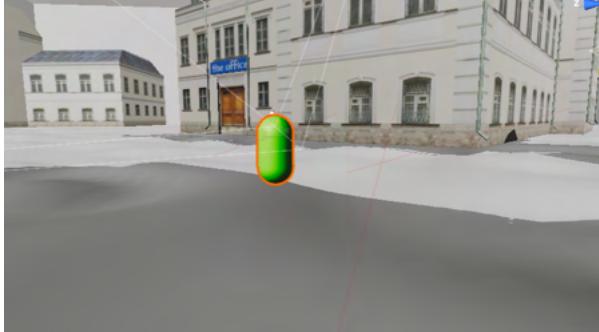
```

Civilians

After experimenting with civilians in each level, I realised that this feature did not add much value to the game. Furthermore, generating lots of civilians was very computationally expensive and meant that the performance of the program was sacrificed to an extent. Therefore, I decided to remove this feature from my game and leave it out.

Testing 2

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Select play from menu (start gameplay)	-	-	-
↳	Base mesh generated successfully		Pass
↳	Random biome applied successfully		Pass
↳	Biome specific environment features spawned correctly		Pass

↳	Fuel stations spawned correctly		Pass
↳	Door/link and other room generated successfully	- No second room feature currently implemented	Fail
↳	Enemies spawned successfully		Pass
↳	Player spawned successfully		Pass
↳	Process repeated to generate new level	 This level spawned correctly after the forest level	Pass

Progress Review 2

Since the last progress review, the environment / level generation has been developed completely, with the exception of the double room and civilians features. **This fulfills requirements 2.1, 2.2, 2.3 (to an extent), 2.4, 2.6.**

At this point, the level generation has been fully completed, i.e. levels now generate when the players click play to start the game, and then also when the player completes a level and wishes to move onto the next one. This level generation includes the generation of custom random/procedurally generated terrain mesh to form a base, and then the generation of environment features randomly throughout the level depending on the randomly selected biome, from; forest, snow, city/town, desert. Fuel stations also are randomly spawned onto the level so that the player can refuel if they run out of fuel. Features such as the 2nd room feature and civilians feature have been left out due to various reasons such as; lack of time, not relevant / add no value to the game, better performance without these features.

I presented the current prototype to my clients for feedback, they were really happy with the program so far. I also made sure to get their take on the fact that some small features were left out, so that if they really wanted these features, I could add them. However, the clients were happy with these features being left out as they understood the performance reasons and would also prefer the solution to be completed earlier.

"We agree that it is not necessary to implement these features at this time since the program is still working and is playable, i.e. we have a viable product that we can sell, the game is not incomplete without these features. At a later stage, we may decide that we want these features implemented and ask you to do so."

I was happy with this feedback as I agree that these features can be implemented at a later stage if need be.

Overall, at this stage, roughly 70% of the development has been completed. With an approaching deadline, this is a little bit behind where I would like to be, but I am generally happy with the progress and believe that I can increase the rate of progress as I become more fluid with working with Unity and programming the game in C#.

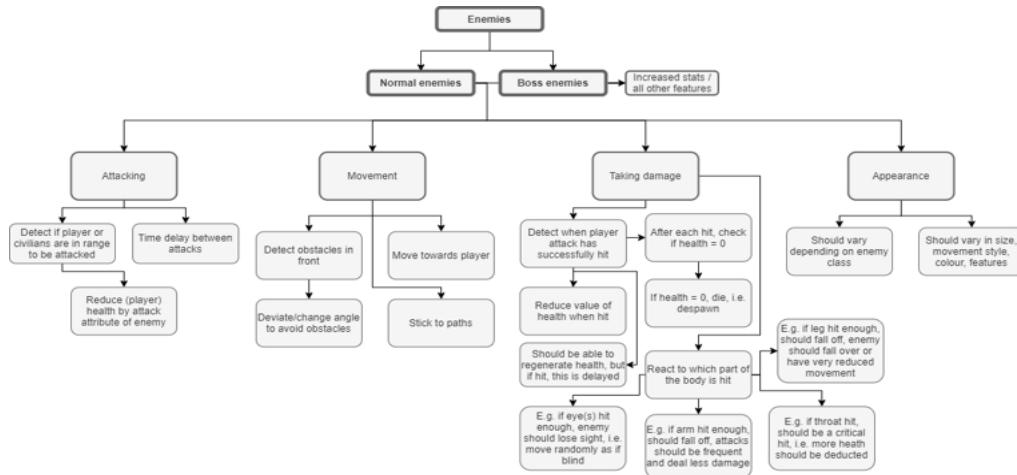
In the end, the client was happy for me to move forwards with the development.

Overall, this review has indicated that the rate of progress must increase at least a little to be able to complete the development in time. I believe that this is possible as I have learnt a lot more now, for example, working with unity game objects, colliders, random generation and even how to better work with data structures as ADT lists. I will aim to be more consistent and therefore make less mistakes, wasting less time dealing with errors.

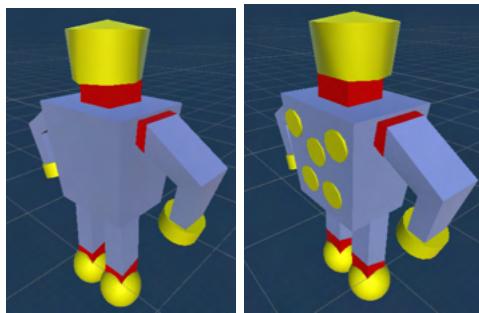
Enemies

This section meets requirement(s): 7.1, 7.2, 7.3, 7.4, 7.5 and relates to all of ‘Enemies’, i.e. ‘enemy movement’, ‘enemy appearance’, ‘enemy attack’, ‘enemy damage’ and from the design section.

Link to design section:



I started by creating an enemy model and converted it into a prefab so that it can be instantiated multiple times. I also added a character controller component to this prefab.



The images show the enemy model from the front and back respectively. I decided to use quite an unusual choice of colours for 2 reasons, firstly the player needs to know what the weak points of the enemy are, and this is achieved using distinctly different colours. Furthermore, one of the limitations of my game I discussed during the analysis section, was that people with visual impairment would find it difficult to play the game as it is a video game, however, by using contrasting colours, this helps people with visual impairments or partial vision loss to better be able to play the game.

As this enemy model consists of multiple gameobjects, I couldn't use a rigidbody for the gravity of the whole model. Instead I had to simulate gravity for the enemy myself, through script. Therefore I developed the following class which, if the player is not on the floor already, moves the player downwards by a velocity value calculated using the equation, $v = \frac{1}{2} a t^2$.

```

public class enemyGravity : enemy
{
    // reference to the enemy's character controller component
    2 references
    private CharacterController controller;

    // velocity vector
    3 references
    private Vector3 velocity;

    // constant value of gravity to calculate velocity at any time
    1 reference
    private float gravity = -9f;

    // bool flag to check whether the enemy is on the floor or not
    3 references
    private bool grounded;

    0 references
    void Start()
    {
        // gets the enemy's character controller component
        controller = GetComponent<CharacterController>();
    }

    void Update()
    {
        // a ray is cast from the enemy, downwards
        Ray groundCheck = new Ray(transform.position, -transform.up);
        RaycastHit groundHit;

        // if the ray hits an object,
        // and the collision is <= 5 units from the enemy,
        // then the enemy is grounded
        if (Physics.Raycast(groundCheck, out groundHit, 5f))
        {
            grounded = true;
        }
        else
        {
            grounded = false;
        }

        // if the enemy is grounded, then its velocity is reset to 0
        if (grounded)
        {
            velocity.y = 0f;
        }
        // otherwise, velocity is calculated using the following function
        else
        {
            velocity.y += 0.5f * gravity * Time.deltaTime * Time.deltaTime;
        }

        // enemy is moved by velocity vector
        controller.Move(velocity);
    }
}

```

Then, I created a base enemy script for the enemy's stats and attributes:

```
public class enemy : MonoBehaviour
{
    // reference to the environment script
    2 references
    private EnvironmentGenerator envScript;

    // the enemy's health
    3 references
    private int health = 100;

    // the enemy's speed
    2 references
    private float speed = 10f;

    // the enemy's dead or alive status
    1 reference
    private bool dead = false;

    // the enemy's damage stat
    3 references
    private int damage = 10;

    // the enemy's attack range stat
    1 reference
    private int attackRange = 15;

    // the enemy's follow range stat
    1 reference
    private int followRange = 75;

    // the enemy's attack delay stat
    1 reference
    private int delay;

    private void Start()
    {
        // finds the environment generator script, so that functions can be used such as remove enemy
        envScript = GameObject.FindGameObjectWithTag("Environment Generator").GetComponent<EnvironmentGenerator>();
    }
}
```

```

#region GETTERS AND SETTERS for the enemy's attributes
2 references
public int getHealth()
{
    return health;
}

0 references
public void setHealth(int _health)
{
    health = _health;
}

3 references
public float getSpeed()
{
    return speed;
}

2 references
public void setSpeed(float _speed)
{
    speed = _speed;
}

1 reference
public int getDamage()
{
    return damage;
}

0 references
public void setDamage(int _damage)
{
    damage = _damage;
}

0 references
public int getAttackRange()
{
    return attackRange;
}

1 reference
public int getFollowRange()
{
    return followRange;
}

0 references
public int getDelay()
{
    return delay;
}
#endifregion

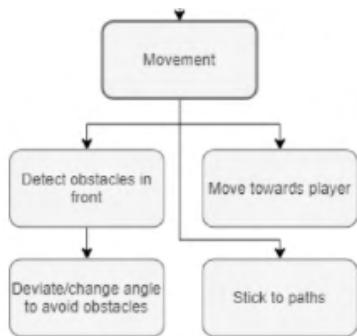
```

These methods and attributes can now be used by other classes such as enemy movement and enemy attack to manipulate the enemy as an entity.

Enemy movement

This class relates to the enemy movement class from the design section.

[Link to design section:](#)



Enemy movement involves 4 things;

- the enemy must be able to move forwards,
- the enemy must be able to move in random directions,
- the enemy must be able to change direction to avoid obstacles,
- the enemy must be able to move towards the player when in range.

```

public class enemyMovement : enemy
{
    // reference to the enemy script
    3 references
    private enemy enemy;

    // reference to the character controller component
    2 references
    private CharacterController controller;

    // reference to the player
    4 references
    private GameObject Player;

    // flag for whether the enemy should move towards the player or not
    4 references
    private bool moveTowardsPlayer = false;

    // flag for whether the enemy should stop moving towards the player
    5 references
    private bool stop = false;

    // counter for walking before changing direction
    7 references
    private int moveCounter = 0;

    // flag for whether this is the first time the enemy is within range
    // in which case they should instantly look at the player initially
    3 references
    private bool initialFindPlayer = true;

    // mask to only detect environment objects
    1 reference
    public LayerMask environmentObjects;

    // mask to only detect other enemys
    1 reference
    public LayerMask enemies;
}

```

```

private void Start()
{
    // gets the base enemy script
    enemy = GetComponent<enemy>();
    // finds the player gameobject
    Player = GameObject.FindWithTag("Player");
    // gets the character controller component of the enemy
    controller = GetComponent<CharacterController>();
}

private void Update()
{
    // if the enemy is close enough to the player, i.e. the distance is less than the enemy's follow range
    // and if the enemy is not too close (the enemy should not keep walking towards the player if its right next to the player)
    // then the enemy should move towards the player and should not stop
    // otherwise, the enemy should not move towards the player
    if ( (Vector3.Distance(Player.GetComponent<Transform>().position, this.transform.position) < enemy.getFollowRange()) )
    {
        if ( (Vector3.Distance(Player.GetComponent<Transform>().position, this.transform.position) < 25) )
        {
            stop = true;
        }
        else
        {
            moveTowardsPlayer = true;
            stop = false;
        }
    }
    else
    {
        moveTowardsPlayer = false;
        initialFindPlayer = true;
        stop = false;
    }
}

```

The program constantly evaluates the distance between the player and enemy, and if this distance is within the enemy's range, then the moveTowardsPlayer bool is set accordingly. This dictates how the enemy will move, i.e. randomly or towards the player. If the enemy is extremely close to the player, then it should not keep moving towards the player, it should stop so the stop bool is set accordingly.

```

// if the enemy should move towards the player,
if (moveTowardsPlayer && !stop)
{
    // it should first look at the player
    if (initialFindPlayer == true)
    {
        initialFindPlayer = false;
        lookAtPlayer();
    }

    // then the move function is called,
    // which moves the enemy forwards,
    // dodging any obstacles in the way
    move();

    // if the enemy has moved forwards in that one direction,
    // for 120 frames, then they should look at the player again
    if (moveCounter % 120 == 0 || moveCounter == 0)
    {
        moveCounter = 0;
        lookAtPlayer();
    }
}

// if the enemy is not to move towards the player
// then it should move randomly
else if (!moveTowardsPlayer && !stop)
{
    // the move function is called,
    // which moves the enemy forwards,
    // dodging any obstacles in the way
    move();

    // if the enemy has moves forwards in that one direction
    // for 600 frames, then they should change direction
    if (moveCounter % 600 == 0 || moveCounter == 0)
    {
        // enemy looks at random direction
        moveCounter = 0;
        lookAtRandom();
    }
}
}

```

```

public void lookAtPlayer()
{
    // first the enemy has to look towards the player
    // the whole enemy rotates to look at the player
    this.transform.LookAt(Player.transform);
    // only the y value of this rotation is desired
    float yRot = this.transform.eulerAngles.y;
    // then the rotation of the enemy is set to the desired y rotation
    // x and z components are reset to 0 so that enemy stands upright
    this.transform.eulerAngles = new Vector3 (0, yRot, 0);
}

public void lookAtRandom()
{
    // random y rotation is chosen,
    // and applied to the enemy's rotation
    int yRot = Random.Range(0, 361);
    this.transform.eulerAngles = new Vector3 (0, yRot, 0);
}

```

Then, depending on the value of the stop and moveTowardsPlayer bool variables, the enemy either moves randomly or towards the player.

If the enemy is to move towards the player, then it rotates to face the player, and moves forwards in that direction (avoiding obstacles in its way). After 120 frames, it re-evaluates its direction and rotates to face the player again (since the player has likely moved). This repeats until the player is dead or not in range of the enemy anymore.

For the random movement, the enemy chooses a random direction to face by choosing a random y-rotation value between 1, and 361 (tail value is exclusive). It then moves forwards in that direction (avoiding obstacles in its way) for 600 frames. Afterwards, if it is still not in range of the player, then a new random direction is chosen, and this repeats until the player is dead or is in range of the enemy, in which case the enemy will move towards the player.

```

public void move()
{
    // a ray is cast from the enemy roughly 18 degrees at an angle of depression from the horizontal
    Ray lineSight = new Ray(this.transform.position, this.transform.forward * 3 - this.transform.up);
    RaycastHit lineSightHit;
    Debug.DrawRay(this.transform.position, (this.transform.forward * 3 - this.transform.up) * 15, Color.red);

    // if this ray hits an enemy or an environment object, then the enemy needs to act to avoid the obstacle
    if (Physics.Raycast(lineSight, out lineSightHit, 100f, environmentObjects))
    {
        if (Physics.Raycast(lineSight, out lineSightHit, 100f, enemies))
        {
            float currentX = this.transform.eulerAngles.x;
            float currentY = this.transform.eulerAngles.y;
            float currentZ = this.transform.eulerAngles.z;

            // the enemy rotates by -5 degrees to avoid the obstacle,
            // and this will repeat until the ray is no longer hitting an obstacle
            this.transform.eulerAngles = new Vector3(currentX, currentY - 5, currentZ);
        }
    }

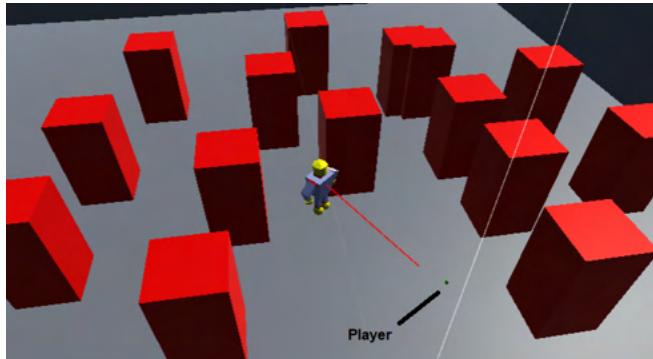
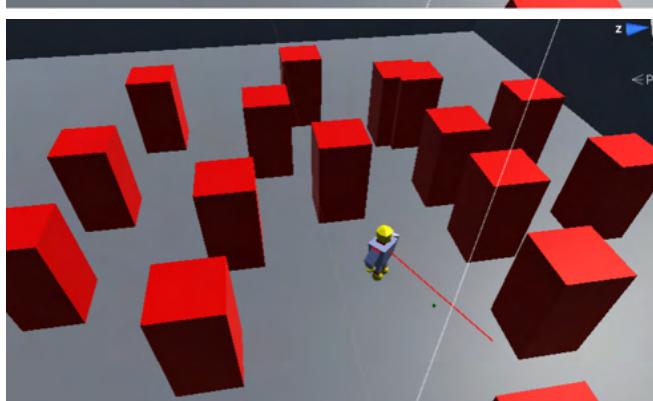
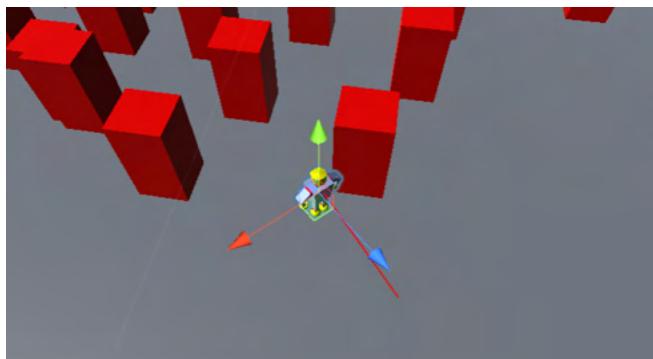
    // then, the enemy moves forward by its speed attribute
    controller.Move(transform.forward * enemy.getSpeed() * Time.deltaTime);
    // move counter is updated so that the main function can check if its time for the enemy to change direction
    moveCounter += 1;
}

```

The move function moves the enemy forwards by a factor of the enemy's speed.

The enemy, however, takes action to dodge obstacles in its way such as environment objects or other enemies. This is done using a ray, the ray is cast from the enemy, at an 18 degree angle of depression. If this ray collides with any environment objects or other enemies, then the enemy rotates -5 degrees per frame, i.e. left, until the ray is no longer colliding with any obstacles, then it continues moving forwards.

Testing

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
As the player, be within the enemy's range.	Enemy should move, generally towards the player, avoiding obstacles.	  ! [13:07:50] MOVE TOWARDS PLAYER NOW UnityEngine.Debug:Log (object)	Pass
As the player, be out of the enemy's range.	Enemy should move in random directions, avoiding obstacles/	 ! [13:06:07] MOVE RANDOMLY NOW UnityEngine.Debug:Log (object)	Pass

Iteration 2

Although all tests were passed, I wasn't happy with the way the enemy would avoid obstacles. At the moment, the enemy keeps rotating left until the obstacle is no longer in its path. However, this isn't very realistic, the enemy should rotate left or right, not just one direction every time.

Therefore, I made the enemy rotate left or right, and it would switch between these 2 options every 5 seconds. This was done using a boolean variable, `left`, and toggling it between true and false every 5 seconds using a coroutine.

```
private IEnumerator SwitchLeftRight()
{
    while (true)
    {
        left = !left;
        yield return new WaitForSeconds(5f);
    }
}
```

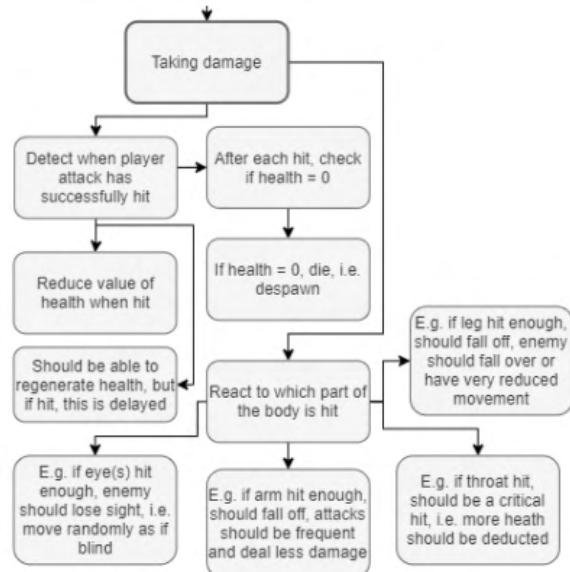
Then, depending on this variable, the enemy either rotates +5 or -5 degrees per frame when avoiding obstacles.

```
// if the enemy is to dodge left, then it rotates by -5 degrees,
// and this will repeat until the ray is no longer hitting an obstacle
if (left)
{
    this.transform.eulerAngles = new Vector3 (currentX, currentY - 5, currentZ);
}
// otherwise it rotates by +5 degrees, i.e. right, to try and avoid the obstacle
else
{
    this.transform.eulerAngles = new Vector3 (currentX, currentY + 5, currentZ);
}
```

All tests continued to pass with these minor changes.

Enemy take damage

This class relates to the enemy take damage class from the design section.



The enemy should take damage and react differently depending on where they are hit.

Therefore, I have assigned different weak spot levels to the enemy prefab. Red spots on the enemy are the weakest, therefore they are assigned the Weak3 tag, yellow is assigned "Weak2" and blue is "Weak1", i.e. the least weak parts of the enemy.

When taking damage, the enemy should firstly lose health and if its health is reduced to 0, then it should die. Therefore, I added the following methods to the base enemy class to enable this:

```
// reduces enemy's health by damage amount
1 reference
public void takeDamage(int _damage)
{
    health -= damage;
}
```

```
1 reference
public void Die()
{
    // if this enemy is now dead,
    // the enemy count must be decreased
    envScript.removeEnemy();
    Destroy(this.gameObject);
}
```

```
0 references
private void Update()
{
    if (dead)
    {
        Die();
    }
}
```

The takeDamage method is called in the player attack class as this is where the enemy is actually damaged/hit.

Next, the enemy needs to take damage differently depending on which part of the enemy is hit. This can be done using a damage multiplier for the player, and by destroying the gameobject, i.e. body part hit according to some conditions.

I developed this as part of the player attack class as it was most appropriate, so when the player lands a successful hit, this code runs:

```
// accesses the body part gameobject of the enemy that was hit
GameObject enemyPartHit = attackPoint.collider.gameObject;
// accesses the base enemy script
enemy enemyHit = attackPoint.collider.transform.root.GetComponent<enemy>();

// damage multiplier set according to weak spot level
if (enemyPartHit.tag == "Weak1")
{
    damageMultiplier = 1;
}
else if (enemyPartHit.tag == "Weak2")
{
    damageMultiplier = 2;
}
else if (enemyPartHit.tag == "Weak3")
{
    damageMultiplier = 4;
}

// calls the damage method of the enemy,
// and uses the weapon's damage attribute, multiplied by the damage multiplier
// to reduce enemy health
enemyHit.takeDamage(player.getWeapon().getDamage() * damageMultiplier);
// player gains 5, multiplied by the multiplier, score for a hit
player.addScore(5 * damageMultiplier);

// if the enemy is overall weak enough (looking at its health)
// and a weakspot is hit, this part of the enemy is to be destroyed
// due to the hierarchical nature of the enemy prefab,
// some body parts of the enemy will also be destroyed if their parent is
if (enemyPartHit.tag == "Weak3" && enemyHit.getHealth() < 50)
{
    Destroy(enemyPartHit);
}
else if (enemyPartHit.tag == "Weak2" && enemyHit.getHealth() < 30)
{
    Destroy(enemyPartHit);
}
```

The multiplier means that more damage is inflicted if the weak spots are hit, and destroying parts of the enemy if they are weak enough means that the player receives a visual sense of progression when fighting an enemy. The way the enemy is designed, the hierarchical nature of the body parts of the enemy mean that, for example, if the shoulder is destroyed, all child gameobjects of the shoulder will also be destroyed, i.e. the rest of the arm and the hand, and this is more realistic, if the shoulder is cut off or destroyed, the whole arm should too.

Testing

To be able to test more easily, without having the weapon system developed yet, I developed the following class to be able to damage the enemy by simply clicking.

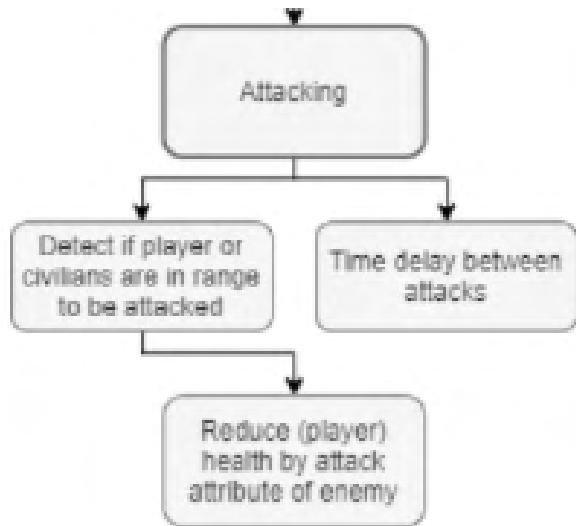
```
public class testDamage : enemy
{
    3 references
    enemy enemy;
    1 reference
    public int damage = 20;
    0 references
    void Update()
    {
        if ((Input.GetKeyDown("space")))
        {
            Debug.Log("enemy health: " + enemy.getHealth());
            enemy.takeDamage(damage);
            Debug.Log("enemy health: " + enemy.getHealth());
        }
    }
}
```

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Enemy starts with 50 health, attack with player for 30 damage. (Normal)	Enemy health is reduced to 20, enemy does not die.	 [00:25:04] enemy health: 50 UnityEngine.Debug:Log (object)  [00:25:04] enemy health: 20 UnityEngine.Debug:Log (object)	Pass
Enemy starts with 20 health, attack with player for 40 damage (Normal)	Enemy health is reduced to 0 and enemy dies after 5 secs. (despawns)	 [00:27:24] enemy health: 20 UnityEngine.Debug:Log (object)  [00:27:24] enemy health: -20 UnityEngine.Debug:Log (object)  [00:27:24] DIE now UnityEngine.Debug:Log (object)	Pass
Enemy starts with 10 health, attack with player for 10 damage (Boundary)	Enemy health is reduced to 0 and enemy dies after 5 secs. (despawns)	 [00:28:02] enemy health: 10 UnityEngine.Debug:Log (object)  [00:28:02] enemy health: 0 UnityEngine.Debug:Log (object)  [00:28:02] DIE now UnityEngine.Debug:Log (object)	Pass

Enemy attack

This class relates to the enemy attack class from the design section.

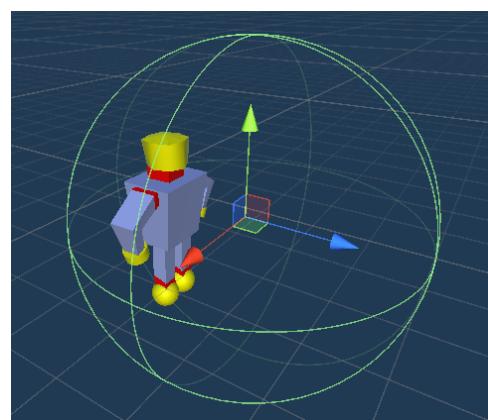
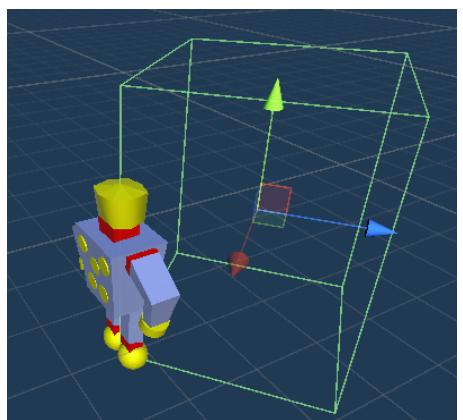
Link to design section:



The enemy attacking is a sequence of 2 stages, when the enemy is first in range to attack the player, the enemy initiates the attack, however there should be a time delay while this occurs before the player is actually hit. Only if the player is within range after the delay, will they be damaged.

This detection can be done using colliders with the trigger option chosen, so that they do not interfere with other colliders, but rather detect their presence when they interact.

For the 2 different detections, I set up the following colliders:



The first image shows the triangular prism collider that is used as an initial attack detection. The second image shows the second attack detection, i.e. if the player is within this region even

after the time delay, then they will be attacked and take damage.

```
public class hitDetect1 : enemy
{
    // bool for if the player is colliding
    // with this trigger collider
    3 references
    private bool hit1;

    // bool for if this collider has been initialised
    3 references
    private bool isActive = false;

    1 reference
    public bool getHit1()
    {
        // returns the value of hit1
        // if the collider is initialised
        // otherwise, false
        if (isActive)
        {
            return hit1;
        }
        else
        {
            return false;
        }
    }

    0 references
    private void OnTriggerEnter(Collider other)
    {
        // if this collider collides with the player
        // then hit1 is set to true
        if (other.tag == "Player")
        {
            isActive = true;
            Debug.Log("HIT 1");
            hit1 = true;
        }
    }

    0 references
    private void OnTriggerExit(Collider other)
    {
        // if the player Leaves this collider,
        // then hit1 is reset to false
        if (other.tag == "Player")
        {
            isActive = true;
            Debug.Log("HIT 1 OFF");
            hit1 = false;
        }
    }
}

public class hitDetect2 : enemy
{
    // bool for if the player is colliding
    // with this trigger collider
    3 references
    private bool hit2 = false;

    // bool for if this collider has been initialised
    3 references
    private bool isActive = false;

    1 reference
    public bool getHit2()
    {
        // returns the value of hit2
        // if the collider is initialised
        // otherwise, false
        if (isActive)
        {
            return hit2;
        }
        else
        {
            return false;
        }
    }

    0 references
    private void OnTriggerEnter(Collider other)
    {
        // if this collider collides with the player
        // then hit2 is set to true
        if (other.tag == "Player")
        {
            isActive = true;
            Debug.Log("HIT 2");
            hit2 = true;
        }
    }

    0 references
    private void OnTriggerExit(Collider other)
    {
        // if the player Leaves this collider,
        // then hit2 is reset to false
        if (other.tag == "Player")
        {
            isActive = true;
            Debug.Log("HIT 2 OFF");
            hit2 = false;
        }
    }
}
```

These scripts are attached to the collider gameobjects and alter bool values hit1 and hit2 according to collisions between the player collider and these colliders. These boolean values can be used to decide whether the player should be hit or not in the enemy attack class.

```
public class enemyAttack : enemy
{
    // reference to the base enemy script
    6 references
    private enemy enemy;

    // reference to the base player script
    2 references
    private player player;

    // reference to the hit1 and hit2 bools
    // used to decide whether player should be attacked
    2 references
    private bool hit1;
    2 references
    private bool hit2;

    // reference to the collider gameobjects
    2 references
    private GameObject hd1;
    2 references
    private GameObject hd2;

    // multiplier to slow the enemy when attacking
    // must be less than 1
    2 references
    private float slowFactor = 0.75f;

    // bool for whether the enemy is currently slowed
    4 references
    private bool slowed = false;

    0 references
    private void Start()
    {
        // accessing the base enemy and player scripts
        enemy = GetComponent<enemy>();
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<player>();

        // accessing the collider gameobjects
        hd1 = GameObject.FindGameObjectWithTag("HITDETECT1");
        hd2 = GameObject.FindGameObjectWithTag("HITDETECT2");

        StartCoroutine(attackLoop());
    }
}
```

```

private IEnumerator attackLoop()
{
    // constant loop which loops every fixed frame
    while (true)
    {
        // gets the current status of the hit1 bool
        hit1 = hd1.GetComponent<hitDetect1>().getHit1();
        //Debug.Log("HIT 1: " + hit1);

        // if the player is in range of hit1,
        // then the enemy is slowed and the time wait begins
        if (hit1)
        {
            if (!slowed)
            {
                enemy.setSpeed(enemy.getSpeed() * slowFactor);
                slowed = true;
            }

            yield return new WaitForSeconds(1f);
            Debug.Log("WAITED FOR 1 SEC");
            yield return new WaitForSeconds(1f);
            Debug.Log("WAITED FOR 2 SECS");
            yield return new WaitForSeconds(1f);
            Debug.Log("WAITED FOR 3 SECS");

            // gets the current status of the hit2 bool
            hit2 = hd2.GetComponent<hitDetect2>().getHit2();
            //Debug.Log("HIT 2: " + hit2);

            // if the player is still in range of hit2,
            // even after the time wait, then the player takes damage
            // and the enemy undergoes a time wait before it can attack again
            if (hit2)
            {
                Debug.Log("HIT PLAYER NOW");
                player.takeDamage(enemy.getDamage());
                Debug.Log("Wait for 5 secs now");
                yield return new WaitForSeconds(5f);
            }
        }

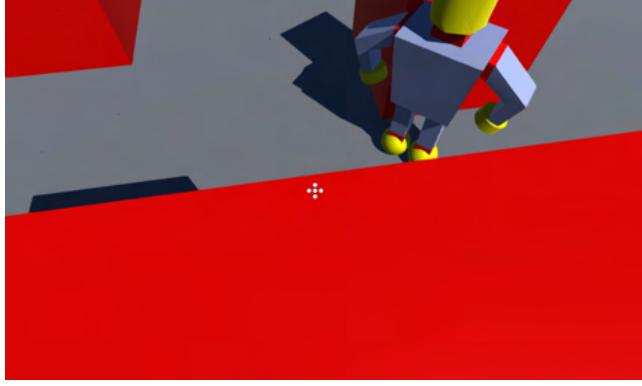
        // if the player is not within either collider region
        // then the enemy's speed is reset
        else
        {
            if (slowed)
            {
                enemy.setSpeed(enemy.getSpeed() / slowFactor);
                slowed = false;
            }
        }

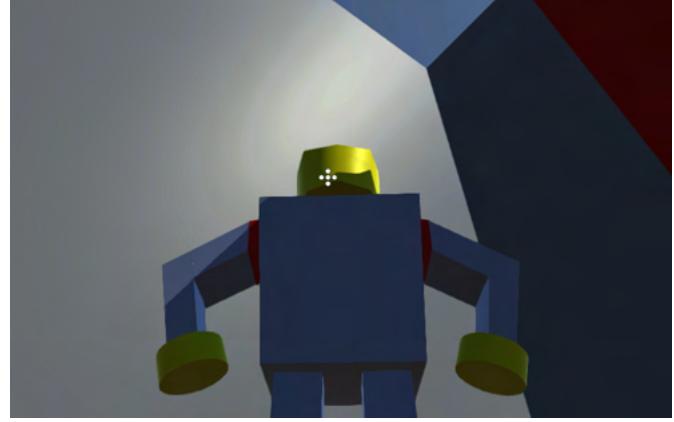
        yield return new WaitForFixedUpdate();
    }
}

```

This class references the collider game objects to be able to read hit1 and hit2 to decide whether the player should be attacked or not. If the player is in range of the enemy, i.e. hit1 is activated, then the enemy is slowed to allow the player to have a chance to escape, and a time wait of 5 seconds begins. The player can, of course, move freely at this time. If the player, after the time wait, is within the region of the second (spherical) collider, then hit2 is activated and the player is then hit and therefore takes damage. If the player is not within either of the enemy attack colliders and if the enemy is slowed, then its speed is reset to its default.

Testing

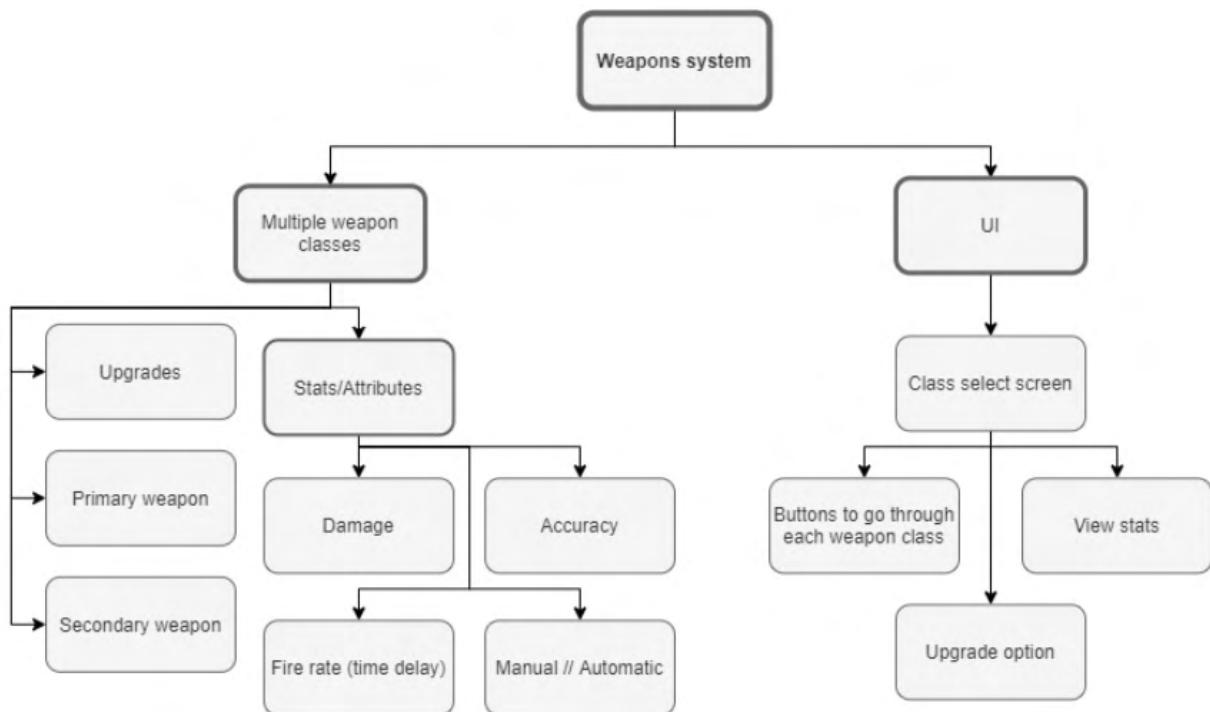
<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
As the player, stay out of the enemy range (Normal)	Enemy does not attack and the player does not take damage.	 No damage was taken (no damage logs)	Pass
As the player, be horizontally in range of the enemy (i.e. in the triangle) but be much higher than the enemy's range. (Normal)	Enemy does not attack and the player does not take damage.	 No damage was taken (no damage logs)	Pass

<p>As the player, be horizontally and vertically in range of the enemy, but within the 3 second time period, move out of even the larger hit zone / range. (Normal)</p>	<p>Enemy attempts to attack the player, however, the player does not take damage.</p>	 <p>No damage was taken (no damage logs)</p>	<p>Pass</p>
<p>As the player be horizontally and vertically in range of the enemy, then during the 3 second time period, move out of the initial hit zone but stay within the second, larger zone. (Normal)</p>	<p>Enemy attempts to attack the player and succeeds, i.e. the player takes damage.</p>	 <div style="background-color: black; color: white; padding: 5px;"> <ul style="list-style-type: none"> ! [09:22:55] WAITED FOR 1 SEC UnityEngine.Debug:Log (object) ! [09:22:56] WAITED FOR 2 SECS UnityEngine.Debug:Log (object) ! [09:22:49] WAITED FOR 3 SECS UnityEngine.Debug:Log (object) ! [09:22:49] HIT PLAYER NOW UnityEngine.Debug:Log (object) ! [09:22:41] HEALTH: 90 UnityEngine.Debug:Log (object) ! [09:22:49] Wait for 5 secs now UnityEngine.Debug:Log (object) </div>	<p>Pass</p>

Weapons

This section meets requirement(s): 4.2 and relates to all of 'Weapons' from the design section.

Link to design section:



The weapons will all instantiate from a base weapon script which has the required methods and attributes, e.g. fire rate, damage, fire delay, fire() method, etc. Therefore, I started by developing this script:

```

public class Weapon : MonoBehaviour
{
    #region attributes
    // weapon damage attribute
    2 references
    private int damage;

    // weapon fire range attribute
    2 references
    private int range;

    // weapon fire delay attribute
    3 references
    private float delay;

    // bool flag for whether weapon is in a state,
    // where it can fire or not (during delay it cant)
    4 references
    private bool canFire = true;

    // the projectile, e.g. bullet, shot by this weapon
    5 references
    private GameObject projectile;

    // position which the projectile is fired from
    3 references
    private Transform projectilePos;

    #endregion

    #region methods
    // GETTERS AND SETTERS
    0 references
    public void setDamage(int _damage)
    {
        damage = _damage;
    }

    0 references
    public void setRange(int _range)
    {
        range = _range;
    }

    0 references
    public void setDelay(float _delay)
    {
        delay = _delay;
    }

    0 references
    public int getDamage()
    {
        return damage;
    }

    0 references
    public float getDelay()
    {
        return delay;
    }

    0 references
    public int getRange()
    {
        return range;
    }

    0 references
    public bool getCanFire()
    {
        return canFire;
    }

    0 references
    public GameObject getProjectile()
    {
        return projectile;
    }

    0 references
    public void setProjectile(GameObject _projectile)
    {
        projectile = _projectile;
    }

    0 references
    public Transform getProjectilePos()
    {
        return projectilePos;
    }

    0 references
    public void setProjectilePos(Transform _projectilePos)
    {
        projectilePos = _projectilePos;
    }
}

```

```

// method to fire the weapon
// alters can fire bool flag, spawns and fires projectile (if needed)
0 references
public void fire()
{
    Debug.Log("FIRE!");
    canFire = false;

    // projects the projectile
    projectile = Instantiate(projectile, projectilePos.position, Quaternion.identity);
    projectile.GetComponent().AddForce(GameObject.FindWithTag("MainCamera").transform.forward * 10000);
}

0 references
private void Start()
{
    // starts the coroutine initially
    StartCoroutine(FireDelay());
}

1 reference
private IEnumerator FireDelay()
{
    // loops imitating the fixed update method,
    // loops at fixed intervals while the game runs
    while (true)
    {
        // if the weapon has been fired,
        // canFire will have been set to false,
        // it should be reset to true (able to fire again),
        // after waiting the amount of time the fire delay is
        if (!canFire)
        {
            yield return new WaitForSeconds(delay);
            canFire = true;
        }

        yield return new WaitForFixedUpdate();
    }
}

#endregion
}

```

As well as the main attributes, getters and setters, this class consists of 2 more methods. fire() takes care of what should happen and what the weapon does when it is fired. It should spawn and shoot the required projectile, and also start a timer for the weapon to wait before being able to fire again. This second requirement is handled using a boolean flag which indicates the weapons state at any time, it can either fire or not fire. The fire() method sets the canFire flag to false, and the FireDelay() coroutine waits for an amount of time equal to the fire delay of the weapon, before resetting the canFire boolean flag to true.

Sword

The sword does not consist of any projectiles, it must simply swing (using an animation) and deal damage. To instantiate it with the required values for its attributes, and to set it as the player's chosen weapon, I used the following script:

```
public class setSword : MonoBehaviour
{
    // reference to the base weapon script
    7 references
    private Weapon weaponBase;

    // reference to the base player script
    3 references
    private player player;

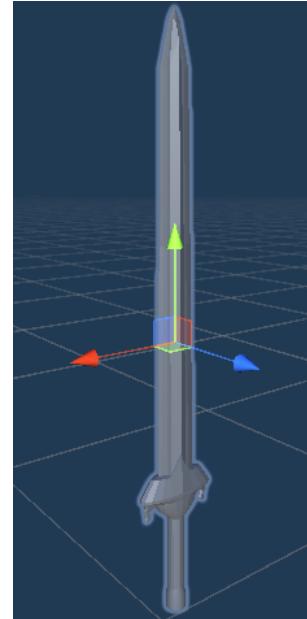
    // when the script is initially run,
    // the values should be set
    0 references
    private void Awake()
    {
        setter();
    }

    // when the script is re-enabled,
    // the values should be set
    0 references
    private void OnEnable()
    {
        setter();
    }

    2 references
    private void setter()
    {
        // gets the base weapon script
        weaponBase = GetComponent<Weapon>();
        // gets the base player script
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<player>();

        // sets attribute values according to what they should be for a sword
        weaponBase.setDamage(30);
        weaponBase.setDelay(1.5f);
        weaponBase.setRange(20);
        weaponBase.setProjectile(null);
        weaponBase.setProjectilePos(null);

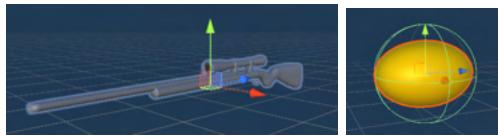
        // sets this weapon script as the player's chosen weapon
        player.setWeapon(weaponBase);
    }
}
```



The values are set when the script is first run, or when it is re-enabled after switching from the other (sniper) weapon. The values are assigned accordingly, and the player is assigned this weapon.

Sniper

The sniper will work in a very similar way to the sword, except that a projectile has to be passed as a parameter so that the bullet can be fired. Of course, the values used for the sniper will be very different to the sword.



```
public class setSniper : MonoBehaviour
{
    // reference to the base weapon script
    7 references
    private Weapon weaponBase;

    // reference to the base player script
    3 references
    private player player;

    // reference to the bullet projectile
    1 reference
    public GameObject projectile;

    // reference to the gun tip,
    // from where the bullet will be spawned and shot
    1 reference
    public GameObject gunTip;

    // when the script is initially run,
    // the values should be set
    0 references
    private void Awake()
    {
        setter();
    }

    // when the script is re-enabled,
    // the values should be set
    0 references
    private void onEnable()
    {
        setter();
    }

    2 references
    private void setter()
    {
        // gets the base weapon script
        weaponBase = GetComponent<Weapon>();
        // gets the base player script
        player = GameObject.FindWithTag("Player").GetComponent<player>();

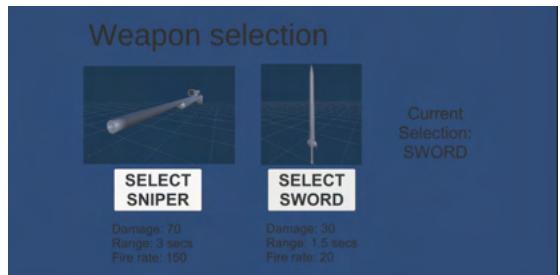
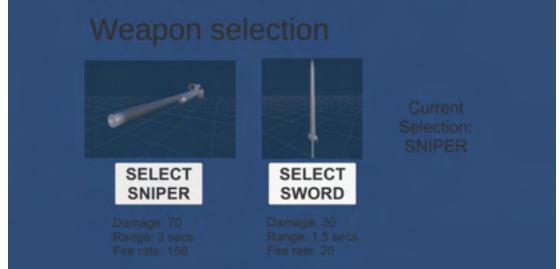
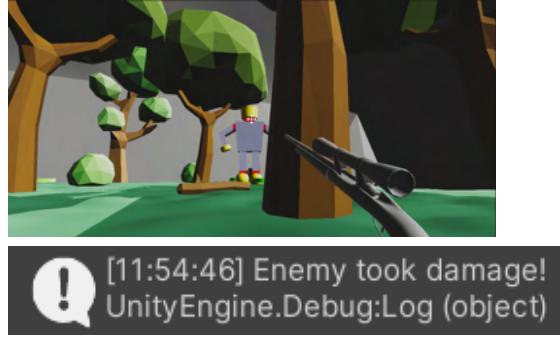
        // sets attribute values according to what they should be for a sniper
        weaponBase.setDamage(70);
        weaponBase.setDelay(3f);
        weaponBase.setRange(150);
        weaponBase.setProjectile(projectile);
        weaponBase.setProjectilePos(gunTip.transform);

        // sets this weapon script as the player's chosen weapon
        player.setWeapon(weaponBase);
    }
}
```

The values are set when the script is first run, or when it is re-enabled after switching from the other (sword) weapon. The values are assigned accordingly, and the player is assigned this weapon.

Testing

Note: first and third tests, switching and selecting weapon tests conducted at a later date after the weapons menu completed.

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Equip sword	Sword is shown to be the selected weapon		Pass
Attack an enemy with sword	Enemy takes damage equal to the damage attribute of the sword		Pass
Switch to equip sniper	Sniper is shown to be the selected weapon		Pass
Attack an enemy with sniper	Enemy takes damage equal to the damage attribute of the sniper		Pass

Progress Review 3

Since the last progress review, the enemies and weapons have been developed completely, with the exception of different types of enemies and boss enemies. **This fulfils requirements 6, 7.1, 7.2, 7.3 and 7.4.**

Overall, the clients are really happy with the progress made and state of the game so far, having seen the current prototype. They are also happy with the rate of progress. The main feedback they provided was “We want you to keep up the great work, at this rate, something successful can be achieved”.

Enemies can be considered the main objective of the game, as the player must defeat all the enemies in a level to be able to progress to the next one. At this point, the enemies have been developed as much as I intend to, they have all the main functionalities of being able to; move, attack, take damage and avoid obstacles (i.e. navigate intelligently). These have all been tested and work well, as intended. Some enemy features have been left out, the main 2 being lack of variation between enemies and no boss enemies. As enemies had to be modelled by myself (due to required combat functionalities), it was difficult to create more than one enemy. With a very strict time limit and lack of 3D modelling skills, it would have been a waste of time to create many different enemies as it is not an essential feature of the game, the game is still fun regardless and my clients agreed. A boss enemy, again, is not essential but would provide the player more excitement and fulfilment, making the game more fun. However, due to a lack of time, this feature was not implemented. Overall, the clients did not mind as these are not essential features, and are therefore ones that can be implemented at a later date.

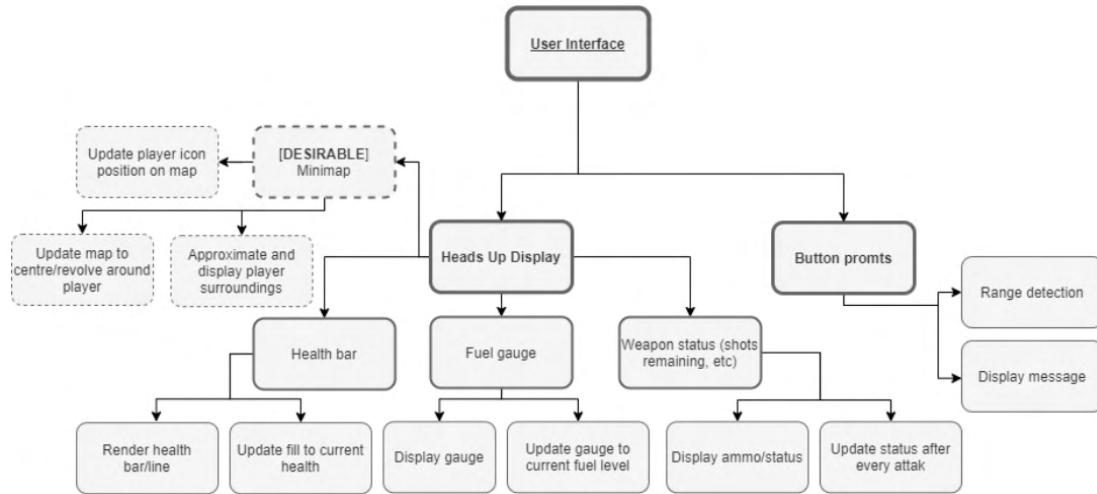
2 weapons have been implemented, the sword and the sniper. Weapons are essential to the game as they actually allow the player to fight the enemies. Having more than one weapon, while not essential, keeps the game from being repetitive and boring. The 2 weapons have all their required functionalities, they have damage, range, fire delay, etc. A projectile system has also been implemented for projectile based weapons such as the sniper. All these features have been tested and work as intended. One thing that can be considered to be left out is the third weapon, a pistol. The plan was to originally have 3 weapons. However, due to the nature of the implementation, the pistol would be too similar to the sniper to be worth spending time implementing. More weapons can always be implemented in the future, and the client was happy with this.

Overall, at this stage, roughly 90% of the development has been completed. At this point, the rate of progress has sped up considerably which is good and should allow me to complete the required features in time. I will aim to keep at this rate of development and the client was happy for me to keep moving forwards with the development. This review has been crucial in helping me acknowledge that things are going well and that I need to keep working at this level.

UI

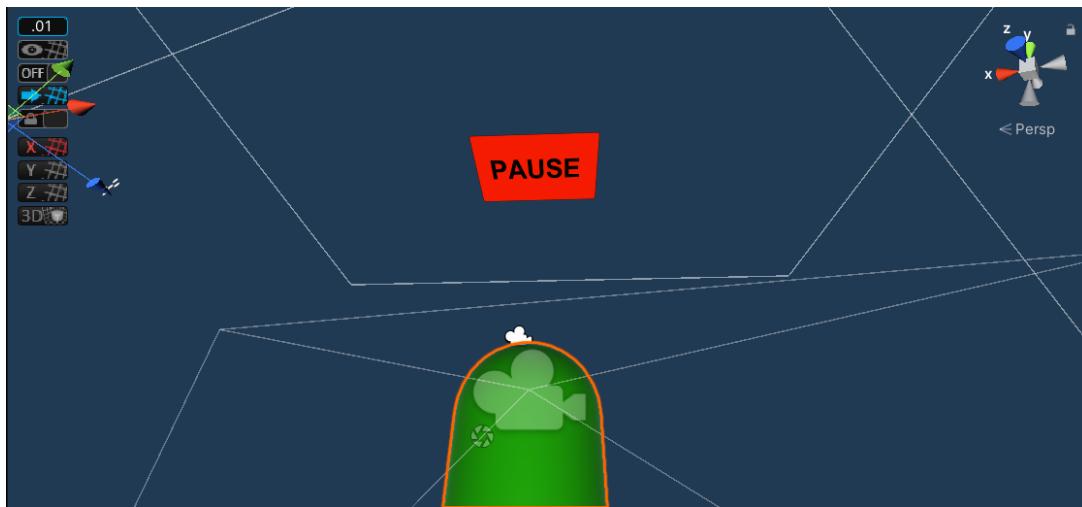
This section meets requirement(s): 5 and relates to all of 'UI' from the design section.

Link to design section:

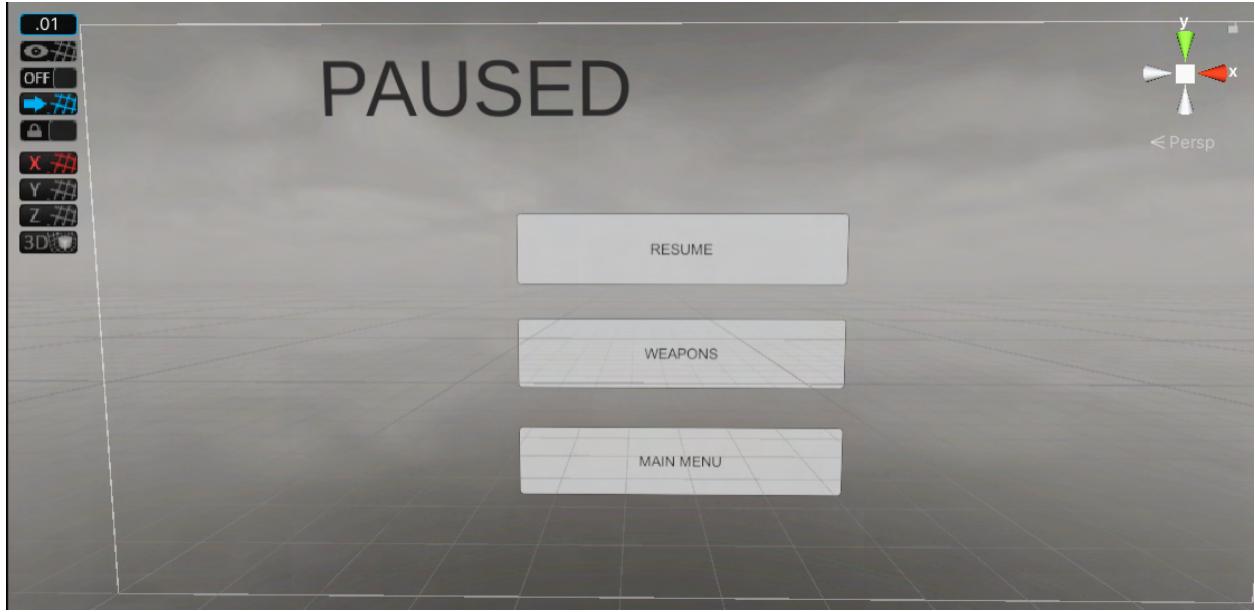


Pause menu

The pause menu should be accessed via a pause button which can be placed directly above the player and can follow them so that the button can be used by the player whenever they want to pause. I started by creating this button and attaching it as a child to the player prefab:



The pause button needs to show the player the pause menu when clicked on, so I created a pause UI canvas to be displayed when the game is paused.



The pause menu consists of the pause title, and 3 buttons; resume (returns the player to the game), weapons (takes the player to the weapons menu) and main menu (returns the player to the main menu).

To make these buttons work, as well as the player's pause button, I developed a script with the required functions and assigned them to the corresponding buttons:

```
public class Paused : MonoBehaviour
{
    // reference to the main camera
    // which is used in gameplay
    3 references
    private GameObject MainCam;
    // reference to the pause camera
    // which is used when paused
    3 references
    private GameObject PauseCam;

    // reference to the pause canvas
    // which is displayed when paused
    3 references
    private GameObject PauseCanvas;
    // reference to the weapons canvas
    // for when the player is choosing weapon
    3 references
    private GameObject WeaponsCanvas;

    0 references
    private void Start()
    {
        // gets the main camera
        MainCam = GameObject.FindGameObjectWithTag("MainCamera");
        // gets the pause camera
        PauseCam = GameObject.FindGameObjectWithTag("Pause Camera");

        // gets the pause canvas
        PauseCanvas = GameObject.FindGameObjectWithTag("Pause Canvas");
        // gets the weapons canvas
        WeaponsCanvas = GameObject.FindGameObjectWithTag("Weapons Canvas");
    }
}
```

```

0 references
public void Pause()
{
    // pausing should stop all the movement of the player and enemies etc.
    // this is done by reducing the time scale to 0
    Time.timeScale = 0;

    // main camera disabled
    // pause camera enabled instead
    MainCam.SetActive(false);
    PauseCam.SetActive(true);

    // pause canvas shown
    // weapons canvas hidden for now
    PauseCanvas.SetActive(true);
    WeaponsCanvas.SetActive(false);
}

0 references
public void Weapons()
{
    // weapons canvas shown
    // pause canvas hidden
    PauseCanvas.SetActive(false);
    WeaponsCanvas.SetActive(true);
}

0 references
public void MainMenu()
{
    // finds the main menu scene and sets it as the current scene
    Scene mainMenuScene = SceneManager.GetSceneByName("Menu");
    SceneManager.LoadScene("Menu", LoadSceneMode.Single);
    SceneManager.SetActiveScene(mainMenuScene);
}

0 references
public void Unpause()
{
    // time scale reset to 1 to unpause
    Time.timeScale = 1;

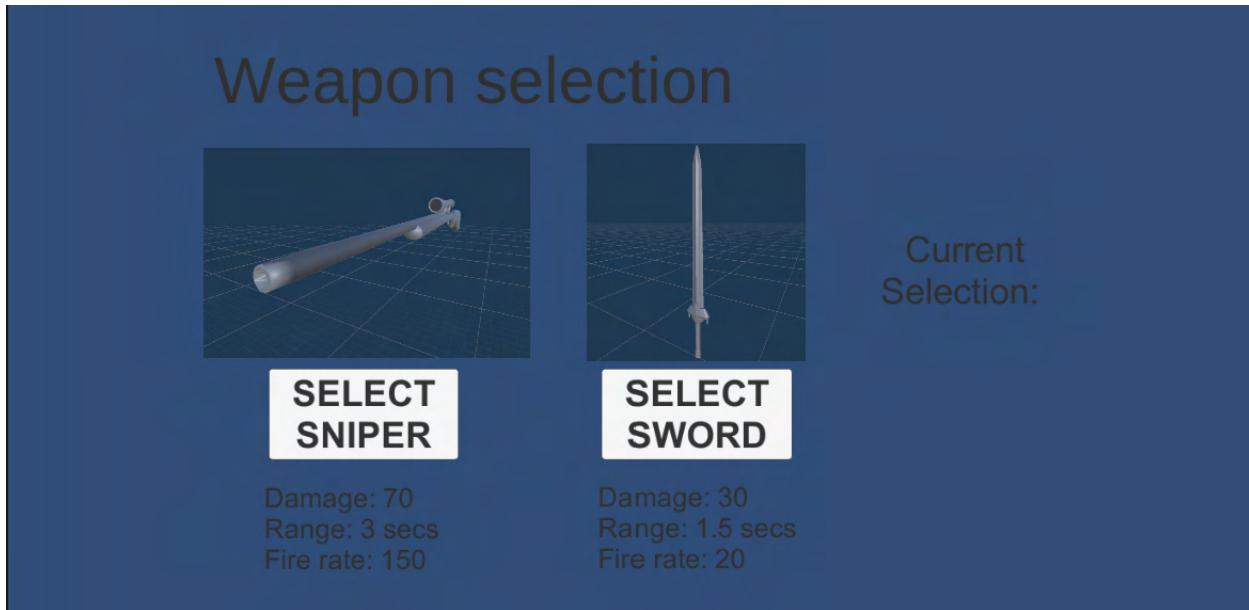
    // main camera re-enabled so player can play
    // pause camera disabled to hide pause UI
    MainCam.SetActive(true);
    PauseCam.SetActive(false);
}

```

The pause method pauses the game by setting the time scale to 0 and it also switches to the pause camera. The unpause method does the exact opposite of this. The weapons method switches the visible canvas to the weapons one. And the main menu method returns the player to the main menu by loading the correct scene for this.

Weapon selection

Weapon selection should be done via a weapons menu accessed via the pause menu. The pause menu can be swapped out for the weapons menu when required. Therefore, I initially created this weapons UI canvas:



The menu shows both weapons, their stats and a button to select them. It also shows some test telling the player what their currently selected weapon is. This will change when the player clicks on one of the buttons.

I then wrote a script which sets the player's weapon depending on the button they click, and also updates the current selection text:

```

public class weaponMenu : MonoBehaviour
{
    // reference to the player gameobject
    1 reference
    public GameObject playerObject;
    // reference to the base player script
    1 reference
    private player player;

    // reference to the weapon selected text
    1 reference
    public GameObject weaponSelection;
    // reference to the text component of the above object
    3 references
    private Text text;

    // reference to the sword weapon
    3 references
    public GameObject sword;
    // reference to the sniper weapon
    3 references
    public GameObject sniper;

    0 references
    private void Start()
    {
        // gets all the components
        text = weaponSelection.GetComponent<Text>();
        player = playerObject.GetComponent<player>();
        sniper = GameObject.FindWithTag("Sniper");
        sword = GameObject.FindWithTag("Sword");
    }

    0 references
    public void selectSniper()
    {
        // updates the text to show that sniper is selected
        text.text = "Current Selection: SNIPER";
        // activates the sniper and disables the sword
        sniper.SetActive(true);
        sword.SetActive(false);
    }

    0 references
    public void selectSword()
    {
        // updates the text to show that sniper is selected
        text.text = "Current Selection: SWORD";
        // activates the sword and disables the sniper
        sniper.SetActive(false);
        sword.SetActive(true);
    }
}

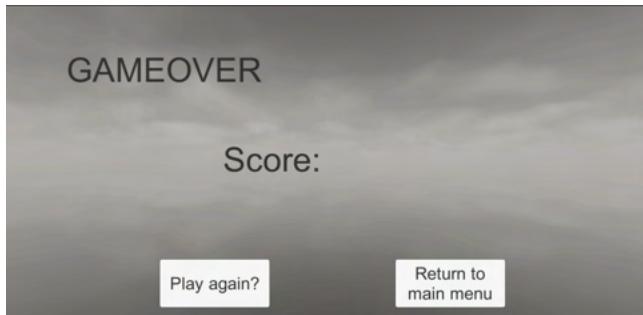
```

The select sword and sniper methods work in the same way; they update the text to show the relevant weapon is selected and they enable the relevant weapon and disable the other.

Game Over

When the player dies, if they have now lost all their lives, it is game over. At this point, a screen should appear with their stats such as score, and should be provided with 2 options; play again and exit to the main menu.

I started by creating this game over UI:



The return to main menu button can use the same script as the one from the pause menu as it contains a method to do the exact same thing as required. For the play again button, I created a script with the following method:

```
public class playAgain : MonoBehaviour
{
    // reference to the base player script
    4 references
    private player player;

    0 references
    private void Start()
    {
        // gets the player script
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<player>();
    }

    0 references
    public void PlayAgain()
    {
        // resets the player's score to 0
        player.addScore(-player.getScore());
        // calls the player respawn function
        player.Respawn();
    }
}
```

It simply resets the player's score and calls the player's respawn function. The respawn function simply restarts the environment generated (to create a new level) by calling the NewLevel method:

```
public void Respawn()
{
    envGen.NewLevel();
}
```

To display the score, I wrote the following script to update the text to the player's score:

```
public class score : MonoBehaviour
{
    // reference to the score text
    1 reference
    private Text scoreText;

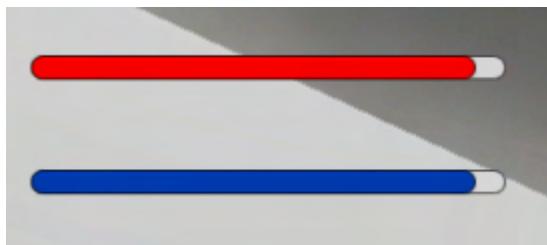
    // reference to the base player script
    2 references
    private player player;

    0 references
    private void Start()
    {
        // gets the player script
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<player>();
    }

    0 references
    public void showText()
    {
        // updates the text to show the player's score
        scoreText.text = "Score = " + player.getScore();
    }
}
```

Health & Fuel bars

The health and fuel bars will be 2 sliders which represent the player's health and fuel out of 100 at any time. I started by creating these 2 sliders:



Then, I created a script to control the value of each of these sliders to represent the player's health and fuel.

```

public class HealthFuel : MonoBehaviour
{
    // reference to the base player script
    3 references
    private Player player;

    // reference to the health slider
    2 references
    private Slider healthSlider;
    // reference to the fuel slider
    2 references
    private Slider fuelSlider;

    0 references
    private void Start()
    {
        // gets the player script
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();
        // gets the health slider
        healthSlider = GameObject.FindGameObjectWithTag("Health Slider").GetComponent<Slider>();
        // gets the fuel slider
        fuelSlider = GameObject.FindGameObjectWithTag("Fuel Slider").GetComponent<Slider>();

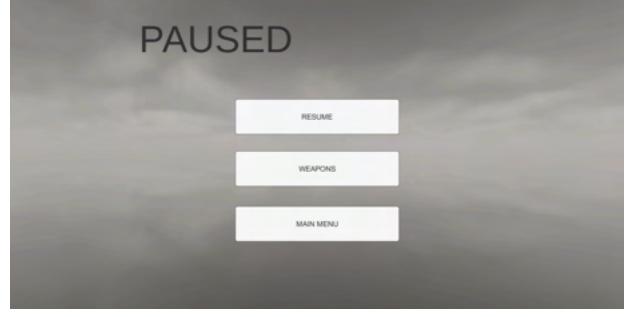
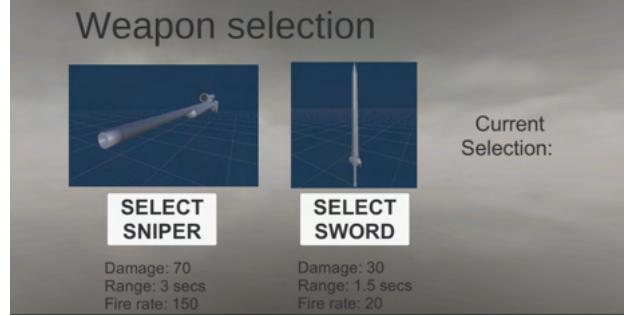
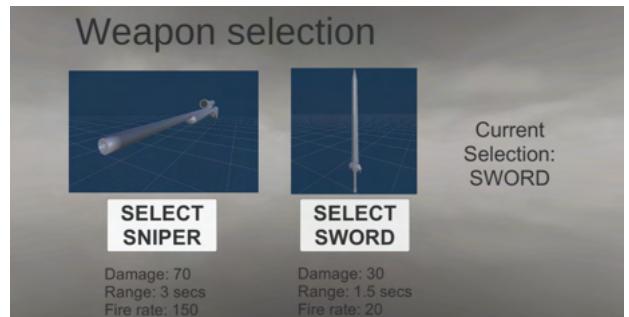
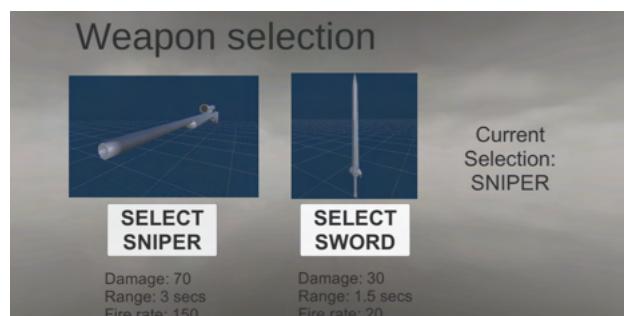
    }

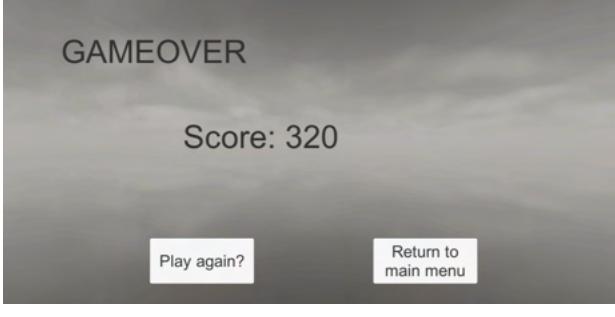
    0 references
    private void Update()
    {
        // updates both the health and fuel to current value
        healthSlider.value = player.getHealth();
        fuelSlider.value = player.getFuel();
    }
}

```

It simply sets the value of the sliders equal to the player's health and fuel attributes every frame.

Testing

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Press pause button	Game pauses and pause menu displayed		Pass
Press the weapons menu button	Game remains paused and weapons menu displayed		Pass
Equip the sword	Sword shown as the equipped weapon and is equipped		Pass
Equip the sniper	Sniper shown as the equipped weapon and is equipped		Pass

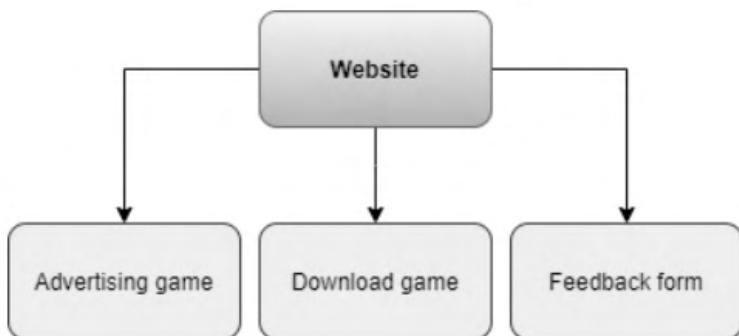
Press the back button	Game remains paused and returns to pause menu (weapons menu not shown anymore)		Pass
Press the exit to main menu button	Main menu loaded and displayed		Pass
During gameplay, use some fuel and lose some health	Both the health and fuel sliders should decrease but the respective amount		
Lose all lives (to trigger game over)	Game pauses and game over screen is displayed		Pass

Press play again button	Game generates a new level and player can play again		Pass
On the game over screen, press the exit to main menu button	Main menu loaded and displayed		Pass

Website

This section meets requirement(s): 9 and relates to all of 'Website' from the design section.

Link to design section:



The website will be coded using HTML, CSS and JavaScript, and also with PHP for form handling. The site will consist of 4 pages, a home page, an about page, a download page, and a feedback form page. I will also include a button on all pages that allow the user to toggle between light and dark modes/themes.

Home page

The home page should be a simple page with a title and 4 buttons to each of the pages, with the last button being the theme toggle button.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- Link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- Link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

    <body>
        <!-- page heading -->
        <h1 class=main>ATTACK ON THE UNKNOWN</h1>

        <!-- about page button -->
        <div class=main>
            <!-- Link to about page -->
            <a href="about.html">
                <button>
                    <!-- button label -->
                    ABOUT
                </button>
            </a>
        </div>

        <br>

        <!-- download page button -->
        <div class=main>
            <!-- Link to download page -->
            <a href="download.html">
                <button>
                    <!-- button label -->
                    DOWNLOAD
                </button>
            </a>
        </div>

        <br>
```

```

<!-- feedback form page button -->


This HTML page uses divs to separate the buttons and provide a background for them. The toggle theme button calls the swapTheme javascript function which will be documented after all the HTML pages. The page, by default, uses the darkAOTU style sheet which can be changed using the toggle theme button to the lightAOTU style sheet. Both these CSS files will also be documented after the HTML pages.



This is the result for this page:



The screenshot shows a dark-themed web page with a black header containing the text "ATTACK ON THE UNKNOWN". Below the header are four purple rectangular buttons, each with white text. The first button says "ABOUT", the second says "DOWNLOAD", the third says "FEEDBACK FORM", and the fourth button at the bottom says "Toggle theme". The "Toggle theme" button is slightly larger than the others and has a different font style.



Harsh Brahmbhatt  
Watford Boys Grammar School



5124  
17655  
220


```

About page

The about page should have 3 main information sections; information about what the game is about, a photo gallery of images from the game and some information about how the user can contact me/us.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- Link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- Link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

    <body>
        <!-- page heading -->
        <h1>ABOUT</h1>

        <!-- list of information -->
        <div class=region>
            <div class=section>
                <h2>What is Attack on the unknown?</h2>
                <ul>
                    <li>Attack on the Unknown is a cardboard VR mobile game</li>
                    <li>Defeat the giants with your sword or sniper!</li>
                    <li>Lots of biomes to explore!</li>
                    <li>Levels are randomly generated, the game will never end...</li>
                </ul>
            </div>
        </div>

        <br>

        <!-- group of images -->
        <div class=region>
            <div class=section>
                <h3>Gallery</h3>
                
                
                
                
                
                
            </div>
        </div>

        <br>
```

```

<!-- list of supported devices -->
<div class=region>
    <div class=section>
        <h3>Supported Devices</h3>
        <ul>
            <li>IOS mobile devices</li>
            <li>Android mobile devices</li>
            <li>Windows devices (alternate PC version of the game)</li>
        </ul>
    </div>
</div>

<br>

<!-- help and support information -->
<div class=region>
    <div class=section>
        <h3>Help and support</h3>
        <ul>
            <li>Use the feedback form to get in touch or provide any feedback</li>
            <li>The feedback form page can be found on the home page</li>
        </ul>
    </div>
</div>

<br><br>

<!-- toggle theme button -->
<div class=main>
    <!-- Link to js function called by button -->
    <button id="smallButton" onclick=swapTheme()>
        <!-- button label -->
        Toggle theme
    </button>
</div>

<br>

</body>

```

This page also uses divs to separate the 3 main sections clearly with a nice border. The toggle theme button calls the swapTheme javascript function which will be documented after all the HTML pages. The page, by default, uses the darkAOTU style sheet which can be changed using the toggle theme button to the lightAOTU style sheet. Both these CSS files will also be documented after the HTML pages.

This is the result:

ABOUT

What is Attack on the unknown?

- Attack on the Unknown is a cardboard VR mobile game
- Defeat the giants with your sword or sniper!
- Lots of biomes to explore!
- Levels are randomly generated, the game will never end...

Gallery

Supported Devices

- iOS mobile devices
- Android mobile devices
- Windows devices (alternate PC version of the game)

Help and support

- Use the feedback form to get in touch or provide any feedback
- The feedback form page can be found on the home page

Toggle theme

Download page

The download page should be very simple with only 2 buttons; one to start a download for the executable game file and one to again, toggle the theme.

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- Link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- Link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

    <body>
        <!-- page heading -->
        <h1>DOWNLOAD</h1>

        <br>

        <div class="main">
            <!-- downloadable resource location -->
            <a href="C:\Users\harsh\programmingProjects\Unity games\BUILDs\AOTU.exe" download>
                <!-- download button -->
                <button id=smallButton>
                    <!-- button label -->
                    Click here to download Attack on the Unknown
                </button>
            </a>
        </div>

        <br><br><br>

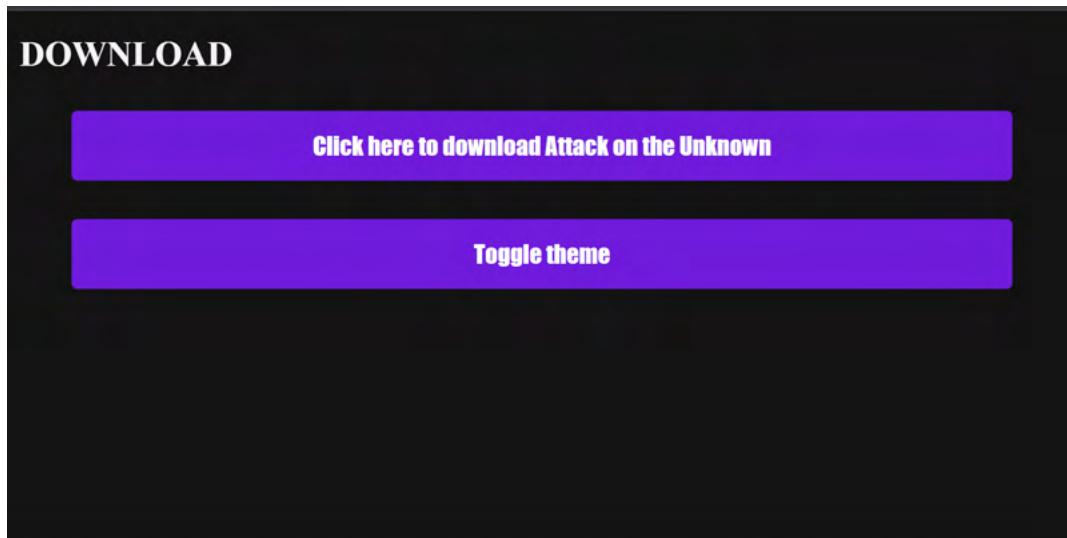
        <!-- toggle theme button -->
        <div class=main>
            <!-- link to js function called by button -->
            <button id="smallButton" onclick=swapTheme()>
                <!-- button label -->
                Toggle theme
            </button>
        </div>

        <br>

    </body>
</html>
```

This page also uses divs to separate the buttons and provide a border. See javascript file below for swapTheme() function and see both CSS files below for style sheets.

This is the result of this HTML page:



Feedback form page

The feedback form page should consist of the HTML form and the toggle theme button. The form should be as designed in the GUI designs, it will consist of a mix of text, radio, checkbox, etc. questions:

1. Enter your name (text type)
2. Enter your email address (email type)
3. Select age range (radio type)
 - a. 18-20
 - b. 20-25
 - c. 25-40
 - d. 40+
4. Choose the best features (checkbox type)
5. Choose the worst features (checkbox type)
 - a. Fast-paced gameplay
 - b. Random level generation
 - c. Weapons
 - d. Enemies
 - e. UI
 - f. Art
6. Any other feedback or suggestions? (text type)
7. Reset? (button type)
8. Submit (button type)

```

<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

    <body>
        <!-- page heading -->
        <h1>Feedback form</h1>

        <div class=region>
            <!-- form of questions (for feedback) -->
            <!-- sends the answers to actionPage.php to send responses to my email -->
            <form action="actionPage.php" method="post" style="padding-left: 20px;">

                <!-- text input -->
                <p><strong>Name:</strong></p>
                <input type="text" name="name" value="Enter name">

                <!-- email input -->
                <p><strong>Email address</strong></p>
                <input type="email" name="age" value="Enter email address">

                <br>
                <br>
                <br>

                <!-- radio input -->
                <p class=default><strong>Select your age range:</strong></p>
                <p class=default>18-20<input type="radio" value="18-20"></p>
                <p class=default>20-25<input type="radio" name="type" value="20-25"></p>
                <p class=default>25-40<input type="radio" name="type" value="25-40"></p>
                <p class=default>40+<input type="radio" name="type" value="40+"></p>

                <br>
                <br>

                <!-- radio input -->
                <p class=default><strong>Choose the features which you liked most about the game:</strong></p>
                <p class=default>Fast-paced gameplay<input type="radio" name="type" value="Fast-paced gameplay"></p>
                <p class=default>Random level generation<input type="radio" name="type" value="Random level generation"></p>
                <p class=default>Weapons<input type="radio" name="type" value="Weapons"></p>
                <p class=default>Enemies<input type="radio" name="type" value="Enemies"></p>
                <p class=default>UI<input type="radio" name="type" value="UI"></p>
                <p class=default>Art<input type="radio" name="type" value="Art"></p>

                <!-- radio input -->
                <p class=default><strong>Choose the features which you liked least about the game:</strong></p>
                <p class=default>Fast-paced gameplay<input type="radio" name="type" value="Fast-paced gameplay"></p>
                <p class=default>Random level generation<input type="radio" name="type" value="Random level generation"></p>
                <p class=default>Weapons<input type="radio" name="type" value="Weapons"></p>
                <p class=default>Enemies<input type="radio" name="type" value="Enemies"></p>
                <p class=default>UI<input type="radio" name="type" value="UI"></p>
                <p class=default>Art<input type="radio" name="type" value="Art"></p>

                <br>
                <br>
            </div>
        </form>
    </body>

```

```

<!-- textbox input -->
<p><strong>Any other feedback or suggestions?</strong></p>
<input type="text" name="comments" value="">

<br>
<br>
<br>
<br>
<br>
<br>

<!-- form reset button -->
<input type="reset" value="RESET">
<br><br>
<!-- form submit button -->
<input type="submit" value="SUBMIT">
<br><br>
</form>
</div>

<br><br><br>

<!-- toggle theme button -->
<div class=main>
    <!-- link to js function called by button -->
    <button id="smallButton" onclick=swapTheme()>
        <!-- button label -->
        Toggle theme
    </button>
</div>

```

This HTML page uses a form to be able to allow user inputs as answers for the questions. It uses the PHP file actionPage.php to handle the submission of the inputs/answers/responses. The actionPage.php PHP file is as below:

```

<?php

// gets all the form answers
$name = htmlspecialchars($_POST['name']);
$age = (int)$_POST['age'];
$bestFeature = htmlspecialchars($_POST['bestFeature']);
$worstFeature = htmlspecialchars($_POST['worstFeature']);
$otherFeedback = htmlspecialchars($_POST['otherFeedback']);

// concatenates the variables and produces a sentence
$feedback = "Name: $name \r\n Email: $email \r\n Age range: $age \r\n Best feature: $bestFeature \r\n Worst feature: $worstFeature \r\n Other feedback $otherFeedback .";

// headers for sending HTML emails
$headers[] = 'MIME-Version: 1.0';
$headers[] = 'Content-type: text/html; charset=iso-8859-1';

// sends an email with the information to me
mail($_POST["15brahmbhatt@students.watfordboys.org"], "Feedback from" $name, $feedback, implode("\r\n", $headers));

?>

```

The file uses the values filled from the form to generate a neat message with all of the information. This message is then sent to me via email using the mail() function.

The HTML page itself looks like this:

The screenshot shows a feedback form titled "Feedback form". It includes fields for Name and Email address, a section for selecting age range (18-20, 20-25, 25-40, 40+), a section for choosing liked features (Fast-paced gameplay, Random level generation, Weapons, Enemies, UI, Art), a section for choosing disliked features (Fast-paced gameplay, Random level generation, Weapons, Enemies, UI, Art), and a text area for "Any other feedback or suggestions?" with a "RESET" button below it.

Name:
Enter name

Email address:
Enter email address

Select your age range:

18-20 •
20-25 •
25-40 •
40+ •

Choose the feature which you liked most about the game:

Fast-paced gameplay •
Random level generation •
Weapons •
Enemies •
UI •
Art •

Choose the features which you liked least about the game:

Fast-paced gameplay •
Random level generation •
Weapons •
Enemies •
UI •
Art •

Any other feedback or suggestions?

RESET
SUBMIT

Toggle theme

The toggle theme features uses a simple javascript function:

```
function swapTheme()
{
    // gets the current CSS file name
    var currentTheme = document.getElementById("CSS").getAttribute("href");
    // depending on which file it is (light file or dark file)
    // the alternate is set
    // therefore theme swapped between light and dark or visa versa
    if (currentTheme == "lightAOTU.css")
    {
        var newTheme = "darkAOTU.css"
    }
    else
    {
        var newTheme = "lightAOTU.css"
    };

    // new/opposite CSS file set as style sheet for hmtl page
    document.getElementById("CSS").setAttribute("href", newTheme);
}
```

It simply takes the current CSS file in use, and depending on if the light or dark theme is currently in use, it switches the style sheet to the other file, toggling the theme between dark and light.

Styling

For the 2 themes, 2 CSS (style sheet) files are used:

darkAOTU.css

```
body{
    background-color: #rgb(20, 20, 20);
    padding-left: 10px;
    padding-right: 10px;
}

h1{
    font-size: 50px;
    color: #whitesmoke;
}

h2, h3{
    color: #white;
}

p, li{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    color: #rgb(206, 206, 206);
    padding-left: 5px;
    padding-right: 10px;
    padding-bottom: 5px;
}

button{
    background-color: #rgb(111, 28, 219);
    padding: 15px 32px;
    width: 90%;
    height: 200px;
    display: block;

    border: #rgb(111, 28, 219);
    border-radius: 8px;

    color: #white;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 35px;
    font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
    transition-duration: 0.4s;
}

button:hover{
    background-color: #rgb(20, 20, 20);
    color: #white;
    border: 5px solid #rgb(111, 28, 219);
}

input{
    background-color: #rgb(0, 0, 0);
    color: #white;
}

.region{
    background-color: #rgb(46, 0, 153);
    padding: 5px;
}

.section{
    background-color: #rgb(37, 37, 37);
    padding: 10px;
}

.main{
    text-align: center;
    font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
}

.default{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    color: #rgb(206, 206, 206);
}

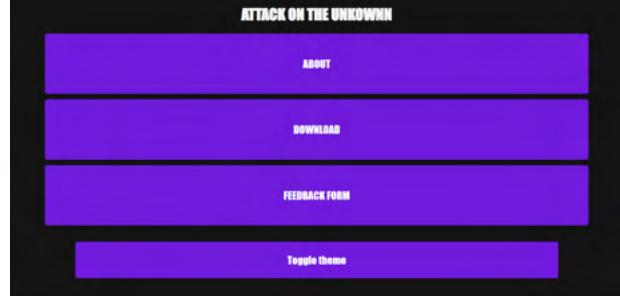
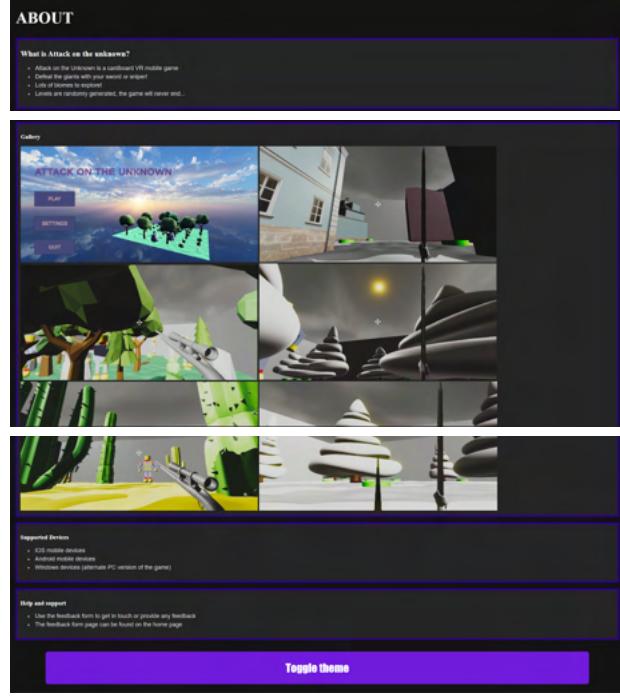
.row{
    background-color: #rgb(20, 20, 20);
}

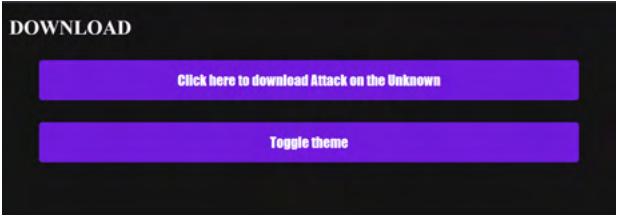
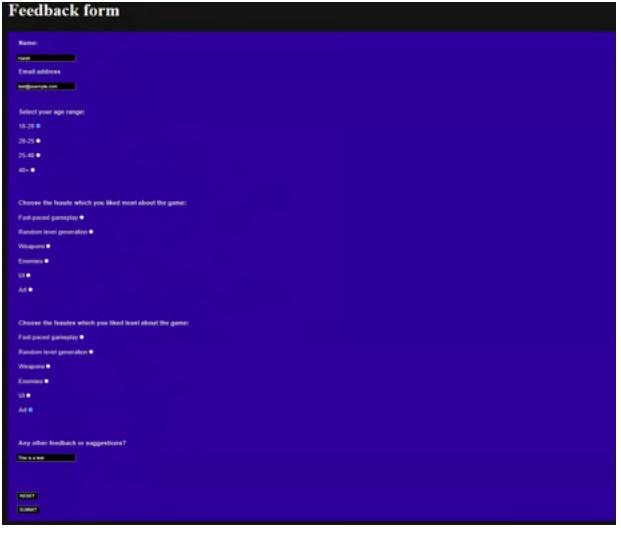
#smallButton{
    height: 100px;
}
```

lightAOTU.css

```
body{  
background-color:rgb(255, 255, 255);  
padding-left: 10px;  
padding-right: 10px;  
}  
  
h1{  
font-size: 50px;  
color: #rgb(0, 71, 165);  
}  
  
h2, h3{  
color: #white;  
}  
  
p, li{  
font-family: Arial, Helvetica, sans-serif;  
font-size: 18px;  
color:rgb(255, 255, 255);  
padding-left: 5px;  
padding-right: 10px;  
padding-bottom: 5px;  
}  
  
button{  
background-color:#rgb(0, 110, 255);  
padding: 15px 32px;  
width: 90%;  
height: 20px;  
display:block;  
  
border: #rgb(111, 28, 219);  
border-radius: 8px;  
  
color: #rgb(255, 255, 255);  
text-align: center;  
text-decoration: none;  
display: inline-block;  
font-size: 35px;  
font-family:Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;  
  
transition-duration: 0.4s;  
}  
  
button:hover{  
background-color:rgb(255, 255, 255) ;  
color: #rgb(0, 71, 165);  
  
border: 5px solid #rgb(0, 71, 165);  
}  
  
.region{  
background-color: #rgb(46, 0, 153);  
padding: 5px;  
}  
  
.section{  
background-color: #rgb(0, 110, 255);  
padding: 10px;  
}  
  
.main{  
text-align: center;  
font-family:Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;  
}  
  
.default{  
font-family: Arial, Helvetica, sans-serif;  
font-size: 18px;  
color:rgb(255, 255, 255);  
}  
  
.smallButton{  
height: 100px;  
}
```

Testing

<u>Input</u>	<u>Expected output</u>	<u>Actual output</u>	<u>Pass/ Fail</u>
Go to the homepage of the website	Home page should be displayed with a title and 3 buttons to navigate to other pages, and one more button to be able to toggle the theme.		Pass
Press the toggle theme button	Theme should switch between light and dark or vice versa.		Pass
Press the about button from the homepage	About page should be displayed with all the relevant information.		Pass

Press the download button from the homepage	Download page should be displayed with a button to download the game that works as intended.		Pass
Press the feedback form button from the homepage	Feedback form page should be displayed with a functional form that the user can fill in.		Pass
Fill out feedback form and submit	Answers should be emailed to 15brahmbhatt@students.watfordboys.org	<p>Feedback from: Harsh External Inbox x</p> <p>Harsh Brahmhatt to Harsh ▾</p> <p>Name: Harsh Email: test@example.com Age range: 18-20 Best feature: Fast-paced gameplay Worst feature: Art Other feedback: This is a test.</p>	Pass

Progress Review 4

Since the last progress review, the UI elements and the website have been developed completely. **This fulfils requirements 5 and 9.**

The client, having reviewed the current prototype, was happy with the development of these final modules of the game. They said, “UI is an essential part of a game experience, without it everything falls apart, it is something that needs to constantly be improved and adapted. Therefore, we are really happy with how it has been developed at the moment but it will need constant refinement going forwards.

The UI developed includes the pause menu with a pause button on the player to be able to access the pause menu. The pause menu is essential for navigation, such as being able to access the weapons menu, another UI element developed. The weapons menu is essential in allowing the player to choose the weapon they want to use. When the player runs out of lives and reaches gameover, they should be shown their score and be given some options about what to do next. This is done by the gameover UI screen. During gameplay, the player has attributes health and fuel which change over the course of a level, these changes need to be shown to the player so that they can plan and play accordingly. This is why the health and fuel bars were implemented. They simply show how much health / fuel the player has left out of their total capacity (100 - when the bars are completely filled). All these UI elements have been tested and work exactly as intended.

The website is essential in advertising the game, providing a place where it can be downloaded from and a place where players can give feedback on the game. This is done via the 3 pages, the about page, the download page and the feedback form page respectively. The website also features a light and dark theme which the user can toggle between to find their preference. The website has been tested and works as it should, with all features functional.

Completion: Final Review

Now, all the intended development has been completed, i.e. 100% of the development. However, this does not mean that some features have not been left out, this includes small features, but more notably, features such as the settings page have been left out. There are multiple reasons for this, often due to lack of time available and lack of importance for some features (see individual sections for specific reasons). However, more importantly, many of the features of the game have been developed, such that the game is in a really good state with all essential features. The game has been tested at each stage of the development with all tests passed for implemented features and the game can now be tested as a complete program to inform evaluation.

Validation

Building a game, there is not much open user input, therefore, validation comes in the form of ensuring that the player plays and the game conforms to the game's rules, implemented by me. There are numerous examples of this throughout the implementation, examples of some are shown below.

Player movement is a module which has user input - the direction the player is looking at. This will of course always be valid as it is not an open input. However, the game's rules which I have set include that the player must only move (forwards) if they are looking between a certain angle of elevation/depression. This is ensured by the dedicated moveCheck validation function:

```
private bool moveCheck(float cameraAngle)
{
    // if the camera angle is within min and max range,
    // player can move
    if (cameraAngle < minAngle || cameraAngle > maxAngle)
    {
        return true;
    }
    // otherwise the player should not move
    else
    {
        return false;
    };
}
```

This function simply returns true if the player's camera angle is within the required angle to move. This validates the movement by returning true only if the player should move.

Another place validation is used to ensure the player conforms with the game's rules is with the refuelling feature. The player must only refuel subject to certain conditions.

One condition of this is that the player must be standing still if they are to refuel, only then is refuelling valid. The checkStanding function is dedicated solely to ensuring this, as seen below.

```

public IEnumerator CheckStanding()
{
    // while true loop to keep running while the game plays
    while (true)
    {
        // get player position (before time wait)
        Vector3 posBefore = this.transform.position;
        // wait for 2 seconds
        yield return new WaitForSeconds(1f);
        // get player position (after time wait)
        Vector3 posAfter = this.transform.position;

        // if the player is in the same position,
        // before and after the time wait,
        // then they are standing and so its set to true,
        // otherwise it is set to false
        if (posBefore.x == posAfter.x && posBefore.z == posAfter.z)
        {
            standing = true;
        }
        else
        {
            standing = false;
        }
    }
}

```

This function ensures that standing is only set to true if the player has been in the same position before and after a short time wait of 1 second. This boolean is later used to ensure that the player only refuels if standing is true:

```

// if the player is standing,
// which is checked in the checkStanding coroutine,
// then we check if the player is on a refuel station
if (standing)
{
    //...
}

```

Another rule for refuelling is that the player must be standing on a refuel station. They should not refuel otherwise. This is validated using a ray cast and layer mask.

```
// if the ray hits an object with the fuel station mask,  
// and the collision is <= 5 units from the player,  
// then the player can refuel  
if (Physics.Raycast(StationRay, out hitInfo, 5f, stationMask))  
{  
    // the refuel station that the player is standing on  
    currentRefuelStation = hitInfo.transform.gameObject.GetComponent<fuelStation>();  
    // canRefuel set to true so that the player now refuels  
    canRefuel = true;  
}
```

As seen, can refuel is only set to true if the ray shot out from the player, downwards, hits an object with layer mask 'stationMask'. Furthermore, the program also ensures that the player is on the refuel station and not just above it, by constraining the ray to a length of only 5 units. This ensures that the player is on the refuel station, and close enough on top of it to refuel.

And finally from this, the program validates that both the player and the fuel station are in states such that refuelling is valid. This includes the player having to not be full of fuel and the station not being empty (out of fuel). This is ensured using an if statement that uses the getter method for fuel of both of these entities:

```
// only if the player fuel is not full,  
// and the fuel station is not empty, should the player refuel  
if (player.getFuel() < 100 && !currentRefuelStation.checkEmpty())  
{  
    // player fuel level is increased  
    player.addFuel(0.2f);  
    // same amount of fuel is decreased from the fuel station  
    currentRefuelStation.useFuel(0.2f);
```

This validates the fuel attributes of both the player and the fuel station, ensuring that they do not go below 0, or exceed 100.

This, along with the standing check validation and the station mask ray cast validation ensures that the player only refuels when they should.

The grapple hook feature also involves validation to ensure that the player grapples correctly and only when in a valid state.

For example, the grappling line should only be drawn when the player is grappling.

```
// only render grapple line if a joint exists
if (joint)
{
    // sets the first point of the line to the player's position
    lr.SetPosition(0, this.transform.position);
    // sets the second point of the line to the grapple point
    lr.SetPosition(1, grapplePoint);
}
```

This if statement ensures that the line is only drawn if there is an existing joint, which only exists when the player grapples.

Grappling also requires the use of fuel. The player should not be able to grapple if they are out of fuel since this is a vital rule for the game.

```
// only if the player has fuel left, should they grapple
if (player.getFuel() > 0)
{
    canGrapple = true;
```

This if statement references the getter function of the player's fuel attribute to ensure that they only grapple if they have disposable fuel at the time of grappling.

This (and other) validation ensures that the player only grapples when they should and also only to what they should (see ray cast in player attack class).

For more examples of validation, please refer to the comments of other modules of this implementation section.

Future maintenance

In the future, if others were to work on my game to maintain or improve it, it should be easy to understand what is going on in each class/module so that the game can be refreshed or improved.

First and foremost, by using comments on each and every line of my code, it is ensured that anyone reading my code with some level of technical knowledge will easily be able to understand what is going on and how everything works.

For adaptation and maintenance, it is often necessary to be able to change some of the rules of the game. Therefore, I have dedicated variables instead of hard-coding values so that these can easily be edited to change the gameplay.

For example, the player grappling module includes a minGrapple and a maxGrapple which dictate the range of the player's grappling ability. Therefore these can easily be edited to change this range and change the way the gameplay works.

```
// min grapple distance
1 reference
private float minGrapple = 2f;
// max grapple distance
1 reference
private float maxGrapple = 50f;
// these can be changed in the future,
// e.g. if a new developer designs new custom maps,
// they can change these values to improve gameplay
```

The comments ensure that a developer can understand what these variables are and what they can be used for

Similarly, the player movement depends on the maxAngle and the minAngle of the camera. These have been designed as distinct variables so that they can be changed to change/improve the gameplay. For example, if it seems that it is better if the player can look and move more, these values can be increased/decreased as necessary.

```
// min and max angle in which player can look to move
2 references
private float minAngle = 30.0f;
2 references
private float maxAngle = 330.0f;
```

Another example is the player looking around. This is dependent on the sensitivity set by me. Therefore, I have designed this as a separate variable so that another developer can change it in the future to change the gameplay, or to add a feature to allow the player to configure this depending on their preferences.

```
// mouse sensitivity,
// change this value to change how quickly the player looks around
2 references
private float sensitivity = 90f;
// not hard coding these values,
// so that they can be changed if they need to be
```

These code comments, and others which are not referenced here, ensure that the program can be updated/maintained by myself or other developers.

For more examples of code comments to support future maintenance, please refer to other modules of this implementation section.

EVALUATION

Testing to inform evaluation

Overall program user testing (Beta testing)

I gathered willing volunteers to participate in a beta test for my game. Their objective was simply to play the game, and report to me anything they found was interesting. This could be things they like / don't like about the game as well as any bugs they come across. This was to test that the modules each developed separately work together as they should and that the game functions properly.

The results of the testing are summarised into the following categories; positives feedback, negatives feedback, specific bugs/problems:

User no.	Positive feedback	Negative feedback	Specific bugs/problems
1	Fast paced gameplay	Art style	Hit animations bugs
2	-	Variation between levels (not at expected standard)	Town level not generating correctly
3	Simplistic art style	Weapon classes (not enough)	Town level not generating correctly, Enemies getting stuck in corners
4	Easy to use UI	(lack of) Accessibility	Spawning inside of environment features
5	Simple UI	-	Town level not generating correctly
6	Large enemies	Attacking enemies too hard (not much hit detection tolerance)	Town level not generating correctly, Hit animation bugs
7	-	Levels - too similar	Spawning inside of environment features
8	-	Attacking enemies too hard to register a successful hit	-
9	Fast paced grappling gameplay	-	Animation bugs
10	-	Repetitive and not fun	Town level not generating correctly

Overall, this beta test provided me with positives about the game that should be focused more on in the future, to maximise the potential of these features, as well as negative feedback about the gameplay which should be focused on in the future to make the game more fun. More crucially, I obtained valuable information about the bugs and problems which the volunteers came across. This is arguably the most important feedback as it allows me to locate and fix these bugs in the future to provide a much smoother user experience.

The positive feedback showed me that the simplistic style of both the art and UI was useful, it allows the user to understand and navigate easily, without much concern or doubt and this means that they can spend more time actually playing the game. The overall game idea of fast-paced gameplay fighting against giant enemies also seemed to be praised.

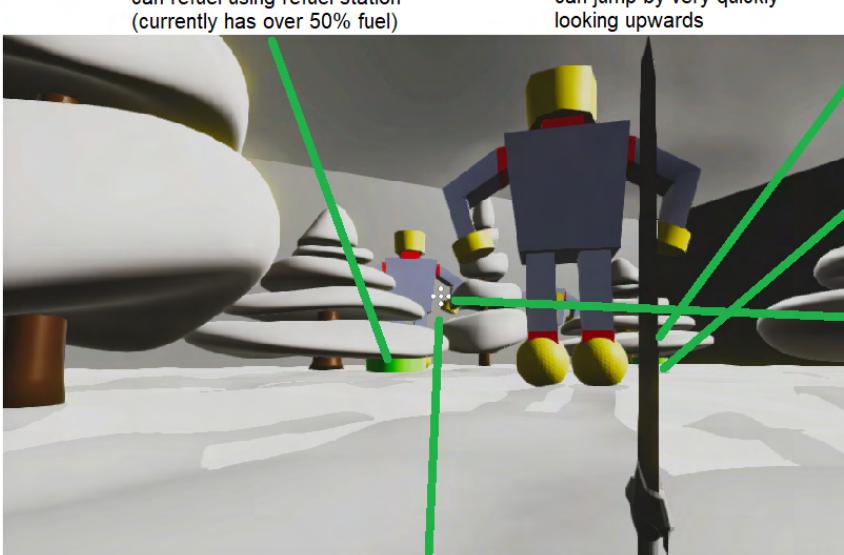
The negative feedback highlighted that although the game at its core is a good idea, it needs some more features to make it more fun, for example, more weapon options, features such as these can be implemented in the future with more time. I also found that the levels seemed to be too similar to each other. Arguably the main feature of the game is that each level is randomly generated and so unique to every other level, however, this seems to not be met to the expected level. In the future, the level generation should be revised and re-implemented in a way in which more unique levels can be generated. This may be using different types of noise and layering them to create more unique terrain, or by increasing the size of the asset set so that there is more variation in the environment of a specific level. The difficulty of the game has also been criticised with many players finding it too difficult to attack enemies. This can be improved using various methods in the future with more time to plan and test. For example, instead of using raycasts, spheercasts can be used to allow for more tolerance when attacking.

Specific bugs picked up by the volunteers are extremely useful, as I can plan to fix them in the future / in the next iteration. The most common bug seemed to be the generation of the town biome / environment. This may have been due to the sizes of the environment features of this biome but should be further investigated in the future. Fixing this bug would require re-evaluation of the town biome assets as well as the level generation algorithm. Players also seemed to have problems with spawning inside of environment features. This bug could be fixed by having a clear zone where no environment features are allowed to spawn and setting the player to spawn in this dedicated area, rather than randomly anywhere on the map. Or, the random spawning could still be used but also spawn the player with a greater y value for position so spawn above the environment features. Animations also caused problems, as these are not managed entirely with code, this problem likely lies with the actual animation file itself. Broken animations would most probably have to be replaced with new ones in the future.

Testing requirements

Test no.	Test subject	Expected output	Actual output	Pass / Fail	Link to requirements & success criteria (starts p30, through to p35)
1	GUI	All GUI tests (see below) are passed.	See GUI testing	See GUI testing	1: Acceptable (p30)
2	Environment generator	Environment generates successfully, i.e. no rendering / spawning errors with objects interfering with each other and sufficient objects spawned. Also the environment generated must be different every time.	<p>Example 1:</p> <p>Overall no rendering/spawning errors</p> <p>Terrain mesh generated successfully</p> <p>Enemies spawned and immediately functional (as seen by debug rays for vision)</p> <p>Relevant environment features and ground colour match biome</p> <p>Player spawned sucessfully (behind this cactus)</p> <p>Environment features not interfering</p> <p>Sufficient objects spawned (all environment features, enemies, player, refuel stations)</p>	Pass	2.1: Acceptable (p30) 2.2: Desired (p30) 2.3: Acceptable (p30) 2.4: Acceptable (p31) 2.5: Unacceptable (p31) 2.6: Desired (p31) 2.7: Unacceptable (p31)

			<p>Example 2:</p> <p>Overall no rendering/spawning errors</p> <p>Terrain mesh generated successfully</p> <p>Fuel stations spawned successfully</p> <p>Enemies spawned and immediately functional (as seen by debug rays for vision)</p> <p>Player spawned sucessfully</p> <p>Environment features not interfering</p> <p>Sufficient objects spawned (all environment features, enemies, player, refuel stations)</p> <p>Relevant environement features and ground colour match biome</p>	
3	Free play mode	All gameplay requirements are met, such as environment generation, player modules etc. But, no enemies.	Not implemented	Fail 3: Unacceptable (p31)

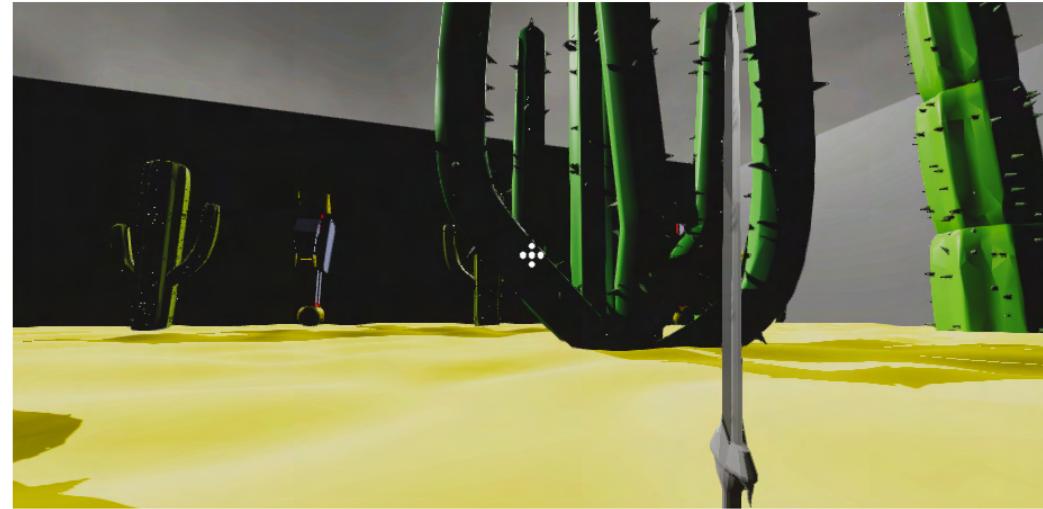
4	Player	Player functions as required, all modules work allowing the player to move, attack, take damage, etc.	 <p>can refuel using refuel station (currently has over 50% fuel)</p> <p>can jump by very quickly looking upwards</p> <p>enemy can damage player</p> <p>can attack with sword</p> <p>can grapple to the point crosshair is pointing to</p> <p>can look around in all directions</p> <p>moves automatically in the direction the player is looking</p>	Pass	4.1: Desired (p31) 4.2: Desired (p32) 4.3: Desired (p32) 4.4: Acceptable (p32)
5	UI	All UI tests (see below) are passed.	See usability testing below	See GUI testing	5: acceptable (p32)

6	Weapon classes	<p>Weapon classes should be able to be changed easily and successfully, these changes should carry over to the actual gameplay. All different weapon classes should work as they are described with respective attributes.</p>	<p>Current Selection: SNIPER</p> <p>Shows current selection</p>	Pass	6: Acceptable (p32)
---	----------------	--	---	------	---------------------

7	Enemies	<p>Enemy functions as required, all modules work allowing the enemy to move, attack, take damage, etc.</p> <p>the enemy has a collider which allows it to take damage</p> <p>enemy can attack the player using the attack colliders</p> <p>enemy moves in direction its facing</p> <p>enemy changes / adjusts direction depending on what the vision ray collides with</p>	Pass	<p>7.1: Desired (p33)</p> <p>7.2: Desired (p33)</p> <p>7.3: Desired (p33)</p> <p>7.4: Acceptable (p33)</p> <p>7.5: Unacceptable (p33)</p> <p>7.6: Unacceptable (p33)</p> <p>7.7: Acceptable (p34)</p>
---	---------	--	------	---

8	Menu (Play/Settings /Quit)	Should display all menu options with functional buttons; Play button should enter the player into gameplay. Settings button should take the player to the settings screen. Quit button should close the game application.	<p>Spinning example level - random and different every time</p>	Pass	8.1: Desired (p34) 8.2: Desired (p34) 8.3: Acceptable (p34) 8.5: Desired (p34)
---	----------------------------------	---	---	------	---

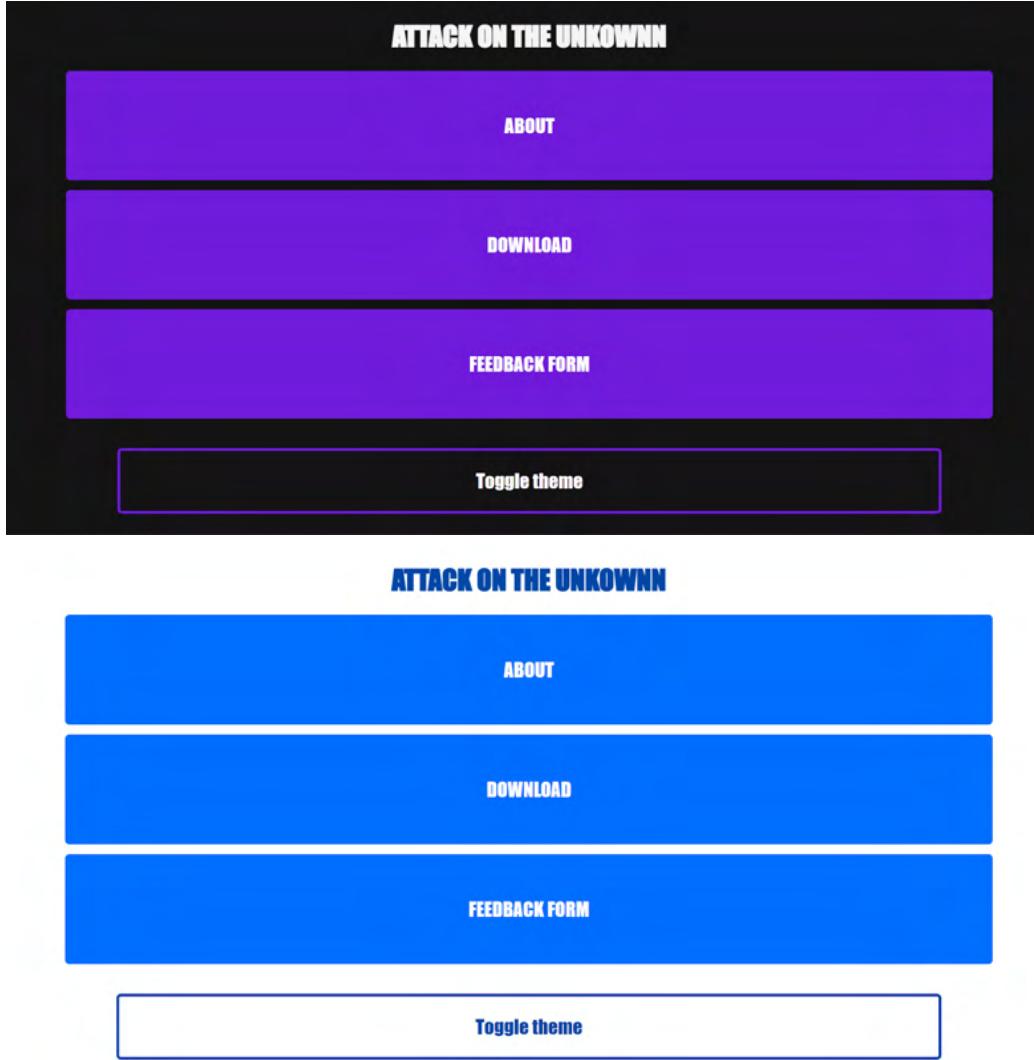
Play button immediately
leads to game



Settings button
immediately leads to
settings page (not
implemented)

SETTINGS

			<p>Quit button exits application as indicated via this message</p> 		
8.4	Settings page	<p>Graphics level should toggle between levels. Controls should change to the button the player presses. Volume should change to the percentage of slider selected.</p>	<p>Settings button immediately leads to settings page (not implemented)</p> 	Fail	8.4: Unacceptable (p34)

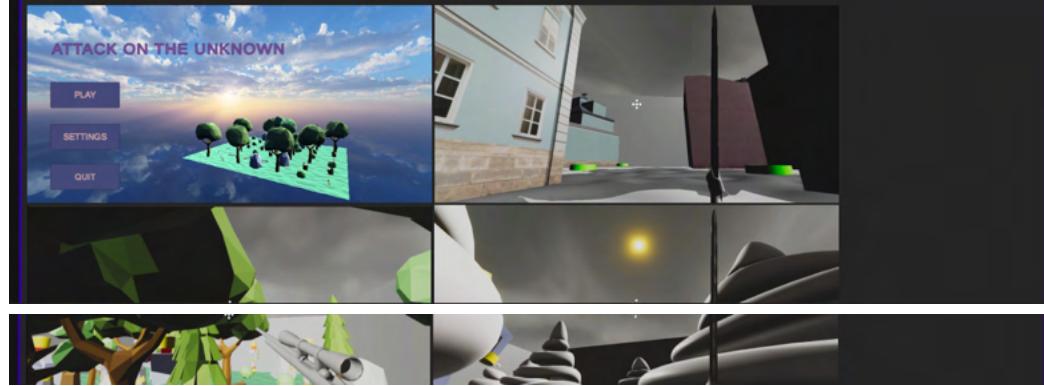
9	Website	<p>Information should be viewable, game should be downloadable, feedback form should accept user inputs.</p> 	Pass	9: Desired (p35)
---	---------	--	------	------------------

ABOUT

What is Attack on the unknown?

- Attack on the Unknown is a cardboard VR mobile game
- Defeat the giants with your sword or sniper!
- Lots of biomes to explore!
- Levels are randomly generated, the game will never end...

Gallery



Harsh Brahmbhatt
Watford Boys Grammar School

5124
17655

251



Supported Devices

- iOS mobile devices
- Android mobile devices
- Windows devices (alternate PC version of the game)

Help and support

- Use the feedback form to get in touch or provide any feedback.
- The feedback form page can be found on the home page

Toggle theme

DOWNLOAD

[Click here to download Attack on the Unknown](#)

Toggle theme

Feedback form

Name

Email address

Are you female

Select your age range
10-12+
13-14+
15-16+
17+

Choose the genres which you liked most about the game
Fantasy genres
Action/beat genres
Horror
Drama
Sci-Fi
Thriller

Choose the genres which you liked least about the game
Fast-paced genres
Action/beat genres
Horror
Drama
Sci-Fi
Thriller

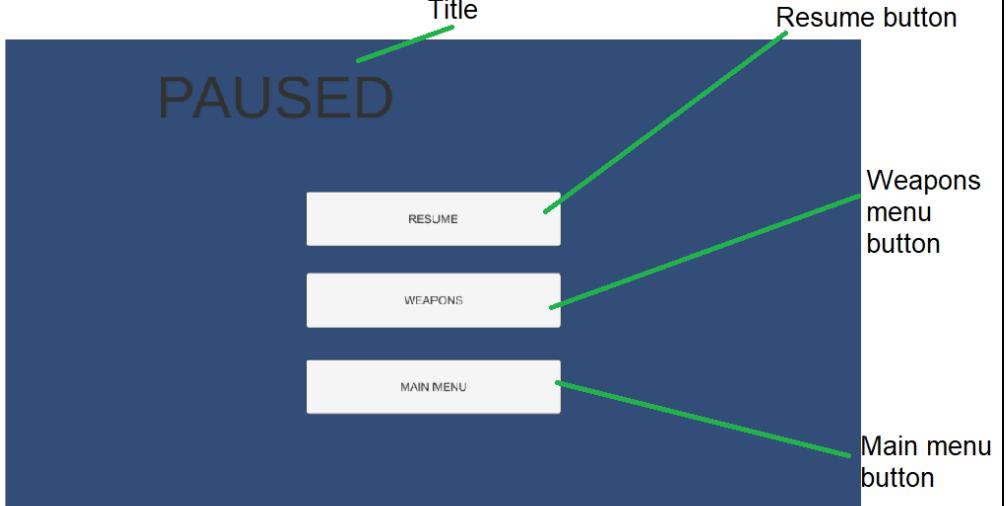
Any other feedback or suggestions?

[Toggle theme](#)

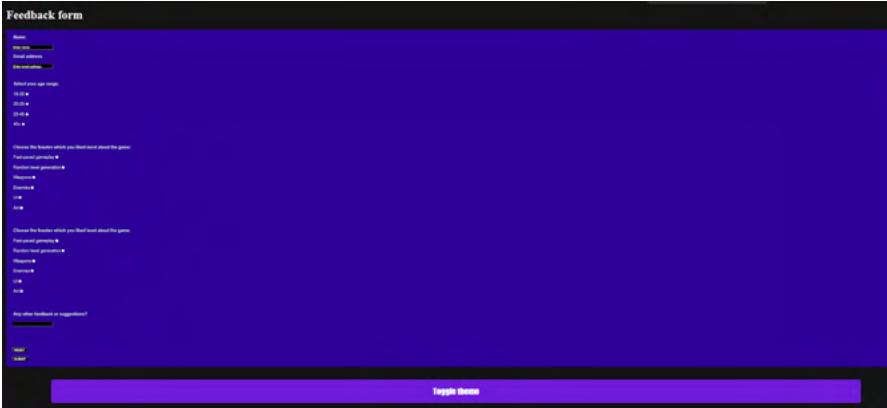
Note: the webpage is very zoomed out to allow the entire form to fit on one page, with the default zoom settings, the questions do not appear so small.

Graphical user interface testing

Test no.	Test subject	Expected output	Actual output	Pass / Fail
1	Main Menu	<p>The Main menu screen should be displayed to the player as designed and required. All 3 buttons should work, i.e. allowing inputs. Play button should take player to game. Settings button should take the player to the settings/options page. Exit button should close the application.</p>	 <p>Title</p> <p>Play button - leads to game scene</p> <p>Settings button - leads to settings page/scene</p> <p>Quit button - exits application</p> <p>Spinning example level - random and different every time</p>	Pass

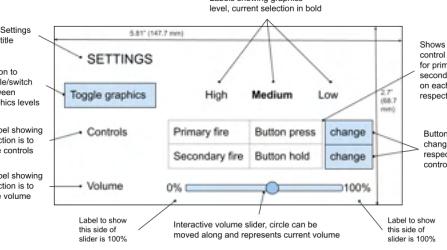
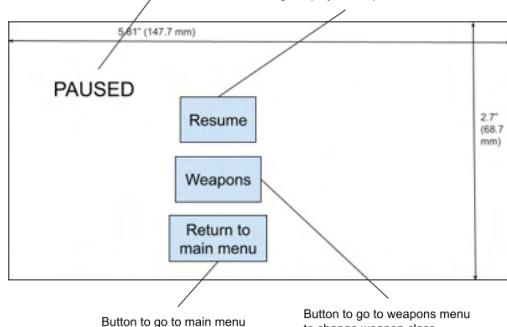
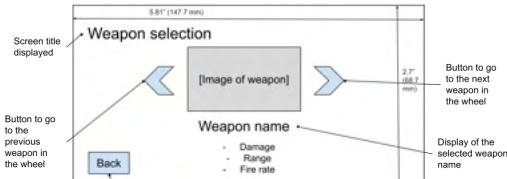
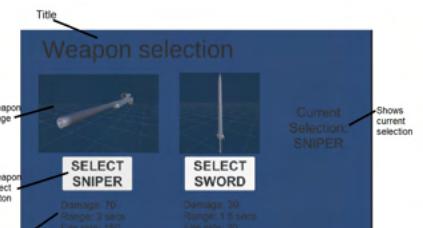
2	Settings	<p>The Settings screen should be displayed to the player as designed and required. The player should be able to change any of the 3 options to their preference.</p> <p>Graphics toggle button - Graphics levels should switch between each other.</p> <p>Change controls button - Primary/secondary fire controls should change the player's inputs.</p> <p>Volume slider - Volume should change to the percentage of the slider player has clicked on.</p>	<p>Settings button immediately leads to settings page (not implemented)</p> 	Fail
3	Pause menu	<p>The pause menu screen should be displayed to the player as designed and required. All 3 buttons should work, i.e. allowing inputs.</p> <p>Unpause button should return the player to gameplay.</p> <p>Weapons menu button should take the player to the weapons menu screen.</p> <p>Return to main menu button should take the player to the main menu.</p>		Pass

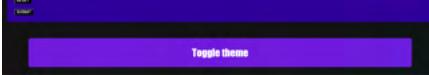
4	Weapons menu	<p>The Weapons menu screen should be displayed to the player as designed and required. All 3 buttons should work, i.e. allowing inputs.</p> <p>The previous/last weapon buttons should switch between the weapons and change the corresponding information such as image, name and attributes.</p> <p>The back button should take the player back to the game in a paused state, i.e. to the pause menu.</p>	<p>Title</p> <p>Weapon image</p> <p>Weapon select button</p> <p>Weapon stats</p> <p>SELECT SNIPER</p> <p>Damage: 70 Range: 3 secs Fire rate: 150</p> <p>SELECT SWORD</p> <p>Damage: 30 Range: 1.5 secs Fire rate: 20</p> <p>Current Selection: SNIPER</p> <p>Shows current selection</p>	Pass
---	--------------	--	--	------

5	Website feedback form	<p>The Website feedback form should be displayed to the player as designed and required.</p> <p>Text data entered should appear in the textbox and be stored when the player clicks submit.</p> <p>Clicked lozenges should display as filled and the corresponding value (option) should be stored when the player clicks submit.</p> <p>Clicked boxes should display as filled and the corresponding value (option) should be stored when the player clicks submit.</p>	 <p>Note: the webpage is very zoomed out to allow the entire form to fit on one page, with the default zoom settings, the questions do not appear so small</p>	Pass
---	-----------------------	--	--	------

Graphical user interface comparison with designs

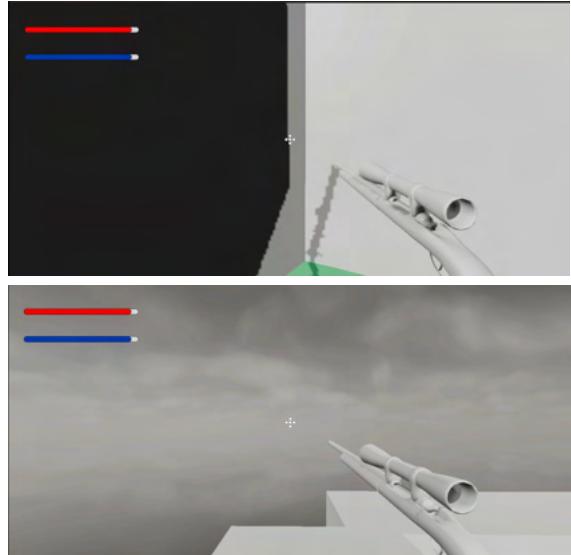
GUI / Usability feature	Design	Implemented	Comparison
Main Menu	 <ul style="list-style-type: none"> Title of the game displayed here Button which allows player to progress to gameplay Button which allows player to go options/settings page Button which exits the application 	 <ul style="list-style-type: none"> Title Play button - leads to game scene Settings button - leads to settings page/scene Quit button - exits application 	<p>The design and implemented main menu GUIs are very similar, the buttons and title are laid out just as designed. The colours were not included in the design, and in the implementation a purple theme was used to try and appeal to a general audience. The main difference is the background and the example level which spins dynamically in the implementation.</p>

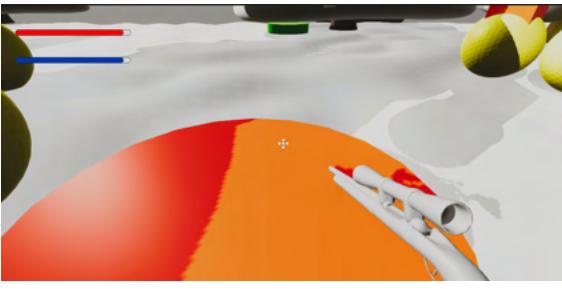
			This makes the main menu more visually appealing and excites the player to play the game.
Settings	 <p>The diagram illustrates the 'SETTINGS' menu. It includes a 'Toggle graphics' button, a 'Volume' slider from 0% to 100%, and sections for primary and secondary fire controls. Dimensions shown are 5.81" (147.7 mm) wide and 2.7" (68.7 mm) high.</p>	Not implemented	-
Pause menu	 <p>The diagram shows the 'PAUSED' menu with three buttons: 'Resume', 'Weapons', and 'Return to main menu'. A dimension of 5.81" (147.7 mm) is indicated for the width.</p>		The pause menu implementation is very simple and very similar to the one designed. It consists of a title and 3 buttons, 'resume', 'weapons', 'main menu' respectively. The resume button returns the player to the game. The weapons button takes the player to the weapons menu. And the main menu button takes the player back to the main menu.
Weapons menu	 <p>The diagram shows the 'Weapon selection' menu with a scroll wheel interface. It displays weapon images, names, and attributes like damage, range, and fire rate. Dimensions are 5.81" (147.7 mm) wide and 2.7" (68.7 mm) high.</p>		The implemented weapons menu is significantly different to the design. First, the design uses buttons to scroll through weapon classes, however, in the implementation, I realised this would be excessive as there are only 2 weapon classes. Therefore 2 simple buttons are used instead. However, this does require extra information about which weapon is

			actually selected, and this is done simply through text in the space on the side which changes depending on which button is pressed.
Website feedback form	<p>We want your feedback...</p> <p>Short written question [textbox answer]</p> <p>Multiple choice question <input checked="" type="radio"/> Option 1 <input type="radio"/> Option 2 <input type="radio"/> Option 3 ...</p> <p>Multi-select question <input type="checkbox"/> Option 1 <input type="checkbox"/> Option 2 <input type="checkbox"/> Option 3 ...</p> <p>Long written question [textbox answer]</p> <p>...</p>	  	<p>Note: the webpage is very zoomed out to allow the entire form to fit on one page, with the default zoom settings, the questions do not appear so small.</p> <p>The website feedback form implemented has only a few differences to the design. The radio options used go down the page instead of across, and this is a minor thing which doesn't improve or worsen the quality or user experience, yet was easier to implement. It also turns out that no longer answer questions were relevant or appropriate for the questionnaire so no implementation of a long text box answer was needed.</p>

Robustness testing

Due to the nature of my game and solution, with no open user input, the game is mostly self-validating. However, the game has its own set of rules that the player should not be able to break. Therefore, as long as there are measures in place to ensure the player cannot break these rules, the solution is robust.

Test	Expected output	Actual output	Pass / Fail	Justification / Robustness feature
Try to leave the level / go outside the walls	Player cannot leave the playable area / level		Pass	Levels have colliders on the walls, even larger than the size of the walls to ensure the player cannot leave the level

Try to use the grappling gear with 0 fuel	Player cannot use the grappling gear		Pass	Implementation such that player cannot grapple if they have 0 fuel
Try to refuel at a refuel station which has 0 fuel left in the station	Player does not refuel		Pass	Implementation such that fuel stations with 0 fuel cannot be used to refuel
Try to pass the floor	Player should not be able to fall/go through the floor		Pass	Collisions and gravity are simulated and handled by unity. This means that this is automatically met as long

Client acceptance testing

I then carried out a user acceptance test where the client carried out the following instructions to test the features of the game, to ensure that they were happy with it

1. Start the program. ✓
2. Through the menu's settings button, go to settings/options page. ✓
3. Change graphics level to whichever you desire using the toggle graphics button. ✗
4. Change the primary and secondary fire controls to whichever you desire, using the change controls button. ✗
5. Change the volume using the volume slider to whichever you desire. ✗
6. Click the back button to go back to the main menu. ✓
7. Click the play button to enter gameplay. ✓
8. Play the game by defeating all enemies to progress to the next level, for at least 3 levels to see how different each level is. ✓
9. Press the pause button to enter the pause menu. ✓
10. Click the unpause button to go back to gameplay. ✓
11. Switch to free play to play with no enemies. ✗
12. Go to the weapons menu via the button on pause menu, to change weapon class using the next/previous buttons. ✓
13. Click the back button to go back to the pause menu. ✓
14. Click the back to main menu button to go back to the main menu. ✓
15. Click the exit button to close the game/application. ✓
16. Go to Website feedback form. ✓
17. Fill out the form and click submit. ✓

The client was then asked to answer the following questions as well as comment on anything that they thought was important

Question 1: What did you like and not like about the program?

Answer 1: The gameplay was good as it feels quite dynamic as you run around using the grapple hook, however, it can be a bit buggy navigating as the mesh colliders seem to sometimes not register the grappling. We liked the implementation of both weapons; however, as we improve the game in the future, we would like there to be a greater pool of weapon classes to keep the game fresh and fun. The random level generation is a really solid start, going forwards we'd love to see this algorithm enhanced and improved such that there is more variation between levels. Currently, the USP we are aiming for is the 'infinite level' feature, however, currently we are not hugely happy with its standard, improving it will really make the game exciting. We're happy with some of the smaller

features being left out after discussing why with you, however we are a little disappointed that some others have been left out. The 2 main left out features that stand out to us are the settings and the freeplay mode. Of course, we understand that these can easily be implemented at a later date with more time, but we did just want to flag that this is something that is really important. If we have anything else that we've forgotten and want to mention we will definitely let you know.

Question 2: What are the key changes you would like to see in the program to better meet the requirements we had set?

Answer 2: Yeah, so we've mentioned these already in the first question really! But just to highlight exactly what we'd want you to focus on in the future; the random level generation, the user settings feature and the freeplay mode are probably of utmost importance to us.

Question 3: What are the changes you would like to see in the program for overall better user experience.

Answer 3: I think the settings feature is really a big thing in this area, allowing users to adjust settings to their personal preferences means that the game will be suited to them that much more. When players can use settings which are similar to that of their other, favourite games, there is a greater chance that this game will also allow them to have just as much fun as their favourites. UI in general is fine, the simplistic style is perfect as its clear, to the point and means that the players can focus more on the gameplay. However, as the graphics of the game are improved in the future, it is likely that the UI will also require a stylistic overhaul so that the quality matches that of the graphics. One small annoyance that we did find though, was that it took a while to find the pause button! Of course, we understand that UI and user input is difficult with a VR game, however, some indication of where the pause button is would be useful, maybe when the player first plays the game, or via a tutorial.

Question 4: What additional features would you like to see in the program? (I.e. features not in the requirements set)

Answer 4: The mobility provided by the grappling gear is incredibly fun, overtime to keep this fresh, maybe a variety of mobility items would be interesting. Possibly items similar to the grapple gun but with altered settings or maybe even completely a type of mobility item could be introduced. We feel that this is a solid game idea that could be fun for a very, very long time as long as it's polished. Therefore, we'd like to focus on fixing problems and implementing the essential features first, before considering new features.

Evaluation of solution

Evaluation of project completion

Success criteria (see p30 through to p35)	1	2	3	4	5	6	7	8	9
Pass / Partial success / Fail	Green	Orange	Red	Green	Green	Green	Orange	Orange	Green
	Green	Orange	Red	Green	Green	Green	Orange	Orange	Green

Requirement 1: Must be playable with a GUI, in VR with Cardboard VR viewer.

Test evidence can be found on page 242, requirement/criteria found on page 30.

This has been attempted multiple times by; myself, my clients, and also the beta testers, none of which had problems with the GUI or VR. Therefore, this criteria is met.

Requirement 2: The game must randomly / procedurally generate levels.

Test evidence can be found on pages 242-243, requirement/criteria found on pages 30-31.

This overall requirement is met to an extent. The levels are all randomly generated and therefore, are all technically distinct from each other, i.e. you can never really have 2 levels which are exactly the same. However, this requirement is not completely met as, the levels may technically be different, but to the player they come across fairly similar, the levels only really vary over the 4 biomes. The level generation does involve biomes and each level consists of 1 room, therefore these particular requirements are met. Levels do successfully spawn enemies to be defeated as well as the player at the start of the level, and refuelling stations are also included, therefore, these requirements are met. Levels do not have boss enemies, nor do they have civilians, therefore, these particular requirements are not met as they have not been implemented.

Requirement 3: The game should have a free-play, practise mode.

Test evidence can be found on page 243, requirement/criteria found on page 31.

This feature was not implemented for various reasons. First of all, a lack of time meant that this feature had to be left out to focus more on the more important features that were essential to the game. Following on from this, a free-play mode, although useful for the player, is not completely essential to the game's function. Therefore this requirement has not currently been met.

Requirement 4: The player.

Test evidence can be found on page 244, requirement/criteria found on pages 31-32.

This overall requirement has been fully met, the player functions as required in all areas. The player can look around and move in first person, with extra mobility features such as jumping. The player can also fight enemies using weapons and moving rapidly using the grappling gun (which can be refuelled), as tested repeatedly by the beta testers. Therefore all of these criteria are met.

Requirement 5: The game should have sufficient in game UI, such as health, fuel, etc.

Test evidence can be found on page 244, requirement/criteria found on page 32.

All the essential UI implemented as seen in the usability testing, therefore this criteria is met.

Requirement 6: The game must have some weapon classes which the player can unlock and choose from.

Test evidence can be found on page 245, requirement/criteria found on page 32.

The game has 2 weapons, a sniper and a sword, that the player can alternate between by choosing the weapon of their choice in the weapons menu. Both weapons work as designed and as desired, with all testing suggesting that the weapons work perfectly as intended.

Therefore, this criteria is met.

Requirement 7: Enemies.

Test evidence can be found on page 246, requirement/criteria found on pages 33-34.

This overall requirement is met to an extent. The enemies have all the basic functionalities as required, this involves automatic enemy movement (avoiding obstructions), attacks and the ability to take damage from the player. The enemy also breaks apart depending on where they are struck, e.g. if they are hit at the weak points. Therefore these particular requirements are met. However, some requirements are not met; the enemies are all identical, and do not come in various types and sizes, nor is there a boss enemy. This is simply because they were not implemented in the limited development time of the game, as they are not completely essential. Therefore these particular requirements are not met, and overall, the enemies requirement is met to a partial extent.

Requirement 8: The game should have a main menu when started.

Test evidence found on page 247-249, requirement/criteria found on page 34.

This criteria is met to a partial extent. The main menu is implemented and works perfectly as intended. It has a title, a nice background and also an example level, and 3 buttons. The buttons; play, settings and quit, work perfectly as intended; the play button takes the player to the game to allow them to play, the settings button takes the player to the settings page and the quit button successfully closes the application. However, one criteria not met is that of the settings. The settings page exists however is not implemented such that the player can change the settings to their liking. This was again done due to a lack of time to learn how to implement this as well as actually implement it. Although still really important, the game still works fine without this feature and so it was left out with other features having higher priority. Therefore, overall this criteria was met to an extent.

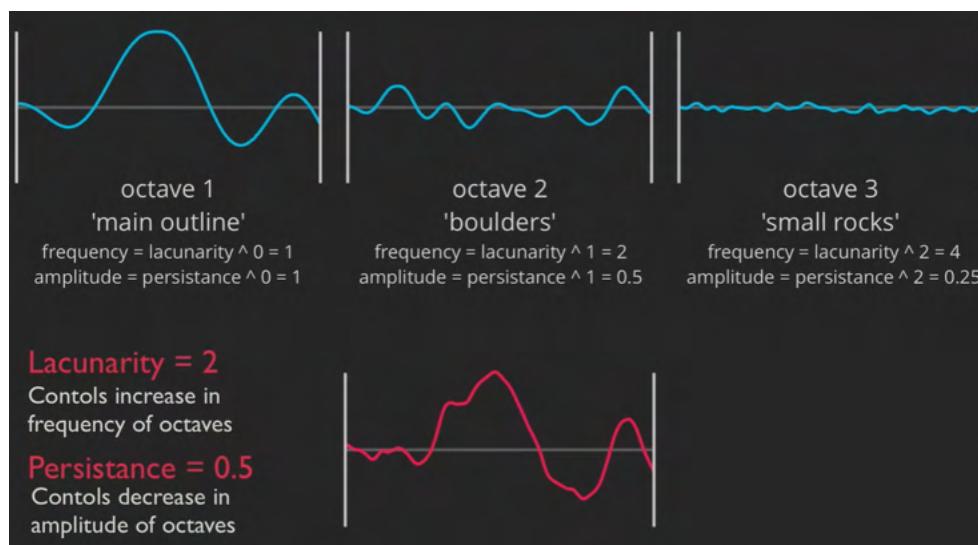
Requirement 9: A website, to advertise the game and allow downloads.

Test evidence can be found on page 250, requirement/criteria found on page 35.

The website criteria is met completely. The website has a main menu, about page, download page, and feedback page. The main menu allows the user to navigate between the other 3 pages. The about page gives details and information about what the game is, the download page provides a link to download the game, and the user feedback page includes a form that the user can fill out to provide their feedback. Each of these work as intended and therefore, this requirement is met completely.

Possible improvements to meet unmet criteria

One of the first, main, things to do to better meet the success criteria is to improve the random level generation such that it provides more variety between levels, making them more unique from one another. This could be done by improving multiple different aspects of the generation, for example, by generating better terrain. The current terrain mesh uses a predominantly perlin noise, layered with other mathematical functions such as the sine function. This can be improved with more time by involving **extra float variables such as lacunarity and persistence**. **Lacunarity > 1, and persistence < 1, allows the use of multiple octaves of noise maps, the lacunarity variable allows the noise maps to increase in detail, and the persistence variable controls the effect of a noise map on the final map, as you want small details to have less of an effect on the final map.**



However, this only improves the terrain mesh, however, there is much more to a level than this. The environment features are what really make up the environment, and this aspect of the level generation should be improved. One way of doing so could be by not spawning environment

objects randomly, for example, they could be spawned depending on the amplitude/height of the terrain at a point. A low valley is likely to have larger rocks and a mountain peak is less likely to have trees, but instead, snow. This can be achieved by sampling points randomly across the level (**possibly using a for loop until a certain number of sample points are collected**) and then observing the height values for these points. **Using data types such as a dictionary or a hash table to store the environment features, environment objects can be assigned height ranges**, so for a certain height, there are only a few environment objects applicable, of which, one can be randomly chosen. This would still generate random levels, as the terrain mesh is randomly generated, however, this would make the levels seem more distinct from each other and more realistic. Adding more assets for a larger set of environment objects would also create more variety.

Another place for improvement to meet the criteria would be the implementation of a free play mode. This is a much easier future implementation, and while it isn't essential to the game, it would be very useful for the player. To achieve this, the same environment generation algorithm can be used, but without the enemy generation function. This would create a level that the player can run around and practise on without the threat of any enemies. **This is a good example of reusable code, and means that the game will be more efficient, smaller and easier to run.** However, it would be more useful if the player could control how many enemies they want to practise with. A function could be developed which takes a **player input, possibly using raycast, of a position. A single enemy can be then spawned using instantiate and passing this position as the parameter.** The player can repeat this as many times as they want to spawn as many enemies as they want. **Furthermore, a queue can then be used to store these enemies as they are spawned and killed. Therefore, functionality of enemy removal can be implemented, perhaps using a UI button. If the player at any point decides they've added too many enemies, via this button, they can be accessed and removed from the queue and level.** This gives the player a lot of control over how they want to play. Access to this freeplay mode can simply be via an extra button on the pause menu.

Another key area to improve and meet the success criteria better, would be to improve the enemies. Although they currently all work as intended, moving, attacking, reacting perfectly, they are all identical. Some of the beta testing feedback found that the game seemed to become repetitive over time as all the enemies are the same. One of the criteria was to create different types of enemies and this would make the game more fun by keeping it fresh. This could be achieved, firstly by creating more enemy models, of various sizes, and assigning different and unique weak points that the player would then have to try and learn from each enemy. Further to this, different enemies should have different behaviours to surprise the player from time to time. This can be done by **introducing more attributes of the base enemy class, such as attack speed, aggression, and vision floats, or even attributes specific to extended classes for special types of enemies.** For example, an enemy with a high aggression attribute might attack more often, and an enemy with a higher vision attribute will be able to lock on to and follow the player at greater distances. By also implementing (better) animations, the enemies would seem more realistic and therefore unique. Furthermore, **in the game loop (while true),**

selection (if/else) can be used to play certain animations depending on certain conditions. For example, if the player is close, then play an animation such as the enemy pointing at the player, to signify to the player that the enemy is coming to attack them.

Evaluation of usability features completion

GUI test no. (see p102 to p103)	1	2	3	4	5
Pass / Partial success / Fail	Green	Red	Green	Green	Green

Requirement 1: Main menu.

The main menu consists of a title, a background and 3 buttons; play, settings, and quit. Play allows the player to play the game, Settings takes the player to the settings page. Quit exits the application. Therefore this requirement is met.

Requirement 2: Settings.

The settings page exists however is not implemented such that the player can change the settings to their liking. This was again done due to a lack of time to learn how to implement this as well as actually implement it. Although still really important, the game still works fine without this feature and so it was left out with other features having higher priority.

Requirement 3: Pause menu.

The pause menu pauses the game and consists of 3 buttons; resume, weapons and main menu. Resume allows the player to return to the game. Weapons takes the player to the weapons menu. Main menu takes the player back to the main menu. Therefore, this requirement is met.

Requirement 4: Weapons menu

The weapons menu shows the currently selected weapon, and allows the player to select 1 of 2 weapons shows (with stats), therefore, this requirement is met.

Requirement 5: Website feedback form

The website feedback form consists of multiple questions and allows users to submit responses, therefore, this requirement is met.

Possible improvements to meet unmet usability criteria

The main unmet criteria of the usability features is that of the settings menu. A settings menu, as discussed, is invaluable as it allows the player to play the game to their preferences. The settings menu would need to consist of 3 main settings, a graphics level toggle, controls binding

and a volume slider. The graphics level toggle would toggle between 3 graphics levels; high, medium and low. **This would be achieved using an integer variable representing the index value of the current graphics level in the graphics level array. When the button is pressed, this value can simply be increased, i.e. pointing to the next graphics level.** Of course, after the last value in the array, the pointer would have to loop around back to the start. The controls binding would be fairly simple as there are only 2 main inputs, the primary fire and the secondary fire, each of these can be changed by first pressing the relevant button, then the next button pressed can be assigned to the primary/secondary fire, as per the player's choice. Lastly the volume slider, would also use a **simple float variable for volume, clamped between 0 and 100, as it represents the volume percentage.** As the user slides the slider, linear interpolation can be used to calculate a percentage value for the volume variable, depending on the position of the slider along the volume bar.

Maintenance & future development limitations and issues + possible improvements to aid

Since my program is a game, there is not a huge amount of complex maintenance required, only occasional tweaks of attribute values to perhaps balance the game and error fixing. In terms of future development, as discussed above, there is a lot of possible future development and the program needs to be developed in a way that aids this. At this moment in time, there are various ways in which this is done such as code comments and the use of variables instead of hardcoding certain values. However, there are definitely some limitations that hurt future maintenance and development.

One way in which the solution would be developed further in the future is by improving the weapons menu. For example, if the solution were to be maintained and developed further such that the client wanted extra, new weapons to be added to the game (**adaptive maintenance**), with the current design, this would include adding another widget for the new weapon, with a button to select it as well as showing its stats. Although this could be fine, over time as more and more weapons may be added this would not be ideal and things may get a little cramped. Therefore, as an example of **perfective maintenance**, the design of the weapons menu could be changed to better suit future development of the game. To better deal with this limitation in the future I could, for example, instead of widgets with separate buttons for each weapon, implement the menu with a scroll feature. This would include information about the currently selected weapon, with buttons to cycle through all the weapons. This means that as many weapons can be added as needed, however, it will remain just as easy to view each weapon and read its stats, while using all the space on the screen optimally. **This can be achieved using a circular queue like structure. A pointer would have to be used for the currently viewed weapon, and as the player uses the buttons to cycle through the weapons, the pointer can be increased/decreased. A circular queue is required so that it loops back around to itself and the player can more easily cycle through the weapons.**

Another future maintenance limitation of my solution is if the game is developed further to introduce new, different types of enemies. **This is adaptive maintenance and would possibly be achieved by developing child classes which extend from the base enemy class.** This would allow the special types of enemies to work differently with additional attributes, such as attack speed, vision and aggression. However, as the program currently works based on the normal enemy type, and so is limited as having new attributes will not be useful or change the behaviour of the enemies at all.

To improve this, to allow enemies which are more varied to be created and used, the program should have more general methods for the base enemy class, and then leave the more varied, specific functionalities to be dealt with by the specific child enemy classes. For example, all enemies will die in the same way, they will all move forwards idly in the same way, however, they are likely to attack differently to one another. Therefore, such methods should be left to be implemented in the specific enemy classes. This would better allow adaptive maintenance such as the addition of new enemy types in the game.

Another way the game may be maintained in the future is by balancing the game, for example, the player movement may be adjusted to make the player faster/slower, or easier to control. Or instead the grappling ability may need to be adjusted, for example the spring joint dampening or spring values may need to be changed to change how the grappling works, how quickly the player is launched to the point which they are grappling to.

This is corrective maintenance, and although there are some comments in place at the moment, it would be very difficult to change these values to provide the desired result without simply using trial and error. To improve this limitation in the future, the program should possibly use better comments, comments which tell a developer in the future what an attribute does if you increase/decrease it, what effect it has on everything else, rather than just what the attribute actually is. Also, attributes can be clamped, this ensures that they are only changed within a range of values. Some attributes may cause the game to completely break / not work if they are changed outside of normal ranges, e.g. the sensitivity of the player movement can never be negative. Therefore, by clamping attributes, it means that future developers don't create new bugs and can easily tweak the values to maintain the game.

BIBLIOGRAPHY

Existing solutions research (analysis section)

Last epoch:

- https://www.youtube.com/watch?v=40yP_Tx6fQ0

Minecraft:

- [https://minecraft.fandom.com/wiki/Seed_\(level_generation\)#World_generation](https://minecraft.fandom.com/wiki/Seed_(level_generation)#World_generation)
- <https://core.ac.uk/download/pdf/250147208.pdf>

The Elder Scrolls V: Skyrim VR:

- <https://core.ac.uk/download/pdf/250147208.pdf>
- <https://arstechnica.com/gaming/2017/11/the-world-of-skyrim-is-thrilling-and-flawed-in-vr/#:~:text=General%20movement%20in%20Skyrim%20VR,walk%2Dforward%22%20option>
- <https://uploadvr.com/preview-skyrim-vr-without-teleportation/>

Limitations research (analysis section)

Problems with VR:

- <https://www.lifelites.org/media/636696/VR-Risk-assessment-FINAL.pdf>
- <https://edition.cnn.com/2017/12/13/health/virtual-reality-vr-dangers-safety/index.html>

Hardware & Software requirements (analysis section)

- <https://docs.unity3d.com/Manual/system-requirements.html>

Main menu (implementation section)

Skybox:

- <https://assetstore.unity.com/packages/2d/textures-materials/sky/allsky-free-10-sky-skybox-set-146014>

Note: Skybox from this resource also used in scenes other than the main menu

Player (implementation section)

Crosshair asset:

- <https://www.kenney.nl/assets/crosshair-pack>

Player movement (implementation section)

Code:

- https://youtu.be/_QajrabyTJc

Player attack (implementation section)

Code:

- <https://www.youtube.com/watch?v=Xgh4v1w5DxU>
- https://github.com/DaniDevy/FPS_Movement_Rigidbody/blob/master/GrapplingGun.cs

Environment generation (implementation section)

Code:

- <https://youtu.be/eJEpeUH1EMq>
- <https://youtu.be/64NbIGkAabk>

Environment features assets:

Forest:

- <https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-simple-nature-pack-162153>

Town:

- <https://assetstore.unity.com/packages/3d/environments/urban/town-houses-pack-42717>

Snowy forest:

- <https://www.turbosquid.com/3d-models/3d-christmas-tree-1233006>

Desert:

- <https://www.turbosquid.com/3d-models/3d-set-cactus-1588053>

Weapons (implementation section)

Weapon model assets:

Sword:

- <https://www.turbosquid.com/3d-models/swords-medieval-fantasy-obj-free/596671>

Sniper:

- <https://www.turbosquid.com/3d-models/free-m24-sniper-3d-model/366847>

Possible improvements to meet unmet criteria (evaluation section)

Procedural mesh generation (image):

- <https://youtu.be/wbpMiKiSKm8>

APPENDIX

'menuButtons.cs'

```
using UnityEngine;

using UnityEngine.SceneManagement;

public class menuButtons : MonoBehaviour

// inherits MonoBehaviour to use Unity specific methods

{

    private string sceneName;

    // scene loaded using this sceneName string,

    // set to different value depending on button pressed


    public void playButtonClick()

    // module for when play button is pressed

    {

        sceneName = "game";

        loadScene(sceneName);

    }

    // sceneName set to required name for gameplay,

    // loadScene module called using this sceneName


    public void settingsButtonClick()

    // module for when settings button is pressed
```

```

{

    sceneName = "settings";

    loadScene(sceneName);

}

// sceneName set to required name for settings page,
// loadScene module called using this sceneName


public void loadScene(string sceneName)

// modules used to load a scene using sceneName

{

    Scene newScene = SceneManager.GetSceneByName(sceneName);

    SceneManager.LoadScene(sceneName, LoadSceneMode.Single);

    SceneManager.SetActiveScene(newScene);

}

// creates a new scene, loads it and sets it as active,
// required so player can play on the scene

public void quitButtonClick()

// module for when quit button is pressed

{

    Debug.Log("Application now exiting");

    Application.Quit();

}

// sends a console message and then exits application
}

```

'envSpin.cs'

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class envSpin : MonoBehaviour

{

    public GameObject target;

    void Update()

    {

        transform.RotateAround(target.transform.position, Vector3.up, 10 * Time.deltaTime);

    }

}
```

'Player.cs'

```
using UnityEngine;

public class player : MonoBehaviour
// inherits MonoBehaviour to use Unity specific methods
// in this class and child classes
{
    // reference to the main camera
    // which is used in gameplay
    private GameObject MainCam;
    // reference to the gameOver camera
    // which is used when it is game over
    private GameObject GameOverCam;
    // reference to the environment generator
    private EnvironmentGenerator envGen;

    //PLAYER ATTRIBUTES
```

```

private int health = 100;
// player's health attribute, set to 100

private int lives = 5;
// player's lives attribute, starts with 5 lives

protected float speed = 15f;
// player's speed attribute, set to 15

private Weapon weapon;
// player's selected weapon

private float fuel = 100f;
// player's fuel attribute used by mobility gear

private int enemiesKilled = 0;
// number of enemies the player has killed

private int score = 0;
// player's score attribute

#region GETTERS/SETTERS

#region health methods
public int getHealth()
{
    return health;
}
#endregion health

#region speed methods
public void addSpeed(float _speed)
{
    speed += _speed;
}

public float getSpeed()
{
    return speed;
}

```

```

public void setSpeed(float _speed)
{
    speed = _speed;
}
endregion speed

#region fuel methods
public void addFuel(float _fuel)
{
    // uses/refills fuel by given amount
    fuel += _fuel;
}

public float getFuel()
{
    // MAINTENANCE
    // allows you to check fuel
    return fuel;
}

public void setFuel(float _fuel)
{
    // sets the fuel level to a given value
    fuel = _fuel;
}
#endregion fuel

#region weapon methods
public void setWeapon(Weapon _weapon)
{
    // sets the weapon to a passed weapon
    weapon = _weapon;
}

public Weapon getWeapon()
{
    // provides access to the weapon
    return weapon;
}
#endregion weapon

#region score methods

```

```

public int getScore()
{
    return score;
}

public void addScore(int _score)
{
    // adds given score to current score
    score += _score;
}
#endifregion score

#endifregion GETTERS/SETTERS

private void Start()
{
    // gets the main camera
    MainCam = GameObject.FindGameObjectWithTag("MainCamera");
    // gets the pause camera
    GameOverCam = GameObject.FindGameObjectWithTag("GameOverCamera");
    // gets the environment generator script
    envGen = GameObject.FindGameObjectWithTag("Environment
Generator").GetComponent<EnvironmentGenerator>();
}

public void takeDamage(int _damage)
{
    // removes from health, given damage amount
    health -= _damage;
    Debug.Log("HEALTH: " + health);
    // only checks death if damage taken
    checkDeath();
}

private void checkDeath()
{
    // when health is reduced to 0,
    // player should die and lose a life
    if (health <= 0)
    {
        lives -= 1;

        // if the player runs out of lives
    }
}

```

```

        // then game over, otherwise respawn
        if (lives == 0)
        {
            gameOver();
        }
        else
        {
            Respawn();
        }
        // only checks for game over,
        // each time player dies
    }
}

private void gameOver()
{
    Time.timeScale = 0;
    MainCam.SetActive(false);
    GameOverCam.SetActive(true);
}

public void Respawn()
{
    envGen.NewLevel();
}
}

```

‘playerAttack.cs’

```

using UnityEngine;

public class playerAttack : player
{
    // VARIABLES

    // reference to the player class
    private player player;

    private Animator animator;

    // reference to the camera
    public Transform cam;
}

```

```

// reference to the enemy mask, set in editor
public LayerMask enemyMask;
// multiplier for damage depending on enemy body part hit
private int damageMultiplier;

// reference to the playerMovement script
private playerMovement movement;

// flag for grappling
private bool canGrapple = false;

// reference to player's line renderer component
private LineRenderer lr;
// reference to spring joint component
private SpringJoint joint;

// vector 3 coordinates for grapple point
private Vector3 grapplePoint;
// float value for distance to grapple point
private float grappleDistance;

// min grapple distance
private float minGrapple = 2f;
// max grapple distance
private float maxGrapple = 50f;
// MAINTENANCE - not hard coding these values,
// so they can be changed in the future,
// e.g. if a new developer designs new custom maps,
// they can change these values to improve gameplay

private int i = 0;

void Start()
{
    // references the player script
    player = GetComponent<player>();
    // accesses the line render component of the player
    lr = GetComponent<LineRenderer>();
    // accesses the playerMovement script
    movement = GetComponent<playerMovement>();
    // gets the animator component
    animator = GetComponent<Animator>();
}

```

```

}

void Update()
{
    // if the player left clicks, they should attack/fire
    if (Input.GetMouseButtonDown(0))
    {
        hit();
    }

    // if the player holds left click, should should grapple
    if (Input.GetMouseButton(0))
    {
        Grapple();
    }

    // if player lifts left click, i.e. stops holding
    // they should stop grappling
    else if (Input.GetMouseButtonUp(0))
    {
        StopGrapple();
    }

    if (player.getFuel() < 0)
    {
        player.setFuel(0f);
    }
}

private void hit()
{
    // if weapon is in a fireable state,
    // i.e. not waiting between fire rate
    // then we should fire
    if (player.getWeapon().getCanFire())
    {
        // shoot a ray from player camera, forwards
        Ray hitRay = new Ray (cam.position, transform.forward);
        RaycastHit attackPoint;

        // calls the fire method of weapon,
        player.getWeapon().fire();

        // if the ray hits an object with the enemy mask,
    }
}

```

```

        // and the distance to this object is within the weapon range,
        // the hit should land and player should inflict damage on enemy
        if (Physics.Raycast(hitRay, out attackPoint,
player.getWeapon().getRange(), enemyMask))
    {
        // accesses the body part gameobject of the enemy that was hit
        GameObject enemyPartHit = attackPoint.collider.gameObject;
        // accesses the base enemy script
        enemy enemyHit =
attackPoint.collider.transform.root.GetComponent<enemy>();

        // damage multiplier set according to weak spot level
        if (enemyPartHit.tag == "Weak1")
        {
            damageMultiplier = 1;
        }
        else if (enemyPartHit.tag == "Weak2")
        {
            damageMultiplier = 2;
        }
        else if (enemyPartHit.tag == "Weak3")
        {
            damageMultiplier = 4;
        }

        // calls the damage method of the enemy,
        // and uses the weapons damage attribute, multiplied by the damage
multiplier
        // to reduce enemy health
        enemyHit.takeDamage(player.getWeapon().getDamage() *
damageMultiplier);
        // player gains 5, multiplied by the multiplier, score for a hit
        player.addScore(5 * damageMultiplier);

        // if the enemy is overall weak enough (looking at its health)
        // and a weak spot is hit, this part of the enemy is to be
destroyed
        // due to the hierarchical nature of the enemy prefab,
        // some body parts of the enemy will also be destroyed if their
parent is
        if (enemyPartHit.tag == "Weak3" && enemyHit.getHealth() < 50)
        {
            Destroy(enemyPartHit);
        }
    }
}

```

```

        }
        else if (enemyPartHit.tag == "Weak2" && enemyHit.getHealth() < 30)
        {
            Destroy(enemyPartHit);
        }
    }
}

private void Grapple()
{
    // shoot a ray from player camera, forwards
    Ray grappleRay = new Ray (cam.position, cam.forward);
    RaycastHit hitPoint;

    // if currently can't grapple, should check for grapple
    if (!canGrapple)
    {
        // if the ray hits an object, then we can check further to grapple
        if (Physics.Raycast(grappleRay, out hitPoint) && hitPoint.point != null)
        {
            grapplePoint = hitPoint.point;
            grappleDistance = Vector3.Distance(this.transform.position,
grapplePoint);

            // if the distance to where we are grappling is between the
limits,
            // then we can grapple
            if (grappleDistance > minGrapple && grappleDistance < maxGrapple)
            {
                // only if the player has fuel left, should they grapple
                if (player.getFuel() > 0)
                {
                    canGrapple = true;
                    movement.disableAutoMove();
                    // the automatic movement is disabled
                    // since the physics of the spring joint takes care of
movement
                }
                else
                {
                    canGrapple = false;
                }
            }
        }
    }
}

```

```

        if (i == 0)
        {
            Debug.Log("OUT OF FUEL");
            i = 1;
        }
    }

}

// if we are to grapple,
if (canGrapple)
{
    // adds a spring joint to the player gameobject
    // only if there isn't one already
    if (!joint)
    {
        joint = this.gameObject.AddComponent<SpringJoint>();
    }

    // attaches the spring joint to the grapple point
    joint.autoConfigureConnectedAnchor = false;
    joint.connectedAnchor = grapplePoint;

    // configures joint to liking
    // i.e. responsible for bounciness, length, etc.
    joint.maxDistance = grappleDistance * 0.3f;
    joint.minDistance = grappleDistance * 0.1f;
    joint.spring = 6f;
    joint.damper = 1.2f;
    joint.massScale = 4.5f;

    // sets the line render to have 2 points
    lr.positionCount = 2;

    // decreases fuel by 0.2 for every frame grappled
    player.addFuel(-0.2f);

    Debug.Log("Player fuel level decreased to: " + player.getFuel());
}
}

private void StopGrapple()

```

```

{
    // player should not grapple any longer
    canGrapple = false;
    // player should move automatically again
    movement.enableAutoMove();

    // line should not be rendered any longer
    // so line should have no points
    lr.positionCount = 0;
    // joint should no longer exist
    Destroy(joint);
}

void LateUpdate()
// render the grapple line in LateUpdate() as this runs after update
// this fixes the problem of the the line not being in sync
{
    // VALIDATION
    // only render grapple line if a joint exists
    if (joint)
    {
        // sets the first point of the line to the player's position
        lr.SetPosition(0, this.transform.position);
        // sets the second point of the line to the grapple point
        lr.SetPosition(1, grapplePoint);
    }
}
}

```

'playerMovement.cs'

```

using UnityEngine;

public class playerMovement : player
{

    // VARIABLES

    // reference to the player class
    private player player;

    // camera's transform to be accessed via editor

```

```

public Transform cam;
// the angle the camera is looking at
private float angle;

// min and max angle in which player can look to move
private float minAngle = 30.0f;
private float maxAngle = 330.0f;

// MAINTENANCE - not hard coding these values ^^,
// so they can be changed in the future,
// e.g. if a new developer designs new custom maps,
// they can change these values to improve gameplay

// half of the maximum speed, a constant to be referenced as speed changes
private float halfMaxSpeed = 5f;
// multiplier used to increase speed depending on angle
private float speedMultiplier;

// flag to define whether player should move
public bool toMove = false;

// mouse sensitivity,
// change this value to change how quickly the player looks around
private float sensitivity = 90f;
// not hard coding these values,
// so that they can be changed if they need to be

// amount of rotation to be applied around x-axis
private float verticleRotation = 0f;

// Start function called once, when program first runs
void Start()
{
    // references the player script
    player = GetComponent<player>();
    // makes sure player cursor is locked to center crosshair
    Cursor.lockState = CursorLockMode.Locked;
    // makes sure player cursor is not seen, only crosshair
    Cursor.visible = false;
}

// Update() runs once per frame, therefore dependant on performance

```

```

void Update()
{
    // look() function called for rotation
    look();
    // move() function called for movement
    move();

    #region CONSOLE LOGS
    //Debug.Log("ANGLE: " + angle);
    //Debug.Log("MULTIPLIER: " + speedMultiplier);
    //Debug.Log("SPEED: " + player.getSpeed());
    #endregion
}

// involves input, and as Update() is more frequent than FixedUpdate(),
// player won't feel input delay

private void look()
{
    // depending on how much the mouse is moved, a value between 1 and -1 is
    returned
    // these floats can then be applied to rotate the player accordingly
    float mouseX = Input.GetAxis("Mouse X") * sensitivity * Time.deltaTime;
    float mouseY = Input.GetAxis("Mouse Y") * sensitivity * Time.deltaTime;

    // player rotated in y-axis depending on how much mouse is moved
    // horizontally
    transform.Rotate(Vector3.up * mouseX);

    // use this intermediate variable so that it can be clamped
    // rotation clamped between 90 and -90 so that the player can't look
    behind
    verticleRotation -= mouseY;
    verticleRotation = Mathf.Clamp(verticleRotation, -90f, 90f);

    // camera rotated by verticleRotation in the x-axis and 0 in the y and z
    cam.localRotation = Quaternion.Euler(verticleRotation, 0f, 0f);
    // camera is rotated instead of player so that entire player body does not
}

private void move()
{
    angle = cam.eulerAngles.x;
}

```

```

        toMove = moveCheck(angle);

        // relevant calculations only carried out when player needs to move
        if (toMove)
        {
            angle = calculateAcuteAngle(angle);
            speedMultiplier = calculateMultiplier(angle);

            // multiplier+1 applied to half the player's max speed
            player.setSpeed( halfMaxSpeed * (speedMultiplier+1) );
            // player moved in the direction they are looking by a factor of their
            speed
            this.transform.position += cam.forward * player.getSpeed() *
            Time.deltaTime;
        }
    }

    private bool moveCheck(float cameraAngle)
    {
        // if the camera angle is within min and max range,
        // player can move
        if (cameraAngle < minAngle || cameraAngle > maxAngle)
        {
            return true;
        }
        // otherwise the player should not move
        else
        {
            return false;
        }
    }

    private float calculateAcuteAngle(float cameraAngle)
    {
        if (cameraAngle > maxAngle)
        {
            angle = 360f - cameraAngle;
        }
        // this is the range in which angle is between 360 and 330
        // changes it to a value between 0 and 30
        else if (cameraAngle < minAngle)
        {
            angle = cameraAngle;
        }
    }
}

```

```

        }

        // Will return a number between (0 - 30.0)

        return angle;
    }

private float calculateMultiplier(float cameraAngle)
{
    // multiplier set to 1,
    // if player/camera is looking completely horizontally
    if (angle == 0)
    {
        speedMultiplier = 1f;
    }
    // otherwise, multiplier is calculated as a function of the angle
    // this gives a multiplier between 1 and 0.1
    else
    {
        float k = ( (-0.026444f) * (Mathf.Pow(angle, 2f)) + (0.86333f) * angle +
(1f) );
        speedMultiplier = (k / (angle+1f));
    }

    return speedMultiplier;
}

public void disableAutoMove()
// disables the toMove bool
{
    toMove = false;
}

public void enableAutoMove()
// enables the toMove bool
{
    toMove = true;
}
}

```

'playerJump.cs'

```

using System.Collections;
using UnityEngine;

```

```

public class playerJump : player
// inherits player class to access attributes such as speed
{
    // VARIABLES

    // reference to the player class
    private player player;

    // reference to the camera
    public Transform cam;

    // reference to the player's rigid body component
    private Rigidbody playerBody;

    // bool for either jumping or not jumping
    private bool jump = false;

    // bool for either grounded or not grounded
    private bool grounded = true;

    void Start()
    {
        // reference to the player script
        player = GetComponent<player>();

        // player rigid body component accessed
        playerBody = GetComponent<Rigidbody>();
        // coroutine for jumping started
        // coroutine used as time waits are required
        StartCoroutine(Jump());
    }

    private IEnumerator Jump()
    {
        // while true and wait for fixed update mimic the fixed update method
        // this means that this loop will run at consistent intervals
        // roughly once per frame
        while (true)
        {
            // if not jumping, checks carried out to check for jump
            if (!jump)
            {
                // angle sample taken before wait

```

```

        float startAngle = calculateLinearAngle(cam.eulerAngles.x);
        // program waits for 0.5 seconds
        yield return new WaitForSeconds(0.75f);
        // angle sample taken before half a second wait
        float endAngle = calculateLinearAngle(cam.eulerAngles.x);

        // difference calculated between 2 angle samples
        float angleChange = startAngle - endAngle;

        // ray shot downwards to check how close ground is
        Ray ray = new Ray(transform.position, -transform.up);
        RaycastHit hitInfo;

        // if the length to the intersection point of the ray on the
ground
        // is less than or equal to 3 units,
        // the player can be considered as on the ground
        if (Physics.Raycast(ray, out hitInfo, 3))
        {
            grounded = true;
        }
        // otherwise, player can't be considered as on the ground
        else
        {
            grounded = false;
        }

        // if this difference is greater than 20 (degrees),
        // and player is on the ground,
        // then jump is set to true
        if (endAngle <= 0 && angleChange >= 20f && grounded)
        {
            jump = true;
        }
    }

    // if jump is true, the player is to jump
    if (jump)
    {
        // jump set to false so that the player only jumps once,
        // until jump criteria are met once again
        jump = false;
    }

```

```

        // velocity and position are altered to mimic the art of jumping
        playerBody.velocity += Vector3.up * 15;
        transform.position += cam.forward * -2f * player.getSpeed() *
Time.fixedDeltaTime;
    }

    // waits for next frame before looping
    yield return new WaitForFixedUpdate();
}
}

private float calculateLinearAngle(float cameraAngle)
{
    // if the camera is looking below the horizontal,
    // the following is applied to alter the angle
    if (cameraAngle >= 0 && cameraAngle <= 150f)
    {
        // 30 degrees is added to the camera angle,
        // this means it is now between 30 and 60
        cameraAngle += 30f;
    }

    // if the camera is looking above the horizontal,
    // the following is applied to alter the angle
    else if (cameraAngle > 180f)
    {
        // 330 degrees are taken away from the angle
        // this means that it is now between 0 and 30
        cameraAngle -= 330f;
    }

    // overall, this function returns a value between 0 and 60
    // this is a linear scale so the difference can be calculated
    return cameraAngle;
}
}

```

'playerRefuel.cs'

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```

public class playerRefuel : player
{
    // reference to the player class
    player player;

    // flag to determine standing or not
    private bool standing;

    // flag to determine if the player is allowed to refuel
    private bool canRefuel;

    // reference to the fuel station to be used
    private fuelStation currentRefuelStation;

    // mask to filter out everything except fuel stations
    public LayerMask stationMask;

    void Start()
    {
        // accesses the player script
        player = GetComponent<player>();
        // starts the checkStanding coroutine/procedure
        StartCoroutine(CheckStanding());
    }

    // fixed update so the refuelling is not frame rate dependent
    void FixedUpdate()
    {
        // if the player is standing,
        // which is checked in the checkStanding coroutine,
        // then we check if the player is on a refuel station
        if (standing)
        {
            // a ray is cast from the player, downwards
            Ray StationRay = new Ray(transform.position, -transform.up);
            RaycastHit hitInfo;

            // if the ray hits an object with the fuel station mask,
            // and the collision is <= 5 units from the player,
            // then the player can refuel
    }
}

```

```

        if (Physics.Raycast(StationRay, out hitInfo, 5f, stationMask)) //  

VALIDATION - player must be within 5 units of the station (vertically)
{
    // the refuel station that the player is standing on
    currentRefuelStation =
hitInfo.transform.gameObject.GetComponent<fuelStation>();
    // canRefuel set to true so that the player now refuels
    canRefuel = true;
}
}

// otherwise, if the player is not standing on a station,
// the player should not refuel
else
{
    canRefuel = false;
}

// if the player is to refuel,
if (canRefuel)
{
    // only if the player fuel is not full,
    // and the fuel station is not empty, should the player refuel
    if (player.getFuel() < 100 && !currentRefuelStation.checkEmpty()) //  

VALIDATION - read earlier comments
{
    // player fuel level is increased
    player.addFuel(0.2f);
    // same amount of fuel is decreased from the fuel station
    currentRefuelStation.useFuel(0.2f);

    Debug.Log("Player fuel level increased to: " + player.getFuel());
    Debug.Log("Fuel station fuel level decreased to: " +
currentRefuelStation.getFuelStore());
}
}

// VALIDATION - ensure the player is standing still on the station
public IEnumerator CheckStanding()
{
    // while true loop to keep running while the game plays
    while (true)
{

```

```

        // get player position (before time wait)
        Vector3 posBefore = this.transform.position;
        // wait for 2 seconds
        yield return new WaitForSeconds(1f);
        // get player position (after time wait)
        Vector3 posAfter = this.transform.position;

        // if the player is in the same position,
        // before and after the time wait,
        // then they are standing and so its set to true,
        // otherwise it is set to false
        if (posBefore.x == posAfter.x && posBefore.z == posAfter.z)
        {
            standing = true;
        }
        else
        {
            standing = false;
        }
    }
}

```

'fuelStation.cs'

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class fuelStation : MonoBehaviour
{
    // the station has a fuel store,
    // i.e. how much fuel the station has
    [SerializeField] private float fuelStore = 100f;

    // method to set the fuel level of the station
    private void setStationFuel(float _fuel)
    {
        fuelStore = _fuel;
    }

    // returns the current fuel store
    public float getFuelStore()

```

```

    {
        return fuelStore;
    }

// method to decrease the fuel store,
// by a given amount of fuel used
public void useFuel(float _fuelUsed)
{
    fuelStore -= _fuelUsed;
}

// method to check if fuel exists/
// /is left in the fuel station
public bool checkEmpty()
{
    if (fuelStore > 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}
}

```

‘fuelStationStates.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class fuelStationStates : fuelStation
{
    private fuelStation station;

    void Start()
    {
        station = GetComponent<fuelStation>();
    }

    void Update()
    {
        // if the fuel station has no fuel left,

```

```

    // the fuel station should turn red
    if (station.checkEmpty())
    {
        GetComponent<Renderer>().material.color = Color.red;
    }

    // if the fuel level is between 0 and 50,
    // the fuel station should turn yellow
    else if (station.getFuelStore() > 0 && station.getFuelStore() < 50f)
    {
        GetComponent<Renderer>().material.color = Color.yellow;
    }

    // otherwise, if the fuel level is over 50,
    // the fuel station should be green
    else
    {
        GetComponent<Renderer>().material.color = Color.green;
    }
}

```

‘enemy.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemy : MonoBehaviour
{
    // reference to the environment script
    private EnvironmentGenerator envScript;

    // the enemy's health
    private int health = 100;

    // the enemy's speed
    private float speed = 10f;

    // the enemy's dead or alive status
    private bool dead = false;

    // the enemy's damage stat
    private int damage = 10;
}

```

```

// the enemy's attack range stat
private int attackRange = 15;

// the enemy's follow range stat
private int followRange = 75;

// the enemy's attack delay stat
private int delay;

private void Start()
{
    // finds the environment generator script, so that functions can be used
such as remove enemy
    envScript = GameObject.FindGameObjectWithTag("Environment
Generator").GetComponent<EnvironmentGenerator>();
}

#region GETTERS AND SETTERS for the enemy's attributes
public int getHealth()
{
    return health;
}

public void setHealth(int _health)
{
    health = _health;
}

public float getSpeed()
{
    return speed;
}

public void setSpeed(float _speed)
{
    speed = _speed;
}

public int getDamage()
{
    return damage;
}

```

```

public void setDamage(int _damage)
{
    damage = _damage;
}

public int getAttackRange()
{
    return attackRange;
}

public int getFollowRange()
{
    return followRange;
}

public int getDelay()
{
    return delay;
}

#endifregion

// reduces enemy's health by damage amount
// and checks if enemy is dead
public void takeDamage(int _damage)
{
    health -= damage;

    if (health <= 0)
    {
        dead = true;
    }
    else
    {
        dead = false;
    }
}

public void Die()
{
    // if this enemy is now dead,
    // the enemy count must be decreased
    envScript.removeEnemy();
}

```

```

        Destroy(this.gameObject);
    }

    private void Update()
    {
        if (dead)
        {
            Die();
        }

        Debug.Log("Enemy took damage!");
    }
}

```

'enemyMovement.cs'

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemyMovement : enemy
{
    // reference to the enemy script
    private enemy enemy;

    // reference to the character controller component
    private CharacterController controller;

    // reference to the player
    private GameObject Player;

    // flag for whether the enemy should move towards the player or not
    private bool moveTowardsPlayer = false;

    // flag for whether the enemy should stop moving towards the player
    private bool stop = false;

    // counter for walking before changing direction
    private int moveCounter = 0;

    // flag for whether this is the first time the enemy is within range
    // in which case they should instantly look at the player initially
    private bool initialFindPlayer = true;
}

```

```

// mask to only detect environment objects
public LayerMask environmentObjects;

// mask to only detect other enemies
public LayerMask enemies;

// flag for if the enemy should turn left to avoid obstacles, or right
private bool left = true;

private void Start()
{
    // gets the base enemy script
    enemy = GetComponent<enemy>();
    // finds the player gameobject
    Player = GameObject.FindGameObjectWithTag("Player");
    // gets the character controller component of the enemy
    controller = GetComponent<CharacterController>();
    StartCoroutine(SwitchLeftRight());
}

private IEnumerator SwitchLeftRight()
{
    while (true)
    {
        left = !left;
        yield return new WaitForSeconds(5f);
    }
}

private void Update()
{

    // if the enemy is close enough to the player, i.e. the distance is less
    // than the enemy's follow range
    // and if the enemy is not too close (the enemy should not keep walking
    // towards the player if it's right next to the player)
    // then the enemy should move towards the player and should not stop
    // otherwise, the enemy should not move towards the player
    if ( (Vector3.Distance(Player.GetComponent<Transform>().position,
this.transform.position) < enemy.getFollowRange()) )
    {
        if ( (Vector3.Distance(Player.GetComponent<Transform>().position,
this.transform.position) < 25) )

```

```

{
    stop = true;
}
else
{
    moveTowardsPlayer = true;
    stop = false;
}
}

else
{
    moveTowardsPlayer = false;
    initialFindPlayer = true;
    stop = false;
}

// if the enemy should move towards the player,
if (moveTowardsPlayer && !stop)
{
    Debug.Log("MOVE TOWARDS PLAYER NOW");
    // it should first look at the player
    if (initialFindPlayer == true)
    {
        initialFindPlayer = false;
        lookAtPlayer();
    }

    // then the move function is called,
    // which moves the enemy forwards,
    // dodging any obstacles in the way
    move();

    // if the enemy has moved forwards in that one direction,
    // for 120 frames, then they should look at the player again
    if (moveCounter % 120 == 0 || moveCounter == 0)
    {
        moveCounter = 0;
        lookAtPlayer();
    }
}

// if the enemy is not to move towards the player
// then it should move randomly

```

```

        else if (!moveTowardsPlayer && !stop)
    {
        Debug.Log("MOVE RANDOMLY NOW");
        // the move function is called,
        // which moves the enemy forwards,
        // dodging any obstacles in the way
        move();

        // if the enemy has moves forwards in that one direction
        // for 600 frames, then they should change direction
        if (moveCounter % 600 == 0 || moveCounter == 0)
        {
            // enemy looks at random direction
            moveCounter = 0;
            lookAtRandom();
        }
    }

    public void lookAtPlayer()
    {
        // first the enemy has to look towards the player
        // the whole enemy rotates to look at the player
        this.transform.LookAt(Player.transform);
        // only the y value of this rotation is desired
        float yRot = this.transform.eulerAngles.y;
        // then the rotation of the enemy is set to the desired y rotation
        // x and z components are reset to 0 so that enemy stands upright
        this.transform.eulerAngles = new Vector3 (0, yRot, 0);
    }

    public void lookAtRandom()
    {
        // random y rotation is chosen,
        // and applied to the enemy's rotation
        int yRot = Random.Range(0, 361);
        this.transform.eulerAngles = new Vector3 (0, yRot, 0);
    }

    public void move()
    {
        // a ray is cast from the enemy roughly 18 degrees at an angle of
        depression from the horizontal

```

```

        Ray lineSight = new Ray(this.transform.position, this.transform.forward *
3 - this.transform.up);
        RaycastHit lineSightHit;
        Debug.DrawRay(this.transform.position, (this.transform.forward * 3 -
this.transform.up) * 15, Color.red);

        // if this ray hits an enemy or an environment object, then the enemy
needs to act to avoid the obstacle
        if ( Physics.Raycast(lineSight, out lineSightHit, 100f,
environmentObjects))
        {
            if (Physics.Raycast(lineSight, out lineSightHit, 100f, enemies))
            {
                float currentX = this.transform.eulerAngles.x;
                float currentY = this.transform.eulerAngles.y;
                float currentZ = this.transform.eulerAngles.z;

                // if the enemy is to dodge left, then it rotates by -5 degrees,
                // and this will repeat until the ray is no longer hitting an
obstacle
                if (left)
                {
                    this.transform.eulerAngles = new Vector3 (currentX, currentY -
5, currentZ);
                }
                // otherwise it rotates by +5 degrees, i.e. right, to try and
avoid the obstacle
                else
                {
                    this.transform.eulerAngles = new Vector3 (currentX, currentY +
5, currentZ);
                }
            }
        }

        // then, the enemy moves forward by its speed attribute
        controller.Move(transform.forward * enemy.getSpeed() * Time.deltaTime);
        // move counter is updated so that the main function can check if its time
for the enemy to change direction
        moveCounter += 1;
    }

}

```

'enemyGravity.cs'

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemyGravity : enemy
{
    // reference to the enemy's character controller component
    private CharacterController controller;

    // velocity vector
    private Vector3 velocity;

    // constant value of gravity to calculate velocity at any time
    private float gravity = -9f;

    // bool flag to check whether the enemy is on the floor or not
    private bool grounded;

    void Start()
    {
        // gets the enemy's character controller component
        controller = GetComponent<CharacterController>();
    }

    void Update()
    {
        // a ray is cast from the enemy, downwards
        Ray groundCheck = new Ray(transform.position, -transform.up);
        RaycastHit groundHit;

        // if the ray hits an object,
        // and the collision is <= 5 units from the enemy,
        // then the enemy is grounded
        if (Physics.Raycast(groundCheck, out groundHit, 5f))
            // VALIDATION - enemy must be within 5 units of the floor (vertically)
        {
            grounded = true;
        }
        else
```

```

    {

        grounded = false;
    }

    // if the enemy is grounded, then its velocity is reset to 0
    if (grounded)
    {
        velocity.y = 0f;
    }
    // otherwise, velocity is calculated using the following function
    else
    {
        velocity.y += 0.5f * gravity * Time.deltaTime * Time.deltaTime;
    }

    // enemy is moved by velocity vector
    controller.Move(velocity);
}

}

```

‘hitDetect1.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class hitDetect1 : enemy
{
    // bool for if the player is colliding
    // with this trigger collider
    private bool hit1;

    // bool for if this collider has been initialised
    private bool isActive = false;

    public bool getHit1()
    {
        // returns the value of hit1
        // if the collider is initialised
        // otherwise, false
        if (isActive)
        {
            return hit1;
        }
    }
}

```

```

        else
    {
        return false;
    }
}

private void OnTriggerEnter(Collider other)
{
    // if this collider collides with the player
    // then hit1 is set to true
    if (other.tag == "Player")
    {
        isActive = true;
        Debug.Log("HIT 1");
        hit1 = true;
    }
}

private void OnTriggerExit(Collider other)
{
    // if the player leaves this collider,
    // then hit1 is reset to false
    if (other.tag == "Player")
    {
        isActive = true;
        Debug.Log("HIT 1 OFF");
        hit1 = false;
    }
}
}

```

‘hitDetect2.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class hitDetect2 : enemy
{
    // bool for if the player is colliding
    // with this trigger collider
    private bool hit2 = false;

    // bool for if this collider has been initialised

```

```

private bool isActive = false;

public bool getHit2()
{
    // returns the value of hit2
    // if the collider is initialised
    // otherwise, false

    if (isActive)
    {
        return hit2;
    }
    else
    {
        return false;
    }
}

private void OnTriggerEnter(Collider other)
{
    // if this collider collides with the player
    // then hit2 is set to true
    if (other.tag == "Player")
    {
        isActive = true;
        Debug.Log("HIT 2");
        hit2 = true;
    }
}

private void OnTriggerExit(Collider other)
{
    // if the player leaves this collider,
    // then hit2 is reset to false
    if (other.tag == "Player")
    {
        isActive = true;
        Debug.Log("HIT 2 OFF");
        hit2 = false;
    }
}

```

'enemyAttack.cs'

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemyAttack : enemy
{
    // reference to the base enemy script
    private enemy enemy;

    // reference to the base player script
    private player player;

    // reference to the hit1 and hit2 bools
    // used to decide whether player should be attacked
    private bool hit1;
    private bool hit2;

    // reference to the collider gameobjects
    private GameObject hd1;
    private GameObject hd2;

    // multiplier to slow the enemy when attacking
    // must be less than 1
    private float slowFactor = 0.75f;

    // bool for whether the enemy is currently slowed
    private bool slowed = false;

    private void Start()
    {
        // accessing the base enemy and player scripts
        enemy = GetComponent<enemy>();
        player = GameObject.FindGameObjectWithTag("Player").GetComponent<player>();

        // accessing the collider gameobjects
        hd1 = GameObject.FindGameObjectWithTag("HITDETECT1");
        hd2 = GameObject.FindGameObjectWithTag("HITDETECT2");

        StartCoroutine(attackLoop());
    }
}
```

```

private IEnumerator attackLoop()
{
    // constant loop which loops every fixed frame
    while (true)
    {
        // gets the current status of the hit1 bool
        hit1 = hd1.GetComponent<hitDetect1>().getHit1();
        //Debug.Log("HIT 1: " + hit1);

        // if the player is in range of hit1,
        // then the enemy is slowed and the time wait begins
        if (hit1)
        {
            if (!slowed)
            {
                enemy.setSpeed(enemy.getSpeed() * slowFactor);
                slowed = true;
            }

            yield return new WaitForSeconds(1f);
            Debug.Log("WAITED FOR 1 SEC");
            yield return new WaitForSeconds(1f);
            Debug.Log("WAITED FOR 2 SECS");
            yield return new WaitForSeconds(1f);
            Debug.Log("WAITED FOR 3 SECS");

            // gets the current status of the hit2 bool
            hit2 = hd2.GetComponent<hitDetect2>().getHit2();
            //Debug.Log("HIT 2: " + hit2);

            // if the player is still in range of hit2,
            // even after the time wait, then the player takes damage
            // and the enemy undergoes a time wait before it can attack again
            if (hit2)
            {
                Debug.Log("HIT PLAYER NOW");
                player.takeDamage(enemy.getDamage());
                Debug.Log("Wait for 5 secs now");
                yield return new WaitForSeconds(5f);
            }
        }
    }

    // if the player is not within either collider region
}

```

```

        // then the enemy's speed is reset
    else
    {
        if (slowed)
        {
            enemy.setSpeed(enemy.getSpeed() / slowFactor);
            slowed = false;
        }
    }
    yield return new WaitForFixedUpdate();
}
}
}

```

'EnvironmentGenerator.cs'

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class EnvironmentGenerator : MonoBehaviour
{
    private bool newLevelAllowed = true;

    #region ENVIRONMENT VARS

    // game object list of all spawned environment objects
    // used to destroy these when a new level is generated
    private List<Object> allEnvObjects = new List<Object>();

    // string array of length 4 storing the different biome names
    private string[] biomes = new string[4] {"town", "forest", "snow", "desert"};

    // string for the chosen biome
    private string chosenBiome;

    // array of gameobjects to be randomly spawned into the level
    private GameObject[] envFeatures;

    // colour for the mesh to be set to depending on biome
    private Color groundColor;

    // amount of space between each environment feature spawned
    private int spaceBetween;
}

```

```

// amount of space above the ground each environment feature should be spawned
private int groundOffset;

// references to all model prefabs used to populate scene with environment
objects

#region town
public GameObject house_1;
public GameObject house_2;
public GameObject house_3;
public GameObject house_4;
public GameObject house_5;
public GameObject townhall;
#endregion

#region forest
public GameObject tree_1;
public GameObject tree_2;
public GameObject tree_3;
public GameObject bush_1;
public GameObject bush_2;
public GameObject branch;
public GameObject boulder;
public GameObject stump;
#endregion

#region snow
public GameObject snowtree_1;
public GameObject snowtree_2;
public GameObject snowtree_3;
#endregion

#region desert
public GameObject giantcacti_1;
public GameObject giantcacti_2;
public GameObject giantcacti_3;
public GameObject giantcacti_4;
#endregion

#endregion

#region FUEL STATION VARS

```

```

// reference to the fuel station model so that it can be spawned
public GameObject fuelStation;

// list of all spawned fuel stations in current level
private List<Object> allFuelStations = new List<Object>();

// layer mask to only detect the environment
public LayerMask environmentMask;

// layer mask to detect only other fuel stations
public LayerMask stationMask;

// flag to check if there is an object directly below a point
private bool objectBelow;

// current position of fuel station before spawning
private Vector3 stationPos;
#endregion

#region ENEMY VARS
// reference to the enemy model so that they can be spawned
public GameObject enemy;

// list of all spawned enemies in current level
private List<Object> allEnemies = new List<Object>();

// number of alive enemies
private int enemyCount = 1;
#endregion

#region PLAYER VARS
// reference to the player prefab so that player can be spawned
public GameObject player;

// variable to store the player that should be deleted, before it is
private GameObject oldPlayer;
#endregion

#region MESH GENERATION VARS
// the mesh that we will configure
private Mesh mesh;

// array which will contain generated vertices

```

```

private Vector3[] vertices;

// array which will contain generated triangles
private int[] triangles;

// length of terrain mesh
private int xSize = 200;

// width of terrain mesh
private int zSize = 200;

// scale for perlin noise
private float noiseScale = 0.1f;
#endifregion

void Awake()
{
    // calls this function when
    // this script is first instantiated
    LevelGenerator();
}

private void LevelGenerator()
{
    // generates the floor
    GenerateMesh();
    // randomly decides which biome will be used
    DecideBiome();
    // randomly spawns relevant environment features
    GenerateEnvironment();
    // randomly spawns fuel station across the map
    GenerateFuelStations();
    // randomly spawns enemies across the map
    SpawnEnemies();
    // spawns the player at a random position
    SpawnPlayer();
}

private void GenerateMesh()
{
    // instantiates a new mesh
    mesh = new Mesh();
}

```

```

GetComponent<MeshFilter>().mesh = mesh;

// instantiates a new array storing vertices
vertices = new Vector3[(xSize+1)*(zSize+1)];

// sets vertices to x, z values with random y value
for (int i = 0, z = 0; z <= zSize; z++)
{
    for (int x = 0; x <= xSize; x++)
    {
        // y value calculated by layering perlin noise multiple times,
        // on top of itself and then layering it with the sin
        float y1 = Mathf.PerlinNoise(x * noiseScale, z * noiseScale) * 2f;
        float y2 = Mathf.PerlinNoise(x * noiseScale, z * noiseScale) * 2f;
        float y = Mathf.Sin(Mathf.PerlinNoise(y1, y2));
        vertices[i] = new Vector3(x, y, z);
        i++;
    }
}

// instantiates a new array of triangles for mesh
triangles = new int[xSize*zSize*6];

// sets triangles using vertices
int vert = 0;
int tris = 0;
for (int z = 0; z < zSize; z++)
{
    for (int x = 0; x < xSize; x++)
    {
        triangles[0 + tris] = vert + 0;
        triangles[1 + tris] = vert + xSize + 1;
        triangles[2 + tris] = vert + 1;
        triangles[3 + tris] = vert + 1;
        triangles[4 + tris] = vert + xSize + 1;
        triangles[5 + tris] = vert + xSize + 2;

        vert++;
        tris += 6;
    }
    vert++;
}

```

```

        }

        // clears any previous mesh
        mesh.Clear();

        // sets the vertices of the mesh to the ones we generated
        mesh.vertices = vertices;
        // sets the vertices of the mesh to the ones we generated
        mesh.triangles = triangles;

        // updates mesh attributes for various functionalities,
        // such as showing textures, adding a collider, etc
        mesh.RecalculateNormals();
        mesh.RecalculateBounds();
        MeshCollider meshCollider = gameObject.GetComponent<MeshCollider>();
        meshCollider.sharedMesh = mesh;
    }

    private void DecideBiome()
    {
        // random number between 0 and the amount of biomes
        int index = Random.Range(0, biomes.Length);
        // this number used to choose random biome via index
        chosenBiome = biomes[index];

        // then depending on the biome,
        // envFeatures is populated with the relevant gameobjects
        // groundColour, spaceBetween and groundOffset are configured

        if (chosenBiome == "town")
        {
            envFeatures = new GameObject[6] {house_1, house_2, house_3, house_4,
house_5, townhall};
           groundColor = Color.grey;
            spaceBetween = 80;
            groundOffset = 0;
        }

        else if (chosenBiome == "forest")
        {
            envFeatures = new GameObject[8] {tree_1, tree_2, tree_3, bush_1,
bush_2, branch, boulder, stump};
           groundColor = new Color32(63, 120, 84, 251);
        }
    }
}

```

```

        spaceBetween = 25;
        groundOffset = 0;
    }

    else if (chosenBiome == "snow")
    {
        envFeatures = new GameObject[3] {snowtree_1, snowtree_2, snowtree_3};
       groundColor = Color.white;
        spaceBetween = 50;
        groundOffset = 0;
    }

    else if (chosenBiome == "desert")
    {
        envFeatures = new GameObject[4] {giantcacti_1, giantcacti_2,
giantcacti_3, giantcacti_4};
       groundColor = Color.yellow;
        spaceBetween = 50;
        groundOffset = 12;
    }
    else
    {
        // this instantiates the list,
        // so that from a logical perspective,
        // it is always defined,
        // before it is referenced in the generate environment function
        envFeatures = new GameObject[0] {};
    }
}

private void GenerateEnvironment()
{
    // sets the render's colour attribute to the required ground colour
    GetComponent<Renderer>().material.color = groundColor;

    // loops through the width of the level,
    // incrementing with the required space between objects
    for (int x = 20; x <= 180; x += spaceBetween)
    {
        // loops through the length of the level,
        // incrementing with the required space between objects
        for (int z = 20; z <= 180; z += spaceBetween)
        {

```

```

        // random number between 0 and the amount of environment features
        int randomint = Random.Range(0, envFeatures.Length);
        // this number used to choose random environment feature via index
        GameObject currentFeature = envFeatures[randomint];
        // the position of this environment feature at this x, z position,
        // with height equal to the required ground offset
        Vector3 envPos = new Vector3(x, 1 + groundOffset, z);
        // the rotation of this environment feature with 0 x, z rotation,
        // with a random y-axis rotation between 0 and 360 degrees
        Quaternion envRotation = Quaternion.Euler(new Vector3(0,
Random.Range(0, 360), 0));
        // spawns the environment feature at this position and rotation
        // and adds this to the list of all environment objects
        var newEnvObject = Instantiate(currentFeature, envPos,
envRotation);
        newEnvObject.transform.parent = this.gameObject.transform;
        allEnvObjects.Add(newEnvObject);
    }
}

private void GenerateFuelStations()
{
    Vector3 randomPos;
    Ray groundCheck;

    // loops 5 times to spawn 5 fuel stations
    for (int i = 0; i < 5; i++)
    {
        objectBelow = true;
        // while the current position where the fuel station is to be placed
        // has an object below it, choose a new random position
        while (objectBelow)
        {
            // generates new x-z random position
            int randx = Random.Range(20, 180);
            int randz = Random.Range(20, 180);
            randomPos = new Vector3 (randx, 1000, randz);

            // cast ray down to check for objects below
            groundCheck = new Ray(randomPos, -transform.up);
            RaycastHit groundHit;

```

```

        // check if there is an object below and set the bool flag
accordingly

        if (Physics.Raycast(groundCheck, out groundHit, environmentMask))
        {
            objectBelow = true;
        }
        else if (Physics.Raycast(groundCheck, out groundHit, stationMask))
        {
            objectBelow = true;
        }
        else
        {
            objectBelow = false;
            stationPos = randomPos;
            stationPos.y = 2;
        }
    }

    // fuel station is instantiated at this random position,
    // and is appended to the list of all fuel stations
    allFuelStations.Add(Instantiate(fuelStation, stationPos,
Quaternion.identity));
    objectBelow = true;
    Debug.Log("Station: " + stationPos);
}
}

private void SpawnEnemies()
{
    // spawns enemies across a number of rows and columns
    for (int x = 20; x <= 180; x += 80)
    {
        for (int z = 20; z <= 180; z += 80)
        {
            if (enemyCount < 10)
            {
                // Instanciates enemy at this position
                // and adds it to list of all enemies
                allEnemies.Add(Instantiate(enemy, new Vector3(x, 15, z),
Quaternion.identity));
                enemyCount++;
            }
        }
    }
}

```

```

}

private void SpawnPlayer()
{
    // chooses a random x and z value for where the player should spawn
    int randx = Random.Range(20, 180);
    int randz = Random.Range(20, 180);
    Vector3 randomPos = new Vector3 (randx, 20, randz);
    // spawns player by instantiating the player prefab
    Instantiate(player, randomPos, Quaternion.identity);
}

public void removeEnemy()
{
    // Decreases enemy count by 1
    enemyCount--;
    Debug.Log("ENEMY COUNT: " + enemyCount);
}

private void Update()
{
    //Debug.Log("Enemies remaining: " + enemyCount);

    // if there are no enemies left,
    // and if a new level is not currently being created,
    // then a new level should start being created
    if (enemyCount <= 0 && newLevelAllowed)
    {
        NewLevel();
    }
}

public void NewLevel()
{
    // ensures that a new level generation,
    // is not start until this one is finished
    newLevelAllowed = false;
    // destroys all previous level objects
    DestroyPrev();
    // generates the floor
    GenerateMesh();
    // randomly decides which biome will be used
    DecideBiome();
}

```

```

    // randomly spawns relevant environment features
    GenerateEnvironment();
    // randomly spawns fuel station across the map
    GenerateFuelStations();
    // randomly spawns enemies across the map
    SpawnEnemies();
    // spawns the player at a random position
    SpawnPlayer();
    // this level has now been generated,
    // so a new level is now allowed
    newLevelAllowed = true;
}

private void DestroyPrev()
{
    // destroy the current terrain mesh
    Destroy(mesh);
    // destroy all current environment features
    for (int i = 0; i < allEnvObjects.Count; i++)
    {
        Destroy(allEnvObjects[i]);
    }
    // destroy all current fuel stations
    for (int i = 0; i < allFuelStations.Count; i++)
    {
        Destroy(allFuelStations[i]);
    }
    // destroy all current remaining enemies
    for (int i = 0; i < allEnemies.Count; i++)
    {
        Destroy(allEnemies[i]);
    }
    // destroy player as new one will be spawned
    oldPlayer = GameObject.FindGameObjectWithTag("Player");
    Destroy(oldPlayer);
}
}

```

'Sun.cs'

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```

public class Sun : MonoBehaviour
{
    // the target which you rotate around
    public GameObject target;

    // the speed at which you rotate, around the target
    public int speed = 1;

    // the x,y,z values of the direction that this object rotates around
    public float x = 1f;
    public float y = 1f;
    public float z = 0f;

    void Update()
    {
        // rotates around target in the direction x,y,z with speed, speed
        transform.RotateAround(target.transform.position, new Vector3(x, y, z),
speed * Time.deltaTime);
    }
}

```

‘Weapon.cs’

```

using System.Collections;
using UnityEngine;

public class Weapon : MonoBehaviour
{
    #region attributes
    // weapon damage attribute
    private int damage;

    // weapon fire range attribute
    private int range;

    // weapon fire delay attribute
    private float delay;

    // bool flag for whether weapon is in a state,
    // where it can fire or not (during delay it can't)
    private bool canFire = true;

    // the projectile, e.g. bullet, shot by this weapon
    private GameObject projectile;
}

```

```

// position which the projectile is fired from
private Transform projectilePos;

#endregion

#region methods
// GETTERS AND SETTERS
public void setDamage(int _damage)
{
    damage = _damage;
}

public void setRange(int _range)
{
    range = _range;
}

public void setDelay(float _delay)
{
    delay = _delay;
}

public int getDamage()
{
    return damage;
}

public float getDelay()
{
    return delay;
}

public int getRange()
{
    return range;
}

public bool getCanFire()
{
    return canFire;
}

```

```

public GameObject getProjectile()
{
    return projectile;
}

public void setProjectile(GameObject _projectile)
{
    projectile = _projectile;
}

public Transform getProjectilePos()
{
    return projectilePos;
}

public void setProjectilePos(Transform _projectilePos)
{
    projectilePos = _projectilePos;
}
#endifregion

// method to fire the weapon
// alters can fire bool flag, spawns and fires projectile (if needed)
public void fire()
{
    Debug.Log("FIRE!");
    canFire = false;

    // projects the projectile
    projectile = Instantiate(projectile, projectilePos.position,
Quaternion.identity);

    projectile.GetComponent<Rigidbody>().AddForce(GameObject.FindWithTag("MainCamera")
.transform.forward * 10000);
}

private void Start()
{
    // starts the coroutine initially
StartCoroutine(FireDelay());
}

private IEnumerator FireDelay()

```

```

{
    // loops imitating the fixed update method,
    // loops at fixed intervals while the game runs
    while (true)
    {
        // if the weapon has been fired,
        // canFire will have been set to false,
        // it should be reset to true (able to fire again),
        // after waiting the amount of time the fire delay is
        if (!canFire)
        {
            yield return new WaitForSeconds(delay);
            canFire = true;
        }

        yield return new WaitForFixedUpdate();
    }
}
}

```

‘setSword.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class setSword : MonoBehaviour
{
    // reference to the base weapon script
    private Weapon weaponBase;

    // reference to the base player script
    private player player;

    // when the script is initially run,
    // the values should be set
    private void Awake()
    {
        setter();
    }

    // when the script is re-enabled,
    // the values should be set

```

```

private void OnEnable()
{
    setter();
}

private void setter()
{
    // gets the base weapon script
    weaponBase = GetComponent<Weapon>();

    // gets the base player script
    player = GameObject.FindGameObjectWithTag("Player").GetComponent<player>();

    // sets attribute values according to what they should be for a sword
    weaponBase.setDamage(30);
    weaponBase.setDelay(1.5f);
    weaponBase.setRange(20);
    weaponBase.setProjectile(null);
    weaponBase.setProjectilePos(null);

    // sets this weapon script as the player's chosen weapon
    player.setWeapon(weaponBase);
}
}

```

'setSniper.cs'

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class setSniper : MonoBehaviour
{
    // reference to the base weapon script
    private Weapon weaponBase;
    // reference to the base player script
    private player player;
    // reference to the bullet projectile
    public GameObject projectile;
    // reference to the gun tip,
    // from where the bullet will be spawned and shot
    public GameObject gunTip;

    // when the script is initially run,
    // the values should be set
}

```

```

private void Awake()
{
    setter();
}

// when the script is re-enabled,
// the values should be set
private void OnEnable()
{
    setter();
}

private void setter()
{
    // gets the base weapon script
    weaponBase = GetComponent<Weapon>();
    // gets the base player script
    player = GameObject.FindGameObjectWithTag("Player").GetComponent<Player>();

    // sets attribute values according to what they should be for a sniper
    weaponBase.setDamage(70);
    weaponBase.setDelay(3f);
    weaponBase.setRange(150);
    weaponBase.setProjectile(projectile);
    weaponBase.setProjectilePos(gunTip.transform);

    // sets this weapon script as the player's chosen weapon
    player.setWeapon(weaponBase);
}
}

```

‘weaponMenu.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class weaponMenu : MonoBehaviour
{
    // reference to the player gameobject
    public GameObject playerObject;

```

```

// reference to the base player script
private player player;

// reference to the weapon selected text
public GameObject weaponSelection;
// reference to the text component of the above object
private Text text;

// reference to the sword weapon
public GameObject sword;
// reference to the sniper weapon
public GameObject sniper;

private void Start()
{
    // gets all the components
    text = weaponSelection.GetComponent<Text>();
    player = playerObject.GetComponent<player>();
    sniper = GameObject.FindWithTag("Sniper");
    sword = GameObject.FindWithTag("Sword");
}

public void selectSniper()
{
    // updates the text to show that sniper is selected
    text.text = "Current Selection: SNIPER";
    // activates the sniper and disables the sword
    sniper.SetActive(true);
    sword.SetActive(false);
}

public void selectSword()
{
    // updates the text to show that sniper is selected
    text.text = "Current Selection: SWORD";
    // activates the sword and disables the sniper
    sniper.SetActive(false);
    sword.SetActive(true);
}

```

```
}
```

'Paused.cs'

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Paused : MonoBehaviour
{
    // reference to the main camera
    // which is used in gameplay
    private GameObject MainCam;
    // reference to the pause camera
    // which is used when paused
    private GameObject PauseCam;

    // reference to the pause canvas
    // which is displayed when paused
    private GameObject PauseCanvas;
    // reference to the weapons canvas
    // for when the player is choosing weapon
    private GameObject WeaponsCanvas;

    private void Start()
    {
        // gets the main camera
        MainCam = GameObject.FindGameObjectWithTag("MainCamera");
        // gets the pause camera
        PauseCam = GameObject.FindGameObjectWithTag("Pause Camera");

        // gets the pause canvas
        PauseCanvas = GameObject.FindGameObjectWithTag("Pause Canvas");
        // gets the weapons canvas
        WeaponsCanvas = GameObject.FindGameObjectWithTag("Weapons
Canvas");
    }
}
```

```

public void Pause()
{
    // pausing should stop all the movement of the player and enemies
etc.

    // this is done by reducing the time scale to 0
    Time.timeScale = 0;

    // main camera disabled
    // pause camera enabled instead
    MainCam.SetActive(false);
    PauseCam.SetActive(true);

    // pause canvas shown
    // weapons canvas hidden for now
    PauseCanvas.SetActive(true);
    WeaponsCanvas.SetActive(false);
}

public void Weapons()
{
    // weapons canvas shown
    // pause canvas hidden
    PauseCanvas.SetActive(false);
    WeaponsCanvas.SetActive(true);
}

public void MainMenu()
{
    // finds the main menu scene and sets it as the current scene
    Scene mainMenuScene = SceneManager.GetSceneByName("Menu");
    SceneManager.LoadScene("Menu", LoadSceneMode.Single);
    SceneManager.SetActiveScene(mainMenuScene);
}

public void Unpause()
{
    // time scale reset to 1 to unpause
    Time.timeScale = 1;
}

```

```

        // main camera re-enabled so player can play
        // pause camera disabled to hide pause UI
        MainCam.SetActive(true);
        PauseCam.SetActive(false);
    }
}

```

‘playAgain.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class playAgain : MonoBehaviour
{
    // reference to the base player script
    private player player;

    private void Start()
    {
        // gets the player script
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<player>();
    }

    public void PlayAgain()
    {
        // resets the player's score to 0
        player.addScore(-player.getScore());
        // calls the player respawn function
        player.Respawn();
    }
}

```

‘score.cs’

```

using System.Collections;
using System.Collections.Generic;

```

```

using UnityEngine;
using UnityEngine.UI;

public class score : MonoBehaviour
{
    // reference to the score text
    private Text scoreText;
    // reference to the base player script
    private player player;

    private void Start()
    {
        // gets the player script
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<player>();
    }

    public void showText()
    {
        // updates the text to show the player's score
        scoreText.text = "Score = " + player.getScore();
    }
}

```

‘HealthFuel.cs’

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HealthFuel : MonoBehaviour
{
    // reference to the base player script
    private player player;

    // reference to the health slider
    private Slider healthSlider;

```

```

    // reference to the fuel slider
    private Slider fuelSlider;

    private void Start()
    {
        // gets the player script
        player =
GameObject.FindGameObjectWithTag("Player").GetComponent<player>();

        // gets the health slider
        healthSlider = GameObject.FindGameObjectWithTag("Health
Slider").GetComponent<Slider>();

        // gets the fuel slider
        fuelSlider = GameObject.FindGameObjectWithTag("Fuel
Slider").GetComponent<Slider>();

    }

    private void Update()
    {
        // updates both the health and fuel to current value
        healthSlider.value = player.getHealth();
        fuelSlider.value = player.getFuel();
    }
}

```

‘AttackOnTheUnknown.html’

```

<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

```

```

<body>
    <!-- page heading -->
    <h1 class=main>ATTACK ON THE UNKNOWN</h1>

    <!-- about page button -->
    <div class=main>
        <!-- link to about page -->
        <a href="about.html">
            <button>
                <!-- button label -->
                ABOUT
            </button>
        </a>
    </div>

    <br>

    <!-- download page button -->
    <div class=main>
        <!-- link to download page -->
        <a href="download.html">
            <button>
                <!-- button label -->
                DOWNLOAD
            </button>
        </a>
    </div>

    <br>

    <!-- feedback form page button -->
    <div class=main>
        <!-- link to feedback form page -->
        <a href="feedbackForm.html">
            <button>
                <!-- button label -->
                FEEDBACK FORM
            </button>
        </a>
    </div>

```

```

</div>

<br> <br> <br>

<!-- toggle theme button -->
<div class=main>
    <!-- link to js function called by button -->
    <button style="height: 120px; width:80%;" onclick=swapTheme()>
        <!-- button label -->
        Toggle theme
    </button>
</div>

<br>

</body>
</html>

```

‘about.html’

```

<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

    <body>
        <!-- page heading -->
        <h1>ABOUT</h1>

        <!-- list of information -->
        <div class=region>
            <div class=section>

```

```

<h2>What is Attack on the unknown?</h2>
<ul>
    <li>Attack on the Unknown is a cardboard VR mobile game</li>
        <li>Defeat the giants with your sword or sniper!</li>
        <li>Lots of biomes to explore!</li>
        <li>Levels are randomly generated, the game will never end...</li>
    </ul>
</div>
</div>

<br>

<!-- group of images -->
<div class=region>
    <div class=section>
        <h3>Gallery</h3>
        
        
        
        
        
        
    </div>
</div>

<br>
<!-- list of supported devices -->
<div class=region>
    <div class=section>
        <h3>Supported Devices</h3>

```

```

        <ul>
            <li>IOS mobile devices</li>
            <li>Android mobile devices</li>
            <li>Windows devices (alternate PC version of the
game)</li>
        </ul>
    </div>
</div>

<br>

<!-- help and support information -->
<div class=region>
    <div class=section>
        <h3>Help and support</h3>
        <ul>
            <li>Use the feedback form to get in touch or provide
any feedback</li>
            <li>The feedback form page can be found on the home
page</li>
        </ul>
    </div>
</div>

<br><br>

<!-- toggle theme button -->
<div class=main>
    <!-- link to js function called by button -->
    <button id="smallButton" onclick=swapTheme()>
        <!-- button label -->
        Toggle theme
    </button>
</div>

<br>

</body>
</html>

```

'download.html'

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

    <body>
        <!-- page heading -->
        <h1>DOWNLOAD</h1>

        <br>

        <div class="main">
            <!-- downloadable resource location -->
            <a href="C:\Users\harsh\programmingProjects\Unity
games\BUILDS\AOTU.exe" download>
                <!-- download button -->
                <button id=smallButton>
                    <!-- button label -->
                    Click here to download Attack on the Unknown
                </button>
            </a>
        </div>

        <br><br><br>

        <!-- toggle theme button -->
        <div class=main>
            <!-- link to js function called by button -->
            <button id="smallButton" onclick=swapTheme ()>
                <!-- button label -->
```

```

        Toggle theme
    </button>
</div>

<br>

</body>
</html>

```

'feedbackForm.html'

```

<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <!-- title -->
        <title>AOTU</title>
        <!-- link to javascript file -->
        <script src="scriptAOTU.js"></script>
        <!-- link to css file -->
        <link id="CSS" rel="stylesheet" href="darkAOTU.css">
    </head>

    <body>
        <!-- page heading -->
        <h1>Feedback form</h1>

        <div class=region>
            <!-- form of questions (for feedback) -->
            <!-- sends the answers to actionPage.php to send responses to
my email -->
            <form action="actionPage.php" method="post"
style="padding-left: 20px;">

                <!-- text input -->
                <p><strong>Name:</strong></p>
                <input type="text" name="name" value="Enter name">

                <!-- email input -->

```

```

<p><strong>Email address</strong></p>
<input type="email" name="age" value="Enter email
address">

<br>
<br>
<br>

<!-- radio input -->
<p class=default><strong>Select your age
range:</strong></p>
<p class=default>18-20<input type="radio"
value="18-20"></p>
<p class=default>20-25<input type="radio" name="type"
value="20-25"></p>
<p class=default>25-40<input type="radio" name="type"
value="25-40"></p>
<p class=default>40+<input type="radio" name="type"
value="40+"></p>

<br>
<br>

<!-- radio input -->
<p class=default><strong>Choose the feature which you liked
most about the game:</strong></p>
<p class=default>Fast-paced gameplay<input type="radio"
name="type" value="Fast-paced gameplay"></p>
<p class=default>Random level generation<input
type="radio" name="type" value="Random level generation"></p>
<p class=default>Weapons<input type="radio" name="type"
value="Weapons"></p>
<p class=default>Enemies<input type="radio" name="type"
value="Enemies"></p>
<p class=default>UI<input type="radio" name="type"
value="UI"></p>
<p class=default>Art<input type="radio" name="type"
value="Art"></p>

```

```

<br>
<br>

<!-- radio input -->
<p class=default><strong>Choose the feautes which you
liked least about the game:</strong></p>
<p class=default>Fast-paced gameplay<input type="radio"
name="type" value="Fast-paced gameplay"></p>
<p class=default>Random level generation<input
type="radio" name="type" value="Random level generation"></p>
<p class=default>Weapons<input type="radio" name="type"
value="Weapons"></p>
<p class=default>Enemies<input type="radio" name="type"
value="Enemies"></p>
<p class=default>UI<input type="radio" name="type"
value="UI"></p>
<p class=default>Art<input type="radio" name="type"
value="Art"></p>

<br>
<br>

<!-- textbox input -->
<p><strong>Any other feedback or suggestions?</strong></p>
<input type="text" name="comments" value="">

<br>
<br>
<br>
<br>
<br>
<br>

<!-- form reset button -->
<input type="reset" value="RESET">
<br><br>
<!-- form submit button -->
<input type="submit" value="SUBMIT">
<br><br>
```

```

        </form>
    </div>

<br><br><br>

<!-- toggle theme button -->
<div class=main>
    <!-- link to js function called by button -->
    <button id="smallButton" onclick=swapTheme()>
        <!-- button label -->
        Toggle theme
    </button>
</div>

<br>

</body>
</html>

```

'darkAOTU.css'

```

body{
    background-color:rgb(20, 20, 20);
    padding-left: 10px;
    padding-right: 10px;
}

h1{
    font-size: 50px;
    color: whitesmoke;
}

h2, h3{
    color: white;
}

p, li{
    font-family: Arial, Helvetica, sans-serif;
}

```

```

    font-size: 18px;
    color:rgb(206, 206, 206);
    padding-left: 5px;
    padding-right: 10px;
    padding-bottom: 5px;
}

button{
    background-color:rgb(111, 28, 219);
    padding: 15px 32px;
    width: 90%;
    height: 200px;
    display:block;

    border:rgb(111, 28, 219);
    border-radius: 8px;

    color: white;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 35px;
    font-family:Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;

    transition-duration: 0.4s;
}

button:hover{
    background-color:rgb(20, 20, 20) ;
    color: white;
    border: 5px solid rgb(111, 28, 219);
}

input{
    background-color:rgb(0, 0, 0);
    color: white
}

.region{

```

```

        background-color: rgb(46, 0, 153);
        padding: 5px;
    }

.section{
    background-color: rgb(37, 37, 37);
    padding: 10px;
}

.main{
    text-align: center;
    font-family:Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
}

.default{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    color:rgb(206, 206, 206);
}

.row{
    background-color:rgb(20, 20, 20) ;
}

#smallButton{
    height: 100px;
}

```

‘lightAOTU.css’

```

body{
    background-color:rgb(255, 255, 255);
    padding-left: 10px;
    padding-right: 10px;
}

h1{
    font-size: 50px;
    color: rgb(0, 71, 165);
}

```

```

}

h2, h3{
    color: white;
}

p, li{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    color:rgb(255, 255, 255);
    padding-left: 5px;
    padding-right: 10px;
    padding-bottom: 5px;
}

button{
    background-color:rgb(0, 110, 255);
    padding: 15px 32px;
    width: 90%;
    height: 200px;
    display:block;

    border:rgb(111, 28, 219);
    border-radius: 8px;

    color: rgb(255, 255, 255);
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 35px;
    font-family:Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;

    transition-duration: 0.4s;
}

button:hover{
    background-color:rgb(255, 255, 255) ;
    color: rgb(0, 71, 165);
}

```

```

        border: 5px solid rgb(0, 71, 165);
    }

.region{
    background-color: rgb(46, 0, 153);
    padding: 5px;
}

.section{
    background-color: rgb(0, 110, 255);
    padding: 10px;
}

.main{
    text-align: center;
    font-family: Impact, Haettenschweiler, 'Arial Narrow Bold', sans-serif;
}

.default{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    color:rgb(255, 255, 255);
}

#smallButton{
    height: 100px;
}

```

‘scriptAOTU.js’

```

function swapTheme()
{
    // gets the current CSS file name
    var currentTheme =
document.getElementById("CSS").getAttribute("href");
    // depending on which file it is (light file or dark file)
    // the alternate is set
    // therefore theme swapped between light and dark or visa versa
    if (currentTheme == "lightAOTU.css")

```

```

{
    var newTheme = "darkAOTU.css"
}
else
{
    var newTheme = "lightAOTU.css"
};

// new/opposite CSS file set as style sheet for html page
document.getElementById("CSS").setAttribute("href", newTheme);
}

```

'actionPage.php'

```

<?php

// gets all the form answers
$name = htmlspecialchars($_POST['name']);
$age = (int)$_POST['age'];
$bestFeature = htmlspecialchars($_POST['bestFeature']);
$worstFeature = htmlspecialchars($_POST['worstFeature']);
$otherFeedback = htmlspecialchars($_POST['otherFeedback']);

// concatenates the variables and produces a sentence
$feedback = "Name: $name \r\n Email: $email \r\n Age range: $age \r\n
Best feature: $bestFeature \r\n Worst feature: $worstFeature \r\n Other
feedback $otherFeedback .";

// headers for sending HTML emails
$headers[] = 'MIME-Version: 1.0';
$headers[] = 'Content-type: text/html; charset=iso-8859-1';

// sends an email with the information to me
mail($_POST["15brahmbhatt@students.watfordboys.org"], "Feedback from"
$name, $feedback, implode("\r\n", $headers));

?>

```

BLANK PAGE

BLANK PAGE

BLANK PAGE