OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name : <ABDULLAH OMAR>

Candidate Number : <9361>

St Margaret’s Academy : <34363>

Title of Project : <AIM TRAINER>

H446-03 – Project CONTENTS

TABLE OF CONTENTS

A. ANALYSIS...................................................................................................................................................................2

Identification of problem...............................................................................................................................2

Justification on why it can be solved through a computational approach..................................................17

Identification of stakeholders.......................................................................................................................3

Presentation of solution for the stakeholders to use..................................................................................16

Problem research.......................................................................................................................................6

Essential feature identification and justification.......................................................................................17

Identification of limitation and justification..............................................................................................18

Specification of systematic requirements for the solution......................................................................19

Identification and justification of success criteria.....................................................................................21

B. DESIGN....................................................................................................................................................................23

SYSTEMS DIAGRAM......................................................................................................................................23

Description of aspect of the solution.............................................................23, 27, 29, 30, 32, 36, 44, 46

Usability features of the solution.............................................................................................................37

Identification of structures...................................................................................................................28,44

Validation.......................................................................................................................................25,31,41

Test data to be used......................................................................................................26,28,31,32,37,41,45

Post development test identification........................................................................................................50

C. DEVELOPING THE CODED SOLUTION......................................................................................................................54

Milestones.................................................................................................................56,60,65,67, 77,80

Code.....................................................................................................................58-62, 64,65, 68-77, 79-85

Validation...........................................................................................................................................62,58,65

Iterative reviews.....................................................................................................................59,66,74,77,85

Prototype evidence..................................................................................................60-64,66,68,74-81,83-85

D. TESTING TO INFORM EVALUATION.......................................................................................................................86

Testing and review of each stage...................................................................................60,66,80,84,85,86

E. EVALUATION...........................................................................................................................................................88

Evidence and comment of post-development test.....................................................................................88

Usability testing......................................................................................................................................93,

Cross reference with success criteria..........................................................................................................94

Evidence and evaluation of usability features...........................................................................................98

Evaluation of partially met features.........................................................................................................101

Evaluation of unmet criteria......................................................................................................................102

Maintenance issues...................................................................................................................................103

Limitations..................................................................................................................................................104

F. PROJECT APPENDIXES..........................................................................................................................................114

# A. Analysis

PROBLEM IDENTIFICATION:

In the majority of first-person-shooter (FPS) games, a player’s performance and skill level are determined by many aspects such as their style of gameplay. Examples are if a player is aggressive and likes to rush in or if they play passively and is opportunistic etc. However, another aspect that represent a player’s performance is statistics from their gameplay and there is a kaleidoscope of them, just to name a few: accuracy, headshot rate, kill-to-death ratio, damage taken per minute, assists, win rate etc. Unlike gameplay style (which is a qualitive aspect of defining a player’s performance), the statistical side of a player’s performance is quantitative, therefore it can be manipulated.

In addition to this, due to the nature of the qualitive aspect of a player’s performance, it is what is usually presented on a player’s profile for others to view. The game can also manipulate and analyse these qualitive aspects of a player to possibly rank them in a ranking system to assign them to a specific competitive league or to match them with players that have a similar calibre of skill as them (this is also known as Skill Based Match Making (SBMM)). Therefore, many players focus on this aspect greatly due to this nature.

Due to this nature, each game a player plays on said FPS game will contribute to their qualitive skill value because their current stats will get adjusted, making it volatile. An issue from this is that of course, these stats can be impeded and hindered for multiple reasons such as: the player being on a losing streak, the player is playing after a while of not playing, the player might be using a new strategy or a new set of key binds(key controls), they may be getting used to a different input device such as moving from controller to mouse and keyboard etc. Such an issue can lead a player to annoyance, disappointment, lack of motivation and possibly making them invest even more time into playing just to get back to a satisfactory skill profile.

I want to design an application that players can use to practice an array of skills that they would need in FPS games so then they can train on these skills to then implement when they player their FPS games. this solution acts as an alternative to training on their actual FPS games but mitigates the issue of a player's performance being reduced while they can still reap the same benefits they desire e.g., better accuracy

STAKEHOLDERS:

My stakeholders/target audience will be those who play FPS

I believe that there should be an alternative for an FPS gamer to hone and sharpen/practice their skills in a way that won’t pose any risk to a player’s rank and their quantitative skill aspects (e.g., kill-to-death ratio). I plan to create an FPS training game/simulator which consists of mainly target practice and that statistics can be gathered to illustrate to a player how their performance is like and how good it is. Due to their being multiple and various FPS game, game mechanics etc will differ from game to game, therefore the player will be able to adjust the properties of their weapon to make it as similar as possible to the weapons they use in their original game. Users can test and practice many different aspects in FPS gameplay such as how accurate they are at firing at targets 75m+ away etc.

I have selected 2 stakeholders, Mo and Jay, who are avid First-Person-Shooter gamers and are fond with the idea of finding ways to practice their FPS skills

Interview:

Here, I’ll interview my 2 stakeholders where this project of mine may be of their interest. From this, I aim to hear out as many recommendations and suggestions on what they’d like from this project I will be presenting. Although I may not be able to apply and solicit all of what the interviewees have to say, I will primarily focus on suggestions they have in common and what they prioritise.

My interview questions for both Mo and Jay (dedicated gamers who play competitively)

1. What are 3 skill sets you prioritise to practice/train on?
2. Which FPS game(s) do you play now?
3. What is current method of practising/honing your skills.
4. How satisfied are you with your method of practice on FPS games and why?
5. How would you like your practice performance to be presented to you?
6. Are there any other features you would like in an FPS practice game?

Question 1: There are a variety of aspects in FPS gameplay such how well one can fire haphazardly moving targets etc. Due to this, it’d be hard to focus on all of these so I will focus on trying to configure a solution on how these gamers can ameliorate on their ‘top 3 skill sets’.

Question 2: This is so I can get a grasp on what type of game mechanics these gamers are used to such as whether it’s a fast-paced FPS game such as CS: GO. With this knowledge, I may be able to slightly mould the game mechanics to be like the current game the interviewees are already playing. Moreover, as an alternative I may also create the option for players structure the mechanics of their game to a style like their original game e.g., tweaking the fire rate of the training gun so it is like the one that they use in their original game.

Question 3: By asking this, I can obtain a good idea on what the gamers are familiar/comfortable with in terms of FPS practice and their current style of practising.

Question 4: I want a general insight on the views of the interviewees on their method of practice. I wanted to do this as I may be able to benefit and possibly implement their reasoning within my work to satisfy and improve the functionality of my project.

Question 5: Ultimately, I desire to use a concrete and genuine method to present to a player their performance for the user to able to view and analyse comparisons between performances. Also, this is much better than just relying on the user’s perception of their self and their opinions and how they feel about improvements.

Question 6: This is for any other additional comments, queries, and suggestions that the interviewees may have to offer that I may be able to effectuate.

conducting the interview:

Mo:

1. What are 3 skill sets you prioritise to practice/train on?

For me, accuracy, kills per minute and target switching are the priority for me.

2. Which FPS game(s) do you play now?

Just escape from tarkov.

3. What is current method of practising/honing your skills.

I just keep playing the game to get better and improve in it

4. How satisfied are you with your method of practice on FPS games and why?

Rather unsatisfied due to my method of practice sometimes having a negative impact to my kill-to-death ratio.

5. How would you like your practice performance to be presented to you?

I would like my performance of my latest training session and my overall ‘summary’ performance to be presented with tables or bar charts because of their clarity and my fondness with them

6. Are there any other features you would like in an FPS practice game?

I would like there to be a variety of weapon types to play with as I am rather versatile with my weapon usage.

Jay:

1. What are 3 skill sets you prioritise to practice/train on?

Accuracy, recoil control and TTK (Time To Kill)

2. Which FPS game(s) do you play now?

I currently play APEX but mainly CS:GO

3. What is current method of practising/honing your skills.

I play APEX for leisure and to improve my shooting skills for CS:GO

4. How satisfied are you with your method of practice on FPS games and why?

Although I do find it find it enjoyable, it isn’t efficient due to the game mechanics of APEX being quite different, so I am unable to fully implement all the benefits of my skill training from APEX into CS:GO.

5. How would you like your practice performance to be presented to you?

I would like there to be a colour coding system presented to me so and stated boundaries. An example of what I mean is after a training run, I can be presented a green colour if I completed the run in 20-25 seconds, yellow if completed within 25-30 seconds etc.

6. Are there any other features you would like in an FPS practice game?

It’d be nice if there were a variety of targets such as slow-moving ones, fast-moving ones, airborne ones etc.

Comparisons with existing models:

3D aim trainer:

This application provides a variety of different methods of training for a set of skills that are categorised within the application and below is a few of the categorisations in detail with a corresponding exercise:

1. FLICKING:

Flicking concerns the speed and the time taken to engage with a target after being in a neutral state. This skill is essential in FPS games as just a few milliseconds can determine whether a player can eliminate a target in time before it's too late (i.e the target gets away or eliminates you)

This is one of the many ‘flicking exercises’ called Botflick. In this exercise, the player aims down their scope and stabilise the centre of their view at a small light that is at first red. Once aiming on it, it turns green and then a target will suddenly appear randomly within the player’s field of view. Then they’re required to disengage (from the central light) and eliminate the new target (the orange figure) in as little time as possible. From this, the game will measure the time taken for your engagements and present them to the player.



1. TRACKING

Tracking concerns how well a player can remain engaged on a moving target. Before the player, a moving target will be presented before the player and the player will be holding a device that ejects a laser towards the centre of their vision. The player must coordinate this laser so that its incident on the moving target for as long as possible. From this the game monitors how long you remain engaged on the target

Particularly in this exercise is this glowing cylinder which moves around 3D space, providing dynamic difficulty as not only is the target moving but it moving away minimises it profile.



1. CLICKING

This skill concerns how well the user can accurately fire at targets under a given constraint such as within a given timeframe, or before the targets leave a vicinity etc. From this, the game measures how long a player can maintain a streak and, in some cases, how many times they slip up such as missing a target or a target escaping etc.

In this particular exercise, targets appear from a central point within a ring (at a constant rate) and travel outwards along the radius of the circle (at a constant speed). The aim is to not allow any of these targets to leave the vicinity of the ring.



Aim lab:

This is a well-known aim training application that can be ran on most modern PCs due to its abstractive and simple graphics and players shoot spheres that appear and/or move haphazardly. The benefit of it is to improve the player’s perception, accuracy and reaction time. They offer several modes of practice so a user can tailor their method of practice to a way they like with a variety of difficulties as well.

Serious and competitive gamers use applications such as aim lab as a firm method of daily practice (usually 30min per day) before they pursue with their actual FPS game so they can perform better once they're on after their session of training.

Another interesting aspect of Aim Lab is their technique of displaying to a user their statistics. 6 statistics are displayed in a polygon-like shape, with the stronger a specific stat is the more attracted the polygon’s designated corner is towards it(e.g., this player has great accuracy hence the polygon’s corner being close towards it.)

A screenshot of a video game

Description automatically generated

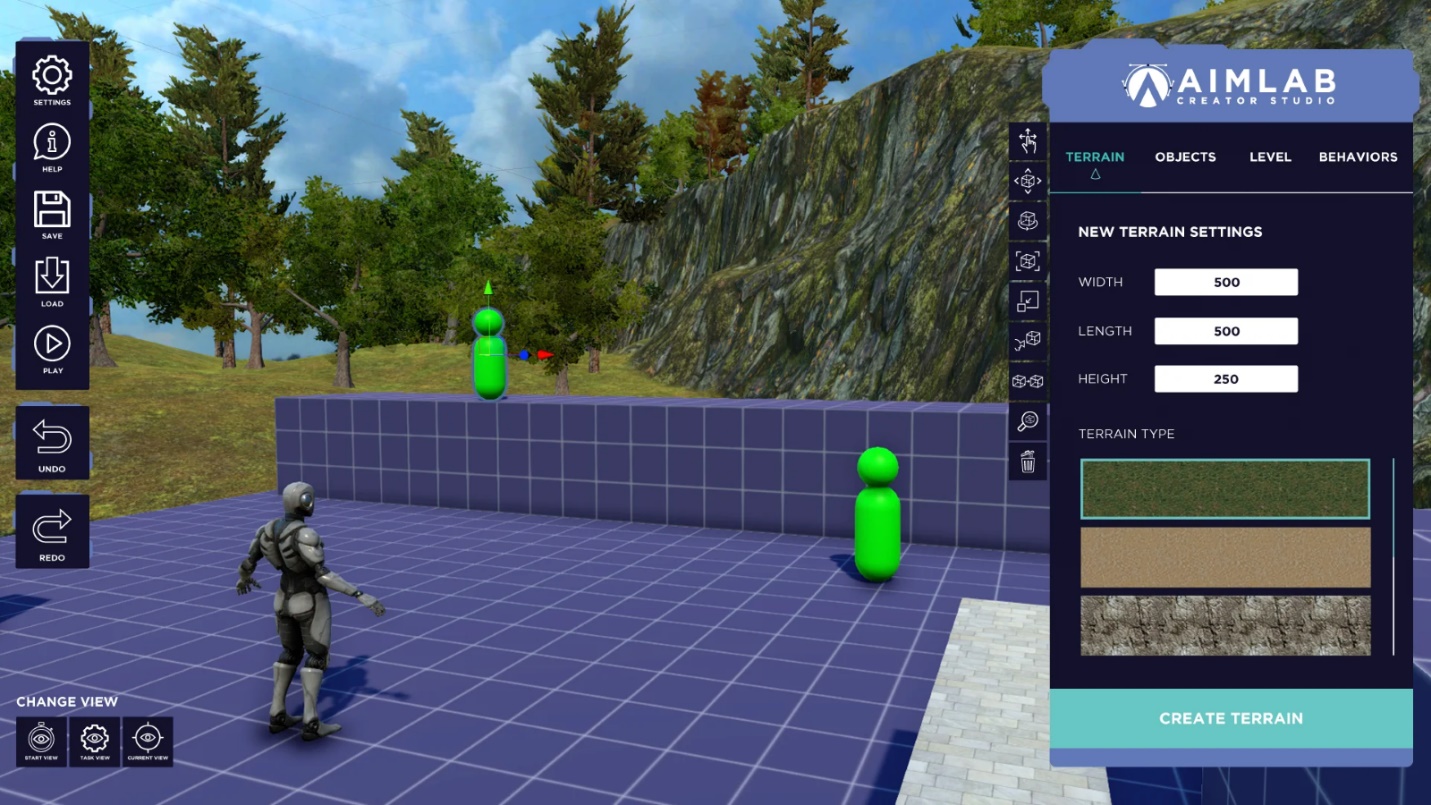
A screenshot of a video game

Description automatically generated

Another unique feature of this game is the level of freedom and customization that a user can have such as a user being able to create their own exercise. In terms of making their own exercise, they can calibrate a large variety of parameters of their scenario’s such as:

* The world of the training room (obstacles etc)
* Size, colour and position of target
* Behavior and nature of target
* Customize AI parameters such as how well the target will try to avoid the player etc

Below is a snapshot of what the creator studio (where a user creates their own scenarios) looks like.



In addition to this aspect of customization, players can also create their own playlists. In this case, playlists are a series of exercises (regardless of whether they’re custom or not) that user’s can order depending on their preferences. A common application of this feature is if a user wants to create a circuit of FPS exercises, specifically tailored to train skills for a particular FPS game a user plays, that the user can conduct before they commence on playing their desired FPS game. This is like a sportsperson performing a warmup that consists of movements that will be heavily involved within the sport they play before they go on to playing the sport itself.

Call Of Duty Mobile Training Mode.

A well-known FPS game out there is the Call Of Duty franchise. In their mobile version, there’s an option to do a training mode. In this mode, the player is placed in a training room with various types of targets. The player uses a weapon of their choice (usually ones they tend to use when playing online matches) and hone their skills.

This is aimed towards competitive gamers who want to practice beforehand before they commence with their actual matched and for casual gamers who may want to practice or get a grasp of a new weapon they began to use. However, this is a part of a game meaning it’s primarily focused for practicing for that specific game (CODm) unlike Aim Lab which is designed to practice on FPS games in general.

An interesting aspect of this training mode is that there’s a designated target which has a smaller version of it by the player to view. The function of this smaller version acts as a display to show what part of the actual target the player hit, an ideal feature especially when the target is far away.

A screenshot of a video game

Description automatically generatedA screenshot of a video game

Description automatically generated

Features of my proposed solution:

Initial concept of my solution after evaluating this research:

I desire my solution to first present the player with a set of basic instruction that will present the common features on what they can do and then possibly a tab that they can click which then allows them to manipulate their target practice to their liking, this can all be skipped by a click if the player desires to do so to avoid repetition. The user is then offered a variety of weapons with different natures (such as whether they want to choose a pistol or a sniper etc). A table of statistics can present the player’s performance and create comparisons between previous games. Furthermore, the player can choose what type of map they want, what type of training session they want and their desired difficulty (which will consider of various aspects which they can each calibrate themself)

Parts I can apply to my solution from AimLab:

1. The presentation of a few live statistics on the HUD so the player can always keep an eye on their performance and not possibly go off on a tangent.

2. An abstractive appearance of the player’s weapon as its appearance nor the need for its similarity to other game weapons are essential.

3. An update on a player’s performance and a comparison between their current performance and their average performance or their penultimate performance

4. A visual way to illustrate to a player their current or average performance. Although I may not be able to replicate a method like this polygon-like one, I may do alternative ways such as bar charts etc.

Parts I can apply to my solution from CODM:

1. A tab that will allow users to modify the nature of their targets to their liking (such as varying their speed and the amount there is).

2. Create multiple maps for players to choose from because this will allow them to train in different environments such as an outdoor map where targets are out in the open and an indoor map where targets are more concealed.

3. A method of displaying to users where they’ve hit their targets so they can have a visual representation of what their performance is like. This can be in the form of a chart beside the player which can show what part of the target ahead of them (that they’ve been firing at). Has been hit.

Parts I can apply to my solution from 3D aim trainer:

1. Various skills that the player can choose to improve on such as exercises for accuracy, exercises for target swapping etc.

Further interview:

This is the email I’ve sent to my stakeholder’s:

Hello,

I have gathered a few general ideas that I think would be appropriate to implement that I will list so please do share your views, what you think on it and if so, what can be improved from it.

I desire to create an application which is designed for users to sharpen and enhance their skills on FPS games as an alternative to playing on their actual game to avoid the possible crippling of their skill stats. The game will gather statistics to present to the player information about their performance. A form of tutorial will automatically run whenever the game begins running but can be skipped at will, the player can then be given a selection of maps and types of training sessions

Summary of the replies of the interviewees:

* No need for a tutorial, a page in the menu with key binds is good enough as skipping through the tutorial will eventually become unnecessary time wasted
* Don’t keep all the stats on the HUD as it will be obstructive, a separate page on the menu containing the statistics would be preferred
* Make sure the game possesses the common mechanics of FPS games such as how the movement is etc. This is so training on it can be as applicable as possible on actual FPS games.

After reviewing each of these existing solutions, taking inspiration of some of the features included in each one and taking on board the feedback from my latest interview with my interviewees, I have established a set of essential features I would like to include in my proposed solution. Below is a list of the essential features that I will include in my proposed solution along with justification on why I will include them:

1. A main menu

I believe it is imperative to have a main menu for any game let alone a game of this specific nature. From this main menu, the user should be able to access other parts of the game such as viewing the controls and adjusting them etc, viewing their performance, in-game practice or to exit the application

1. Gameplay adjustability features:

It's obvious that not all FPS games are the same and will differ from each other in difference ways. Furthermore, players will have their own preferences on certain features throughout their gameplay such as mouse sensitivity etc. Therefore, I believe that it’s imperative that I provide users the ability to calibrate these certain features within this application to suit their preferences.

1. Basic FPS features:

I intend to make this application aimed towards the nature of FPS game therefore its compulsory that I implement the common user interaction features such as basic horizontal motion, advanced movement such as jumping and sprinting etc and a player camera that can look around the world the player is in. I would also like to implement at least the essential shooting mechanics for weapons as this application is going to imitate First-Person-Shooters.

1. An array of weapons and targets

As I have previously mentioned, different players will have different preferences and reasons for why they want to use this application such as if they would like to practice using pistols and/or practice on firing airborne targets. Therefore, I believe that it’s imperative that users are provided with these varieties to satisfy their needs and preferences.

1. An array of statistics about the player’s performance

The main objective of this application is for users to hone their FPS skills. Therefore, it's ideal that there is a firm and quantitative way of portraying to a user their performance (in a numerical manner through statistics) based on data gathered from their performance such as shots on target, headshot count etc.

Limitations:

Firstly, due to the kaleidoscopic  aspects in FPS games, it won't be possible for me to implement all features as it will take a lot of time, resources and it may make the size of the application unnecessarily big. Furthermore, if I were to implement an immense range of training features, this will make the probability for bugs and flaws to arise significantly greater which would then require debugging and fixing and hence much longer to complete the project. In addition to this, if I were to implement an immense variety of training features, it will mean that a lot of effort is required to systemize all these features in an orderly and user-friendly fashion on the GUI. As a result, this too would cause a significant setback to my development.

Nevertheless, my project will lack the implementation of AI which tends to be a desirable feature for targets to have on FPS target practice games. It will be very difficult to enforce a degree of complexity of AI within the targets to pose a significant change in difficulty such as them being able to shoot back at the player, avoid bullets etc. Therefore, this will be a feature I won’t be focusing on.

Ultimately, I will primarily focus on creating a functional game environment where the player can traverse around their world and a main menu that the player can interact with to at least start and exit the game. Then, I will focus creating shooting mechanics and targets. Finally, I will focus on creating the mechanics that’ll govern the capturing of data and statistically displaying them.

Why a computational approach is a suitable solution:

ABSTRACTION:

Due to this being an FPS game, I can ignore the visual features of the player such as body etc and possible also the arms and hands handling the weapon as the focus (primarily) is the weapon and shooting it at targets.in addition to other abstracted visual aspects, I will eliminate any aesthetic and advanced realism features such as the appearance of weapons and the nature of firearms such as gun smoke leaving the barrel. Furthermore, I will focus on implement common and essential features for simplicity reasons on my half and if I did proceed to carve my project to have mechanics similar or correspondingly to a specific game, then this project’s array of possible stakeholders will significantly narrow. In terms of recording shots, rather than giving a specific result depending on what specific part of the target(body), I can implement abstraction to simplify into a few regions e.g., headshot region, body shot region etc.

PATTERN RECOGNITION:

Monitoring the players action and performance is essential therefore pattern recognition will be essential. The player’s input will get analysed and then inferred to then output a suitable result such as if the player pressed ‘W’ to move forwards or a combination of keys to produce a desired output. Furthermore, the player’s accuracy and result of firing must be measured in some way and registered to then be analysed and illustrated to the player if they want to look over how they performed

Decomposition:

This will be vital part when pursuing with my project as it will make me be able to manage and comprehend better what is required for this solution. An example of implementing this is with the targets:

1. The appearance and shape of the targets
2. The nature of the targets (static, airborne etc)
3. Their AOEs (Area Of Effect)
4. Their health and defence

Algorithms:

This will be responsible for determining the suitable step-by-step solution to a given problem. These determination processes can all be done by concurrently running checking processes such as checking if the player has run out of bullets in their gun: informing the player about the status of their gun (it being empty), whether they have enough bullets to reload and if so, how many can be loaded in.

SYSTEMATIC REQUIREMENTS:

Below is a table presenting all the I/O hardware requirements for my project

|  |  |  |
| --- | --- | --- |
| No. | Systematic component (hardware) | Requirement to support such component |
| 1 | mouse | Suitable input port such as Bluetooth or USB slot etc. Installed mouse driver, no more than 10Mb needed |
| 2 | keyboard | Suitable input port such as Bluetooth or USB slot etc. Installed keyboard driver, no more than 15Mb needed. |
| 3 | monitor | Suitable input port such as Bluetooth or USB slot etc. Installed monitor driver, no more than 5Mb needed |

Below is a table representing the systematic requirements for the Unity Editor system (which I will be using in production of this project) specifically for Windows, which is the operating system I’ll be using.

|  |  |
| --- | --- |
| Minimum requirements | windows |
| OS version | Windows 7 (SP1+), Windows 10 and Windows 11, 64-bit versions only |
| CPU | X64 architecture with SSE2 instruction set support |
| Graphics API | DX10, DX11, and DX12-capable GPUs |
| Additional requirements | Hardware vendor officially supported drivers |

Throughout the development of this application, I will be using C# the Unity software is adapted upon it and the Unity software is a one that is specifically made for designing games so unlike a regular standalone coding language in its environment, Unity has game tools which allow creating and manipulating gameObjects much simpler than without one such as using python to make a game with the use of the turtle library. Furthermore, Unity is beneficial for independent developers (me in this case) and is free to use as long as the game that I create and publish makes a revenue of over $100,000 annually

Below is a table representing the hardware and software requirements for each of the following operating system for the Unity Player system

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OS | windows | Universal Windows Platform | macOS | Linux |
| OS version | Windows 7 (SP1+) , Windows 10 and Windows 11 | Windows 10, Xbox One, HoloLens | High Sierra 10.13+ | Ubuntu 20.04, Ubuntu 18.04, and CentOS 7 |
| CPU | x86, x64 architecture with SSE2 instruction set support. | x86, x64 architecture with SSE2 instruction set support, ARM, ARM64. | Apple Silicon, x64 architecture with SSE2. | x64 architecture with SSE2 instruction set support |
| Graphics API | DX10, DX11, DX12 capable. | DX10, DX11, DX12 capable GPUs | Metal capable Intel and AMD GPUs | OpenGL 3.2+, Vulkan capable. |
| Additional requirements | Hardware vendor officially supported drivers.  For development: IL2CPP scripting back-end requires Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK. | Hardware vendor officially supported drivers.  For development: Windows 10 (64-bit), Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK. | Apple officially supported drivers.  For development: IL2CPP scripting back-end requires Xcode. Targeting Apple Silicon with IL2CPP scripting back-end requires macOS Catalina 10.15.4 and Xcode 12.2 or newer. | Gnome desktop environment running on top of X11 windowing system  Other configuration and user environment as provided stock with the supported distribution (such as Kernel or Compositor)  Nvidia and AMD GPUs using Nvidia official proprietary graphics driver or AMD Mesa graphics driver. |

Additional utility software that i will be using for development and their systematic requirements:

Visual studio for windows 11:

* 1.8 GHz or faster processor. Quad-core or better recommended
* 2 GB of RAM; 8 GB of RAM recommended (2.5 GB minimum if running on a virtual machine)
* Hard disk space: Minimum of 800MB up to 210 GB of available space, depending on features installed; typical installations require 20-50 GB of free space.
* Hard disk speed: to improve performance, install Windows and Visual Studio on a solid state drive (SSD).
* Video card that supports a minimum display resolution of 720p (1280 by 720); Visual Studio will work best at a resolution of WXGA (1366 by 768) or higher.

|  |  |  |
| --- | --- | --- |
| No. | Criteria | Justification |
| 1 | To make a main menu that has icons such as play, settings, controls, quit | The main menu is essential as it will act as the intersection between the different parts of the application which the user can use to access whichever part of the application they would like to use whether it be viewing their stats or playing the game etc. |
| 2 | Player can adjust certain features such as properties of their UI (FOV, mouse sensitivity etc) and features of targets (e.g. target speed, amount of targets) | This is so player can calibrate these specific features of the game to their preferences. |
| 3 | Player being able to do basic FPS movement and look around with their camera | An essential feature in all first-person games therefore it must be included |
| 4 | Player can choose from a variety of different weapons | The user may want to train themselves using a specific type of weapon such as pistols therefore a variety of weapons should be provided for them to use to shoot with |
| 5 | Player can view their performance in an array of statistical presentations such as rate of elimination, shot accuracy etc (all on the stats tab from the main menu) | The aim of this application is so that users can better their performance in FPS games therefore I believe it’s crucial that a quantitative form of data should be used to display their performance |
| 6 | There are a variety of targets of different natures that the player can choose from to shoot at. | Just as in FPS games, there will be a variety of targets such as stationary ones and flying ones etc. Therefore, I believe that its essential that there is a variety of targets for the user to shoot at |
| 7 | For the player to be able to reload their weapon and aim with their weapon (which should increase the accuracy of their shot and reduce their movement speed) | These are essential features for any guns in video games. Although aiming increases the accuracy of shots, it should come with a trade-off which in this case is the reduced movement speed |
| 8. | Different maps/environment the player can choose from | This is so the player can play in an environment of preference. |

# B. Design

## Systems diagram

Here is the general structure of the application I wish to create



Where necessary, I took an approach of decomposition with certain aspects of the games that are relatively large/complex. Further on, I will elaborate on each of these components of my application

Here is my algorithmic design for the player movement with a flowchart.

A diagram of a diagram

Description automatically generated

I have structured this flowchart in a way where there is no point of termination. This is because the user should always be able to move and nothing should immobilize the player.

Below is a table of variables and the reason for their usage:

|  |  |
| --- | --- |
| Variable(s) | Justification |
| All 3-dimension coordinate axes | This is so they can be manipulated based on their movement |
| input indication flag for WSAD, space key and tab key | This is so knowledge of whether these input keys are being pressed or not can then be used to produce appropriate movements |
| Collision detection state | To indicate if the player is currently touching the ground, this information can then be forwarded on to dictate whether a player can jump or not |
| Step height | A constant value which indicated the max height an elevated object can be, which can still be walked over |
| Obstruction indicator | This variable will indicate if there’s an object obstructing the player’s motion |
| Ceiling distance | This distance will be compared the jump height to govern if they’ll jump to their max jump height or not |
| Walking speed | A fixed value which is used to apply a certain speed when the player is walking |
| Sprinting speed | A fixed value which is used to apply a certain speed when the player is sprinting (a value greater than that of the walking speed) |
| Jump height | A fixed value which is used to assign a specific degree of elevation when the player jumps |

In terms of validation, I will make use of the obstruction detector which will check if the player is currently getting obstructed by an object. This is so if this were to be the case, they’ll be impeded from proceeding towards the direction of the obstruction, otherwise they’ll permeate through the obstruction which would be undesired as it could lead to several problems such as falling off the map. Another form of validation that I will enforce is the determining of whether the player can scale over an object obstructing their motion. I plan to create an obstruction height checker that’ll check if the obstructive object is less than the step height, in which the player can then be capable of traversing over or otherwise simply get their motion towards that said direction to be stopped to prevent them from being able to permeate through the obstruction which would be undesired as it could lead to several problems such as falling off the map etc.

In terms of testing during development, I will make use of obstacles and use movements as test data in these forms:

* For jumping, I will press the space button while the character is in the air as test data, this is to test that the floor detection mechanics work and what should be observed is that pressing the space button while the character is in the air should result in nothing (no further jumping)
* For the horizontal movement, I will make use of obstructions to test the functionality of the collision detector. I will use the inputs of the movement keys (WSAD) as test data and the expected output from these inputs is that they get nullified when the player is obstructed sufficiently (that is, the obstruction isn’t short enough for them to scale over it) in each direction of movement.
* I will repeat the same entire process above but with the addition of sprinting. Therefore, I will use the inputs of the movement keys (WSAD) as well as the tab key as test data. The expected output from these inputs is that they get nullified when the player is obstructed sufficiently (that is, the obstruction isn’t short enough for them to scale over it) in each direction of movement, regardless of the sprinting function being initiated.

Several of these variables will require iterative testing for calibration including: jump height, walking speed, step height and sprinting speed. During development, I will give my interviewees the opportunity to experiment with these variables to find out what values (as these variables will ultimately be numeric values which describe how intense the given function is i.e a large number for jump height will make the player’s jump height large). Once my interviewees find certain sensitivity values for each of these variables that they’re satisfied with, I will then implement them.

Algorithmic design for shooting function:

A screenshot of a computer screen

Description automatically generated

|  |  |
| --- | --- |
| Variable(s) | Justification |
| Ammo pool | A fixed value which represents the total amount of bullets the player has (excluding what is in the weapon itself) |
| Original magazine capacity | A fixed value which represents the max amount of bullets that can be held within the gun at any time. |
| Current magazine capacity | A counter which indicates how many bullets remain the weapon at any given time |
| Alternate movement speed | During aiming and shooting, the player’s movement should be reduced therefore a smaller movement speed should be assigned in the state of aiming. |
| Magnification factor  Accuracy increase factor | During aiming, the player’s accuracy should be enhanced (i.e bullet spread reduced and weapon stability increased). Therefore, these variables are required to be used to enforce these changes when appropriate |

Structure of the shooting class that I will use in the proposed solution:

A diagram of a computer program

Description automatically generated with medium confidence

As unity is an object orientated programming language, will make use of classes when necessary. For now, I believe I will only need to make a single class for weapons. In this class I will define all aspects of the weapons in the game such as:

* How heavy they are (which will affect movement speed)
* Their fire rate
* Projectile velocity
* Size of ammunition pool and magazine
* Speed of reload
* Degree of magnification when aiming

With all these attributes and methods defined within this ‘weapon’ class, I can make weapons which will inherit this class and then I can implement the usage of polymorphism to change these attributes and methods to fit the style of the weapon accordingly( e.g giving a sniper rifle a low fire rate)

Below are the tests that I will commence with during this part of development:

* To test the compulsory reloading function, I will use the shoot button (mouse left click) repetitively as test data. I will do this until the weapon is out of ammunition and the expected outcome is for the weapon to be automatically reloaded (if there is any ammunition remaining)
* To test the futility of the gun once the player uses the entire ammunition, I will use the shoot button (mouse left click) as a test data input and the expected outcome is for nothing to occur and a small message alerting the player that they’re out of ammunition
* To test the effectivity of the accuracy function, I will record a video of firing the weapon while aiming and another one while not as test data. Then I will compare both videos and what should be observed is that the stability of the weapon should be better in the ‘aiming’ video than in the ‘not aiming’ video. Furthermore, it should be observable that the spread of the fired projectiles should be tighter in the ‘aiming’ video than in the ‘not aiming’ video.

During development, I will receive input from my interviewees on what they believe are suitable ammo capacity values for a variety of weapons and take them into consideration when implementing such values into the game.

Algorithmic design for the user's crouch control:

A diagram of a computer program

Description automatically generated

|  |  |
| --- | --- |
| Variable(s) | Justification |
| Crouching flag | Indicates whether the user is in the state of crouching or not (this is so certain conditions can be applied such as the reduction of the user's movement speed etc.) |
| Crouching speed | A fixed value which is used to apply a certain speed for when the player is crouching |
| Crouching height | A fixed value which is used to transform the player’s height to a certain height |
| Input indication for state alteration | Indicated whether the user wants to change their current state of position (from crouching to standing and vice versa) |
| Action inhibition flag | During the state of crouching, the player should be incapable of sprinting and jumping. |

I will provide my interviewees with the opportunity to experiment and calibrate iteratively the values for the crouch height and the speed of the player. Once they’re satisfied with set values for each of these variables, I will take them into consideration when implementing such in this game.

Algorithmic design for the mouse movement:

When the player looks around the world that they're in, they can orientate their point of view along both the y-axis and the x-axis. Logically speaking, the user should be able to pivot on the spot to view 360 degrees horizontally; however, the user should only be able to vertically view from 90 degrees upwards and 90 degrees downwards. Therefore, I've implemented a checker that will ignore any vertical input from the mouse if they attempt to go past these angular benchmarks.

A diagram of a flowchart

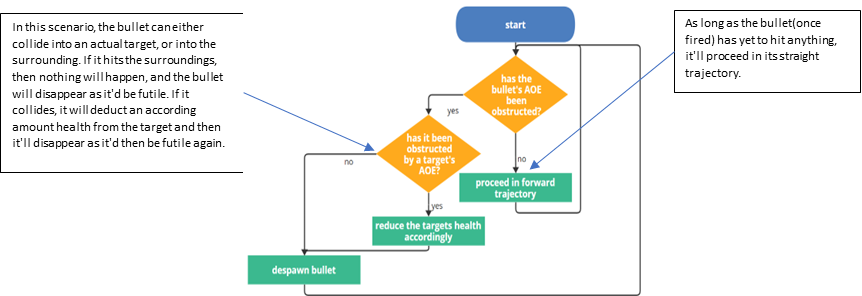
Description automatically generated

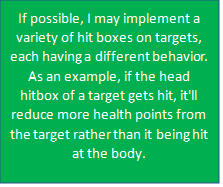
|  |  |
| --- | --- |
| Variable(s) | Justification |
| Vertical and horizontal mouse movement input | These values are required to produce an appropriate angular output (reorientating in player’s camera view) |
| Upper vertical angular limit flag | This indicates if the user is currently looking vertically upwards, in which they’d no longer be able to rotate their camera vertically beyond that point so that they don't end up looking backwards |
| Lower vertical angular limit | This indicates if the user is currently looking vertically downwards, in which they’d no longer be able to rotate their camera vertically beyond that point so that they don't end up looking backwards |
| Mouse sensitivity | This value will alter how fast a user can rotate their camera |

For testing during development, I will be testing the effect of the vertical angular camera view limits (which is the input validation mechanism). To test this, I will use vertical mouse input as test data both up and down when looking vertically up and vertically down (respectively), the expected outcome from this should be that while looking vertically down (90 degrees downwards) any vertical mouse input towards that angle should be nullified. Moreover, while looking vertically upwards (90 degrees) and vertical mouse input towards that angle should be nullified as well.

I will provide my interviewees with the opportunity to experiment and calibrate iteratively the mouse sensitivity until they settle with a value for their sensitivity. Then I will take it onto consideration while creating the camera view function.

Algorithmic design for the bullet mechanics:





|  |  |
| --- | --- |
| Variable(s) | Justification(s) |
| AOE | A defined set of dimensions describing a cuboid encapsulating the projectile is required as it’s contact with other objects will allow the projectile to interact accordingly with these other objects (e.g. with targets etc) |
| Projectile damage | A set amount of damage that projectile will inflict onto targets when it collides with them |
| Trajectory | Once fired, the projectile needs a vector to travel along during its course of motion |

To test this, I will make use of the utilities of the unity software and/or visual basic to observe the presence of a projectile within the running program. As test data, I will fire projectiles towards objects within the environment. What should be observed using the utilities of the unity software and/or visual basic is that these projectiles are present temporarily and then disappear from the program, which is evidence that they do not permeate through objects within the game (which is undesirable).

I will discuss with my interviewees later during development about the specific damage these projectiles will inflict on targets such as in the case of regular shots and in the case of headshots etc.

Algorithmic design for the transfer between the pages of the game:

A diagram of a program

Description automatically generated

|  |  |
| --- | --- |
| Variable(s) | Justification |
| Flags to indicate whether the user wants to open either of the stats, settings, controls tab | This is so then the player can be uploaded the correct tab depending on what they requested |
| Flags to indicate whether the user wants to exit the current tab that they’re on | This is so then the player can be redirected to the main menu page |
| A flag to indicate whether the user clicks on play | This is so the player can get directed to the game scene if they desire to play |
| Application termination flag | This will indicate whether to shut the application or not depending on if the user clicked ‘quit’ |

Once I have the functionality of the menu working along with its other features, I will interview my interviewees to take into consideration their preferences on the layout of the UI of the menu

Game objects:

Avatar: this is the character of the user and will be interacting with the physical environment such as collision with surfaces and scaling objects. The user will be able to move their player along the 3 dimensions, crouch, move while crouching. These are the essential common maneuvers in FPS games so it's essential that I have these implemented. I will also provide the control for these maneuvers with common key binds such as WSAD for movement. I can implement tis by creating a set of methods for the players movement along each dimension and in terms of interacting with the environment such as walls and floors (to not permeate through them) I can apply this by creating a capsule collider which will be able to do such interactions

Weapon: Will be held by the user and will be visible in their FOV. It will be able to perform animations such reloading and shooting etc. It won't have a sophisticated or large appearance for the sake of decomposition and ease for the player. I will try to implement an array of different weapons the user can use to increase the versatility of the options for the players and to suit more users as well. For each weapon class (Assault rifles, sub-machine guns, pistols and snipers are considered the most common) I'll create a recognizable sprite for each one of them and set up their essential properties for each weapon such as: fire rate, reload time, bullet velocity etc.

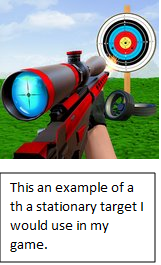
Bullets: These will be small projectiles fired from weapons and exit from the tip of the gun (its barrel). These bullets will have a hitbox around them in which if this hitbox does meet another body a certain result occurs. As an example, if a bullet hits a target, then it will indicate the player, deduct a specific amount of health points from the target, then disappear. I will implement this nature with bullets as it’s like how bullets function in FPS games.

Environment: impermeable solid objects that the user plays in (their world). The maps in this game will be in a layout that are commonly found in FPS games such as a tower to shoot from by a large plane, peek points inside rooms etc.

Targets: Targets will come in various forms from static targets to airborne targets. They will also move around and appear in front of the user is many ways and I can implement a variety of these manners such as slow constant movement to others randomly appearing on screen. Firstly, this is because there are various types of targets amongst FPS games such as ones that remain still (static targets) and enemies that may be flying (airborne targets) such as enemies coming in with parachutes. Secondly, by providing the user with a variety of targets to practice with, this will improve their shooting skills such as their speed of perception, how well they can lead a shot etc.

A screenshot of a video game

Description automatically generated



This is an example of a stationary target I would use in my game

ALGORITHMIC DESIGN FOR THE TARGETS:

A diagram of a health care program

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| Variable(s) | Justification |
| Target speed | This variable controls the speed of the target |
| 3d position coordinates | For the target to move, it must have a destination(s) to go to and from |
| Target health | A fixed value which represents the amount of health points a target has |
| General AOE hitbox | A set of given dimensions that encapsulates a certain region of the target and will lead to deduct a specific amount of health from its health when in contact with a projectile |
| Critical region AOE hitbox | A set of given dimensions that encapsulates a certain smaller region of the target and will lead to deduct a larger specific amount of health from its health when in contact with a projectile |

To test this during development, I will use shots on targets as test data. When the targets are fired at, an according amount of health should be deducted from them, furthermore their course of motion/positioning should also change when fired at. In addition, I will use shots on target as test data to attempt to fully eliminate a target, the expected result from this is that the target disappears or repositions to its original starting position. Nevertheless, appropriate alteration to data should occur due to test data input (firing/eliminating target) i.e the accumulation of the total amount of target eliminations etc

I will provide my interviewees with the opportunity to experiment and calibrate iteratively these variables until they provide me with a set of values (for each variable respectively), in which I will take into consideration during development.

Settings

Sensitivity:

In terms of importance of calibration in FPS games, mouse sensitivity is considered an essential to be adjusted and therefore I will focus on implementing an option for users to adjust it their liking. I desire to present this option of calibration with adjustment bars. One for the sensitivity on the x axis(horizontal) and another for the y axis(vertical) or I may do a single adjust bar to calibrate the mouse sensitivity on both axes.

Here is a screenshot from call of duty mobile of a very small section of their setting calibration options. I would like to use some of these features in this game’s setting in my application for multiple reasons, I like the simple abstract layout which will not possibly perplex the user, each calibration feature is labelled neatly and the current level that the setting is presently on. The visual representation of a bar with a minus and plus sign on either side (for adjustment) is also a feature I’d like to implement in my project.

A screenshot of a computer

Description automatically generated

Controls:

Here is a basic layout that would provide useful information to novice users wanting to know which button does which

A screenshot of a computer screen

Description automatically generated

Visuals:

In all FPS games, there are crosshairs which is a faint central set of lines in the middle the players HUD and is customisable depending on the user’s liking examples are: classical crosshair, ring-like-pattern, square etc. Another customisable feature I could add is crosshair colour as many people have different preferences to what colour they want their crosshair to be as its colour is important to distinguish with the environment

A collection of white circles and squares

Description automatically generated

FOV(field of view):

This too is an essential feature. In FPS games, the field of view is the vertical and horizontal angular view limit for a player’s point of view (POV). Many FPS games provide their players with the option to adjust their FOV (some games providing the adjustment in both axes and others just a combined adjustment)

A screenshot of a video game

Description automatically generated  Below is a list of variables (concerning the functionality of settings) that will be used and their justification.

|  |  |
| --- | --- |
| Variable(s) | Justification |
| Chosen mouse sensitivity | Although I already included this variable previously in the camera mechanics, here the user will be able to adjust it to how they like. This value will control how sensitively the camera will move based on the user’s mouse movement |
| Aiming sensitivity | When aiming, players attempt to stabilize their vision on their target therefore there must be a separate (and usually lower) sensitivity value so that players can perform slight adjustments to the camera orientation easier |
| Selected reticle | This variable will hold the reference to which reticle the player chose to use, which will be then uploaded to the center of their screen once in game |
| FOV angle | This value will hold the angular range of view of the player’s camera, which the player can calibrate to their liking, however this must be bounded within a certain range of angles. This is because FOVs that are too small or too large are unsuitable and will never appear on real FPS games (e.g. a field of view angle too large can make the player almost see what's directly behind them) |

To test this during development, I will use a variety of set adjustments as test data to see their effect in-game. The expected result from these specific adjustments should be the according to the adjustment made, as an example I could compare the effect of a certain mouse sensitivity value with one that is higher than it and it should be evident that the camera movement sensitivity is noticeably higher with the larger value

Nevertheless, as a form of input validation, I will implement a function that allows the user to only select a single reticle at once. This is to prevent the user from selecting multiple reticles simultaneously which may provide them with an unrealistic experience or even dissatisfaction as several reticles selected could result in them overlapping each other on the centre of their HUD and becoming a visual obstruction. To test this, I will use selected reticles as test data and what should be observed that when a new reticle gets selected, the previously selected reticle should be removed, which should be evident both in menu on the reticle selection part of the settings and in-game as well.

I will provide my interviewees with the opportunity to experiment and calibrate iteratively the dimensions of the selected reticle and FOV angle. Once they’re settled with a set/range of values they’re satisfied with, I will then take it into consideration during development.

Gameplay interface:

A screenshot of a video game

Description automatically generated

Stats:

The main role of this application is for users to enhance their quantitative aspects of their FPS and therefore I believe that it is crucial to provide some type of system that can infer from a user’s performance and manipulate it to present a user aspects of their performance such as their accuracy as a percentage etc. Therefore, I must prioritise a vivid display of a user’s performance. Primarily, would like to just focus on a percentage system among each quantitative stat and possibly provide a comparison between the user's latest performance and their overall performance. For now, I would like to focus on displaying the user's stats by just creating separately spaced labels and with an appropriate measurement(such as percentage) underneath it such as this prototype below.



Approaches:

There are 2 ways I can go about presenting the user with their performance comparisons. I can either provide a comment describing the comparison of their performance paired with an actual quantitative comparison or I can take a more formal approach and present a user's average stats with their last stats.

So far this is my idea for the first option. When the user chooses to see their stats an array of tabs like these 3 will appear in front of them, each commenting on a certain stat of the user's performance that differed from their average stats. I can prepare a set of comments, each for a certain type of observation made i.e., a text to comment on a certain stat that the user has improved on, a text to comment on a certain stat the user has impaired on etc.

A group of hexagons with text

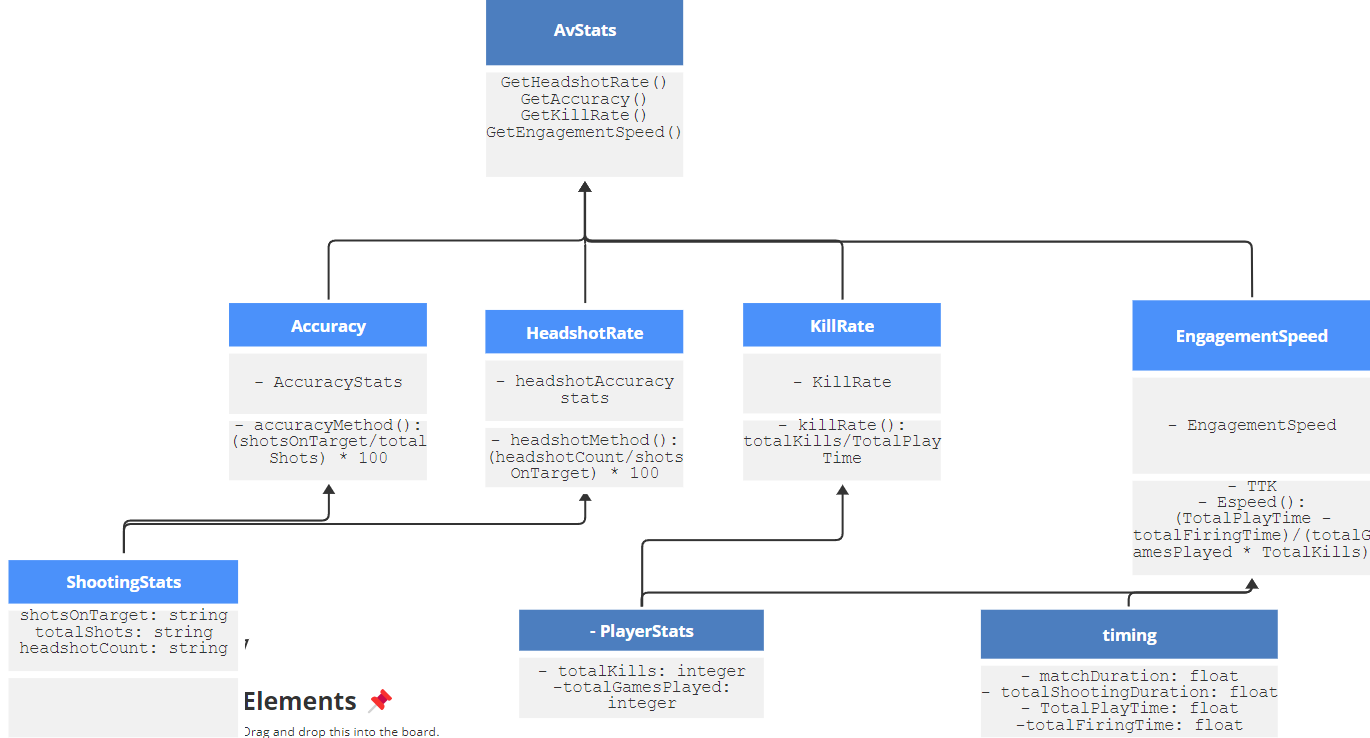
Description automatically generated

This 2nd idea is a more formal and illustrative approach of presenting a user their performance. Unlike the one before, this presents a comparison with all the user's performances. Although the user cannot directly be given a comparison between their latest and average performance boldly, they can view trends in their performance e.g. if they decide to test out  a new technique in the hopes of it being effective, they'd be able to so on a graph(e.g. a positive correlation).

A screenshot of a graph

Description automatically generated

Problem modelling of the statistics:

  
  
Here is a class diagram for the statistics assemblance to be displayed to the user. These statistics are just the user's OVERALL performance; hence they’re always being total values as attributes throughout this diagram (accumulating values since the user started using the application). I can create a modified replica of this diagram but rather than it being for the user's OVERALL performance, it can be for separate recent performances. From this. I can then perform comparisons between recent performances or recent performances and the user's overall performance

  Below is a table of variables and the reason for their usage:

|  |  |
| --- | --- |
| Variable(s) | Justification |
| ShotsOnTarget  TotalShots  HeadshotCount  TotalKills  TotalGamesPlayed  MatchDuration  TotalShootingDuration  TotalPlayTime  TotalFiringTime | These are all numerical variables which are gathered to be used for statistical calculations |
| AccuracyStats  headshotAccuracyStats  KillRate  EngagementSpeed | These variables hold results of statistical calculations based on the user’s performance that are to be uploaded to the user |

To test this, I will use a variety of test data that effect each of these values respectively including the following:

* I can use shots fired (due to left-click mouse input) to test the functionality of TotalShots. Whenever this input is initiated, this value should increment numerically and accordingly. Likewise, I will use a similar approach to test the ShotsOnTarget and HeadshotCount variables and they too should increment numerically and accordingly (whenever a shot lands on target for the ShotsOnTarget variable and whenever a shot lands on a target’s head for the HeadshotCount variable)
* For the time related variables such as TotalShootingDuration, TotalPlayTime and TotalFiringTime, I can record myself using the application and then obtain the corrospondind time values stored within each of these variables respectively as test data. I can then compare each one with the given duration of the recording in which I was testing that specific time variable and if both duration values match, then I can conclude that the behave as intended. As an example, If I had a minute long recording of the testing of these variables and between the time stamps 0:16 and 0:21 (5 second period) I was firing the weapon, then the variable ‘TotalFiringTime’ should have the value of ‘5’ saved within it etc.

During the development, I will ask my interviewees their preferences on where and how these stats can get displayed, in which they can do so by testing and experimenting it themselves before they report back to me.

Here I will use IPSOs charts to display the transformation of transformation of various types of data within my application.

|  |  |  |  |
| --- | --- | --- | --- |
| **input** | **process** | **storage** | **output** |
| Movement buttons(WSAD + shift) | Infers from the user’s input(combination) | Updates the coordinate positioning of the user's avatar(in terms of the X and Z axes). | A course of movement will be outputted based on the user’s input(s)(possibly a combination) |
| Left click(selection) | Analyses if the user’s mouse is hovering over a selectable tab (e.g., the quit button) | N/A | Uploads the selected |
| Right click(aim) | Allows the user to aim down the sight of their weapon | N/A | Provides the user with a view of them aimed down sight (a zoomed in image and/or a tightened FOV) |
| Left click(shoot) | Allows the user to fire their weapon | Decreases the stored number of bullets within the user's weapon | When under correct conditions, fires the user's weapon |
| Crouch/standup(C) | Reviews the current stance of the user and then provides the necessary change to their stance (i.e., from standing to crouching or crouching to standing) | Updates the current stance of the user (i.e., if they're currently stood or crouching) | Provides the necessary change in stance of that the user requested and alters their appearance and/or perspective |
| Reload(R) | Analyses the current state of the user's weapon and then dictate the necessary process needed for such an operation such as if the user has an empty weapon or a half-filled weapon, what is the necessary procedure needed (how many bullets needed to refill) | Alters the number of bullets remaining within the user's store with respect to the specific number of bullets needed for their reload. | The apposite alteration is made to the current weapon's bullet count with regards to how much bullets they have remaining in their store. |
| Jump(space) | Allows the user to temporarily elevate (in the form of jumping) depending on the available room of motion (i.e., whether the user has a ceiling near their head or not) | Updates the coordinate positioning of the user's avatar (in terms of the Z axis) | A vertical movement in the form of jumping will be produced |
| camera view alteration (Mouse movement) | Changes the user's angle of perspective of the world that they're in with respect to the mouse movement they produced. | Updates the angle of the user's camera perspective | Alters the user's view of the world accordingly with the mouse movement they produced |

|  |  |  |  |
| --- | --- | --- | --- |
| **Numerical input of a variety of the user's actions such as number of shots they’ve fired, number of targets eliminated etc.** | **These obtained values are categorized into their fitting classes such as whether the obtained value is accumulated onto a total running value (for the calculation of their overall performance) and/or onto a temporary total(for the calculation of their performance of a certain match). Appropriate calculations are then made for the user's statistical performance** | **New values then replace old ones to present the user with their latest statistical performance** | **If the user wishes to view their latest stats, they can do so** |

Files

It is possible that I may use external sources for my game sprites rather than using the Unity software itself; it'll be convenient if I have files that are encapsulated for each sprite in this game such as weapons, and targets. As an example, I may store in each file the actual sprite body, its dimensions, its color(s) etc (accordingly to each game object I desire to implement this upon throughout my application).

|  |  |
| --- | --- |
| **Airbourne target** |  |
| flyingTargetSprite.OBG | This as an example is a file for a 3D sprite for my airborne targets |
| 50 | Desired height of my sprite |
| 50 | Desired width of my sprite |
| 60 | Desired length of my sprite |

Menu UI:

This is just a very basic layout for now as I would like to focus on the key functionality of the application before I point my concern towards the aesthetic aspect of the UI with my interviewees as I have mentioned beforehand.

A screenshot of a computer

Description automatically generated

Player experience:

In terms of difficulty of the gameplay, I will either implement it in an automatic manner where overtime the difficulty will increase (the speed of the targets and the rate at which they appear). On the other hand, I can apply this through a customization setting where the user can manually calibrate the aspects of the gameplay's difficulty to their liking. Nevertheless, with either solution I would have to assign variables to control the rate at which the enemies appear and their speed, hence me mentioning so in the variable justification table. Due to this being a training game, the user would also like to view their performance in a statistical nature. Therefore, I believe I should create a page that the user can go onto when they open their application to view their stats. If I am unable to do this then I will at the very least insert a live stats board in the world the user plays in so that they can view their stats while playing.

If the player wishes to, they can go onto the settings and calibrate certain parameters to their liking such as mouse sensitivity etc.

Test development:

Throughout my development process, it's imperative that I thoroughly examine the functionality of my application in a similar nature to black box testing. For each aspect of functionality of the application (such as how targets behave when they're shot at), I will list it with its corresponding expected outcome against the actual outcome I receive when I interact with that specific function in its designed appropriate manner.

I will represent this below in tables containing the main milestones of the developments process and their several components within

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Aspect to be tested and according inputs for function** | **Expected outcome** |
| 1 | The user left-clicking the play icon | The play page is uploaded, with a given option to return to main menu |
| 2 | The user left-clicking the stats icon | The stats page is uploaded, with a given option to return to main menu |
| 3 | The user left-clicking the settings icon | The settings page is uploaded, with a given option to return to main menu |
| 4 | The user left-clicking the controls icon | The controls page is uploaded, with a given option to return to main menu |
| 5 | The user left-clicking the return icon upon entering any of the 4 pages above | The original main menu page gets presented to the user |
| 6. | The user being able to exit the game to return to main menu | The game returns the user to main menu |
| 7 | The user left-clicking the exit icon | The application shuts down |

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Aspect to be tested and according inputs for function** | **Expected outcome** |
| 1 | A and D key should move the player left and right(respectively) | An appropriate change of positioning (along the x axis) according to the user's input |
| 2 | W and S key should move the player front and back(accordingly) | An appropriate change of positioning (along the z axis) according to the user's input |
| 3 | Space key should make the player jump and temporarily elevate them | An appropriate change of positioning (along the y axis) according to the user's input |
| 4 | Body collider detector when in contact with a surface in any of the 3 axes | Stops the player from proceeding to move any further along an axis (if obstructed upon so) |
| 5 | Scaling detector detects whether an obstruction (along either the x or z plane) can be scaled over | Allows the player to scale over any obstructing objects under a given height. |
| 6 | Tab key getting pressed simultaneously with at least of the WSAD keys. | Allows the player to proceed in the same direction but with a greater speed(sprinting) |
| 7 | C key getting clicked and changing the stance of the player from upright to crouching (or vice versa) | Changes the stance of the player, their profile will be reduced and so will their speed, they shouldn’t be able to jump |

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Aspect to be tested and according inputs for function** | **Expected outcome** |
| 1 | Movement of the user's mouse should orientate the player's camera view accordingly with their input and the sensitivity | An appropriate change in the player's camera orientation with regards to their input |
| 2 | There should be no limit the horizontal change in orientation of the user's camera view (they should be able to view 360 degrees horizontally) | No matter how far the mouse is moved horizontally, the user's camera view should also change accordingly. |
| 3 | There should be a vertical orientation limit of 90 degrees upwards and 90 degrees downwards | Regardless of the user's vertical mouse input, it should be ignored if its exceeding one of these vertical orientation limits |

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Aspect to be tested and according inputs for functions** | **Expected outcome** |
| 1 | When the user has bullets in their weapon and presses to shoot, a bullet should be shot. | A small projectile exits the gun and travels in a straight course relative to where it was shot from. |
| 2 | When a bullet hits a target, damage should be inflicted | An according amount of health should be deducted from that target. |
| 3 | When a bullet hits any object in its environment (including targets). | The bullet should completely disappear (not just become invisible as then it'd still behave as a bullet). |

|  |  |  |
| --- | --- | --- |
| **Test no.** | **Aspect to be tested and according input for function** | **Expected outcome** |
| 1 | Right click should enable a shooting animation and releases a bullet. During this state, the movement should be lowered. | Shooting animation should occur (such as muzzle flash) and bullets should be fired, simultaneously reducing the bullet count in the weapon's magazine. |
| 2 | The bullet should follow a straight trajectory towards where the player momentarily looked at when they fired | Projectile should follow a fixed course of motion towards where the player aimed at when fired |
| 3 | There should be a variety of weapons, with each one having a unique feature such as different fire rate etc. | Unique characteristics of each weapon type should be distinguishable |
| 4 | Left click should enable an aiming animation. The accuracy should significantly increase but the movement speed will be significantly lowered | (While getting held) the player's view should change so that the weapon looks uniform in the player's line of sight and shots fired during this state should be of better accuracy than not aiming |
| 5 | Pressing R should enable a reload animation and an appropriate number of bullets should be subtracted from the ammunition pool into the weapon | If the magazine is full, nothing should happen.  If the number of bullets required to suffice a refilled magazine is possible |
| 6 | The weapon should become futile once the player is out of bullets | The game should be unable to shoot with the weapon |

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Aspects to be tested and according inputs for function** | **Expected outcome** |
| 1 | Variation of the FOV of the player's camera view | FOV gets adjusted accordingly to how the player adjusted it to. |
| 2 | Variation of the player's mouse sensitivity  and aiming sensitivity | Mouse sensitivity gets adjusted accordingly to how the player adjusted it to. |
| 3 | Customization of the cross hairs on the HUD | Chose cross hairs appear before the player in the center of the screen |

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Aspect to be tested and according inputs for function** | **Expected outcome** |
| 1 | Course of motion of target | Targets to follow their given course of motion such as side to side, teleporting to designated locations etc |
| 2 | Health deduction of target | Targets to have their health deducted accordingly with where they got shot |
| 3 | Elimination of target | Targets get eliminated once their health diminishes and disappears/ repositions appropriately |

|  |  |  |
| --- | --- | --- |
| **Test No.** | **Aspect to be tested and according inputs for function** | **Expected outcome** |
| 1 | A timer to begin once the player begins a match, it should display to a resolution of seconds on the player's HUD, but also to its max resolution (i.e. to the millisecond). | A counter should appear before the player on the edge of their HUD (to the second) whereas another counter with a higher resolution will be running in the background. |
| 2 | A counter to accumulate the number of bullets fired by the player throughout the entire match. | A saved value of an according number of shots the player fired. |
| 3 | A counter to accumulate the number of bullets fired on target. | A saved value of an according number of shots the player fired on target. |
| 4 | A counter to accumulate the number of targets eliminated. | A saved value of an according number of target the player eliminated. |
| 5 | A counter to accumulate the number of headshots landed. | A saved value of an according number of headshots the player landed. |
| 6 | Statistical calculations produced from gathered values | A layout providing the user with their stats about their performance |

# C. DEVELOPING THE CODED SOLUTION

So far, I have approached the development of the game with a technical and theoretical approach. However, I must also consider the fairness of the game and make the mechanics manageable such as to make the jump height fair so the players can manoeuvre around their environment relatively easily. Furthermore, I must also consider the easiness of the game; this would come in many aspects such as:

1. Bullet velocity: if aiming is too easy, I can reduce bullet velocity so then the player would have to lead their shots which would add difficulty

2. Health of the targets: if the targets can easily be eliminated, then increasing their health will make it more challenging to eliminate them.

3. Motion of targets: if it's too easy to shoot the targets or too difficult, I can change their nature of movement accordingly to ensure the game remains fun but challenging

Just as I presented  the testing for my in-game functions before, I will take a similar approach to this part of game testing as presented below:

|  |  |  |
| --- | --- | --- |
| **Post development test:** | **Testing to be performed after development** | **justification** |
| Bullet speed | The difficulty of landing shots at targets | It's considered a core skill to be able to control your aiming with regards to bullet speed therefore making it too fast would make it to easy and making it too slow would make leading shots difficult |
| Health of targets | How many shots required to eliminate a target(ultimately the difficulty in taking down an enemy) | I must calibrate the number of shots required to take down a target to make it similar to a scenario in a real game. |
| Motion of targets | The nature of movement of targets such as their speed of motion and the directions they take. | Being able to eliminate stationary targets will only enhance a player's skill so much, so this is why moving targets will be used and they’ll appear in many forms and natures. |
| Motion of player | The sensitivity of the player’s movements such as jump height, sprinting speed etc |  |

It is likely that I have yet to identify all possible concerns about the mechanics of my game. Throughout my development, it's likely that more and more of these concerns about mechanics will appear to me, which I can then assess and test appropriately when time comes.

A possible way I can do so is by enforcing beta testing upon candidates such as my interviewees Mo and Jay. From this, I can get their viewpoints and take onboard any suggestions and concerns they raise such as additional game mechanics I should address to.

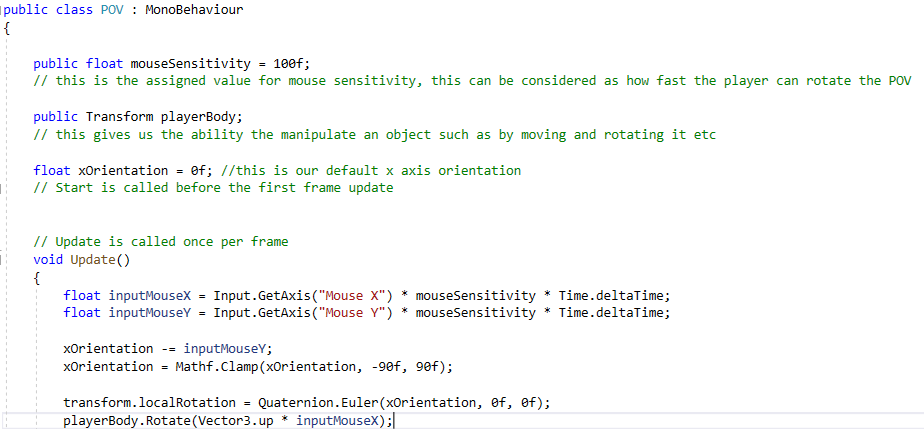
Camera view development milestone:

This milestone concerns about how the player can look around in their world by dragging their mouse around

To achieve this, I will use the quaternion library. Quaternions are an application of complex numbers which can be manipulated to rotate an object by a given angle across the 3 axes (x, y, z). In my case, the object of concern to be rotated will be the player camera. This is an interaction that the user will control which means I have to script some form of input from an external input device (a mouse in this case) to an according output in the game.

As I displayed in my flowchart design, I will also assign angular limits to the y rotation (because just like in normal FPS games, you can't lift your head or bend down your head over 90 degrees). Furthermore, I do not want there to be any terminating conditions, so I would like these mechanisms to run forever (as displayed with the looped flowchart). This can be enforced by integrating this function in a void loop.

While making this, I must also consider a specific mouse sensitivity for the mouse movement. Therefore, I will experiment with mouse sensitivity values until I find a suitable one for now. I will later implement the option for the user to customize their sensitivity but for now I will focus on just getting core functions working. I can allow my previous interviewees to test mouse sensitivity to see what they view as most suitable.



As I said, I do not want this POV (point of view )function to terminate so I have intergrated it into a void loop( which is unconditional)

The player's head will rotate around 2 dimensions, so I've made variable floats for each one (each obtaining their according dimension input from the mouse)

Here is me implementing the pre defined quaternion function as I mentioned earlier to rotate the head

Here is where I enforced the angular rotation limit so the player can't spin their head further than 90 degrees upwards or downwards (by using the pre-defined 'Math.Clamp/ function)

As the angular displacement = angluar speed \* time elapsed, in this case, my angular speed is the mouse inputs multiplied by the sensitivity and the time elapsed 'Time.deltaTime' (which is the duration of each frame)

I inverted to the y-axis purposely so I can obtain an appropriate vertical camera movement as without this alteration, moving the mouse forwards will rotate the player’s camera downwards and vice versa.

As a form of input validation, I implemented the use of vertical angular limits (90 degrees upwards and 90 degrees downwards) so that if the use attempts to vertically rotate their camera beyond these limits, such vertical inputs will be nullified and the player’s camera will remain at that vertical orientation unless the player adjusts the camera vertically in the respective opposite direction.

A diagram of a graph

Description automatically generated with medium confidence  
Here is an abstract diagram of bird's eye view of my character; consider the blue arrow to be the orientation of the player movement and the green arrow to be the orientation of the player's view. The first diagram shows the case of what would occur if I referred to the view camera to rotate rather than the entire body (which is undesired as then the player won't be able to move in the direction of their sight.

In conclusion, I will refer to the vertical orientation regarding the view camera so when the player looks up and down, only the camera spins rather than the entire player body. However, with the horizontal movement, I assign it to the entire player body, which inherits camera view, so viewing horizontally will twist the player and the view camera. I can enforce this by making the player inherit the camera so when the camera rotates up a down, it does so independently, but when the player wishes to rotate their view horizontally, the entire body including the camera will rotate correspondingly. Below is a video of the player camera in effect.

 VID1

Now I have developed a functioning rotating camera view with the control of the mouse, a concern I didn’t foresee was the visibility of the mouse cursor. The mouse cursor being visible on the screen is a big concern as it obstructs the view of the player, which is a crucial matter if the player was in a real situation in a live match. In conclusion, I researched a method to conceal and reveal the cursor when convenient. This is because when the user is on the main menu or elsewhere in the application apart from playing, they're going to need the cursor to interact with the application whereas they won't need it while in a game.

Here is a solution I found from (SEE INDEX) and they used a Boolean variable to express the visibility of the cursor (True corresponding to visible and False corresponding to invisible. I am likely to later edit this code when I am developing the main menu as I'll need to unveil the cursor when the user is not in a match.

A black text with black letters

Description automatically generated

VID2

USER TESTING:

I gave my original interviewees the opportunity to test the sensitivity of the mouse and they suggested keeping the sensitivity at 50.

Reviewing over this stage of development, I can claim that I met all my objectives (below) for my camera view function

* Movement of the user's mouse should orientate the player's camera view accordingly with their input and the sensitivity
* There should be no limit the horizontal change in orientation of the user's camera view (they should be able to view 360 degrees horizontally)
* There should be a vertical orientation limit of 90 degrees upwards and 90 degrees downwards

Character movement milestone

Moving mechanics:

This milestone concerns about all forms of movements the player will be able to do in this game such as walking around, running around, jumping and crouching

Firstly, for basic movement I want to use the WSAD keys as inputs meaning which is already recognized by the unity software. This movement will consist of moving around the x and z plane; therefore, I made 2 floats which will obtain which will obtain the x and z components of the user's input, respectively. The reason why I have given each of these a certain string name in closed brackets after the ‘GetAxis’ phrase is because the Unity software recognizes these terms and maps the inputs from the WSAD keys getting pressed into these variables (that it, ‘W’ for up, ‘S’ for down, ‘A’ for right and ‘D’ for left)From these both, I can then merge them into a single vector which sums up all the current motion input into a single vector(Vector3 in this case) which is done by getting the x component test and multiplying it a rotation function(transform.right) and vice versa for the z axis.

To then enforce this movement, I implement it in a pre-defined function movement with 3 parameters: the resultant vector(move), a movement speed (I will primarily assign this a constant value) and a change in time. This is because the formula for distance is speed × time = distance. Theoretically this would result in a gradual increase in the distances that the player would move for each time they press a movement button. Therefore, I use this function called 'Time.deltaTime' which inserts the duration of each frame as an argument for the function so then the distance travelled within each frame is independent from the other. This ultimately provides us with smooth movement.

A screenshot of a computer program

Description automatically generated

vid3

Jumping mechanics:

Using decomposition, the factors I have coded to create the jumping mechanics are:

1. Some form of vertical acceleration (like gravity). This is needed if I am to maintain a realistic jump movement rather than just shifting the player up in the air temporarily at a uniform velocity and then lowering them at a uniform velocity (which wouldn’t be a realistic jump movement)

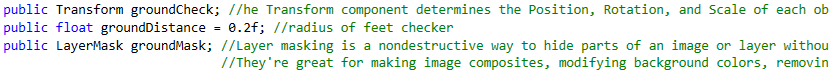
2.    a variable that dictates if the player is touching the ground. Logically, a player should only be able to jump if they're primarily standing on a surface, otherwise they'll be able to basically fly

3.    A suitable jumping speed.

I have enforced this mechanic by implementing these SUVAT equation (a style of formulas in mechanics that relate acceleration, starting speed, finishing speed, time elapsed and distance) v^2 = u^2 + 2 \* a \* s and v = u + a \* t

The first equation will be used for the upward jump and I am concerned about the jump speed, so I rearranged the equation to for v(the final velocity) which becomes u = sqrt(-2 \* g \* h) where g is gravity and h is the jump height(both are assigned variables which I will let my interviewees experiment with to provide me with favorable jump heights and gravity value). I have ignored the v^2 component as that represents the finishing speed which will be zero as the player will have no vertical velocity before they jump. I have this function all integrated in a conditional loop which requires 2 conditions: the player pressing the jump button(spacebar) while they're in contact in the ground (otherwise they'd be able to hop midair and even fly).

Floor detection:



Here I defined the state of the character body being in contact in the floor as a Boolean variable. I have also declared the following variables in respective order:

1. This acts at the coordinate of the bottom of the player body and will be used as a small invisible sphere beneath the player for collision detection

2. This is a constant which will be used as the radius of a small invisible sphere beneath the player for collision detection

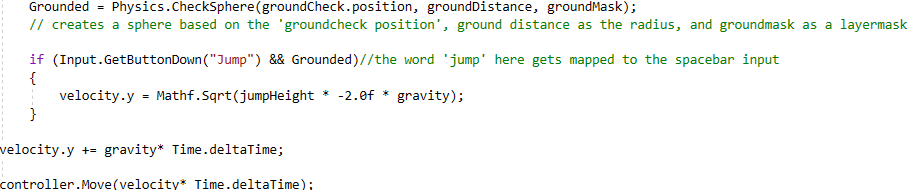
3.  This will act as a detection layer of a small invisible sphere beneath the player for collision detection

With these 3 variables, I can implement them into the 'Grounded' Boolean to use a pre-defined function known as 'Physics.CheckSphere (as shown in the first line of code on the previous picture) and it takes these 3 variables as arguments and implements them accordingly to construct what it known as a capsule collider beneath the player to detect if it's touching a floor(which would occur if it gets overlapped by a floor). With these 3 arguments, the function constructs this capsule collider and checks when it is obstructed (which will be when the player is on the ground) in where it will return the value of 'Grounded' as True, or otherwise false.

The second equation concerns about the free fall of the player and once again, I am concerned about the jump speed, so I keep the equation rearranged as it is but ignoring the u component instead as our starting speed (for the free fall) is zero, leaving me with v = at

As time progresses while the player is airborne, the downward velocity component will eventually overcome the upward velocity (due to the it being accumulative, hence me using +=) resulting in the player to eventually begin descending.

As before with the normal movement, the total resultant vertical velocity of each frame gets multiplied by a frame duration (distance = speed \* time) to provide an appropriate displacement along the y axis while jumping.

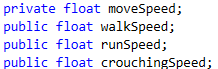
As a form of input validation, I enforced the requirement of the space key being pressed while the player is in contact with the ground (when ‘Grounded’ is true). It's essential that the condition of the player touching the ground must be true otherwise they'd be able to hop midair and even fly.

vid5

vid4

State alteration mechanics (crouching, standing/walking and sprinting).

Firstly, I've allocated a set speed for walking, crouching and sprinting alongside my original ‘moveSpeed’. I have now changed this variable to private because it will make it hold the values of the other movement speed variables and nothing else.



Secondly, I created 2 variables, one for the keyboard key of left shift and the other for 'C', for sprinting and crouching, respectively. These variables possess the type of variable of 'KeyCode' which allows them to store a designated keyboard key.

For movement speed, I created an if statement which requires the parameter of the player being in contact with the ground at that they're currently pressing the sprint key( I have referred to the necessity of pressing the sprint key as in that it needs to be held down for it pass as a parameter). If both conditions are met simultaneously, the movement speed variable gets allocated 'runSpeed’.

If the requirements for this if statement aren’t met, then the ‘moveSpeed’ variable will instead get allocated the ‘walkSpeed’ value.

vid6

For the crouching mechanic, I created an if statement which requires the parameter of the player being in contact with the ground (they can't crouch mid-air) while the crouch button has been clicked (in this case, the crouch button just has have been clicked rather than be held) for crouching to occur. When these conditions are met, the entire player body gets transformed. This is done by using this pre-defined transform function which converts the scaling dimension of the object of concern (the player in this case) and it will take in 3 arguments, which will be the axis scales (x,y,z). For crouching, the only axis that'll be scaled down will be the y-axis (as the player will be reducing their height when crouching). However, I want to keep the original scale size of the player's x and z components so I can do this by providing the 'local' prefix to the x and z components for my argument. This just means that it'll take the player's current x and z scales as an input, resulting in there being no change to them dimensions. For the y axis scaling argument, I just insert a defined constant for the y scale (which will be ultimately the new height for the player) that is less than the original y scale of the player

Likewise, I created an additional if statement which maintains the player's original dimensions when the crouch button is up (not pressed).

When the crouching mechanism does execute, the vertical compression occurs with regards to the centre of the body rather than the surface they’re on. Below is a visual representation:

player

floor

floor

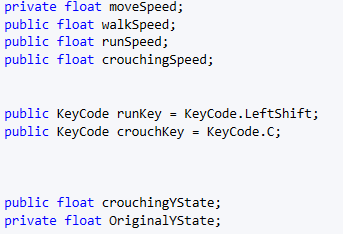
player

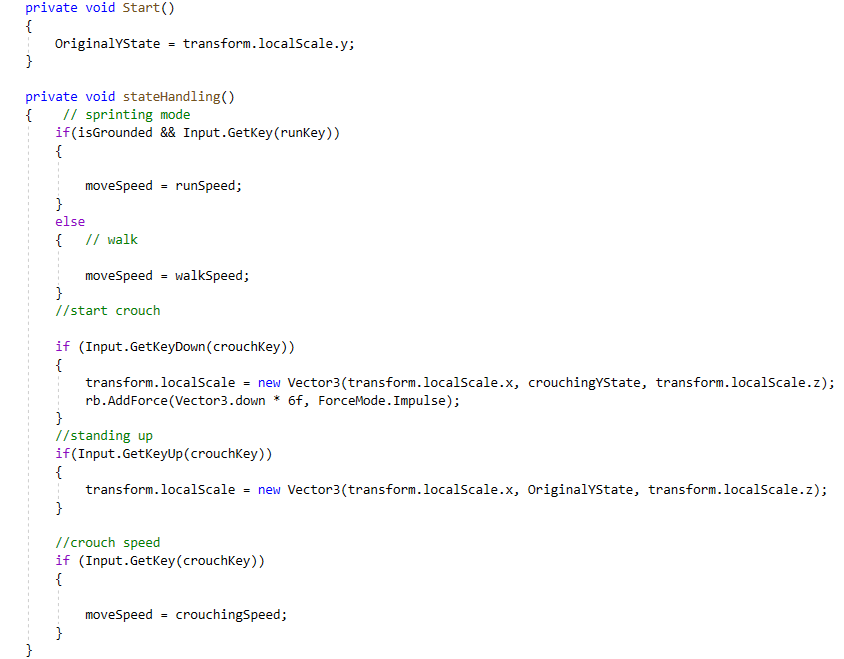
player

On the right is what truly occurs when this vertical compression occurs. Therefore, when it occurs, the player is temporarily airborne and will fall. As undesirable as this consequence is, it's inevitable.

By taking an abstractive approach, when this occurs (crouching), I can apply a significant downwards force instantaneously. By doing this, the player will be quickly placed on the ground when crouching without the user paying much attention to it occurring.

Below are all the pre-defined variables as well as the code that governs these mechanics:





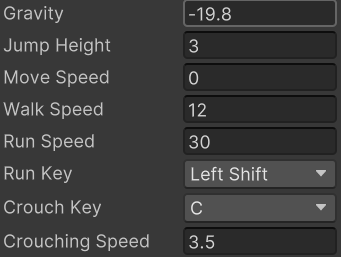
I implemented input validation here by enforcing the requirement for the user to be already in motion for them to be able to sprint. Furthermore, I also implemented the requirement for the user to be holding the crouch button in order to use it to prevent them from crouching by just clicking ‘c’.

 Below is a video of me presenting the crouching mechanics in effect, with the noticeably slower movement speed and the incapability of sprinting and jumping when I attempt so. Furthermore, this video does also present the body collider behaving correctly when the player’s course of motion is obstructed.

vid7

USER TESTING:

Now I have inserted a designated walking, sprinting speed as well as a jumping sensitivity and a gravity constant, I gave my original interviewees the opportunity to test the sensitivity of each of these aspects. I made these variables public, so they were able to alter it how they desired through the inspection tab. Below are the adjustments they’re now satisfied with.



Reviewing over this stage of development, I can claim that I met all my objectives (below) for character movement:

* A and D key should move the player left and right(respectively)
* W and S key should move the player front and back(accordingly)
* Space key should make the player jump and temporarily elevate them
* Body collider detector when in contact with a surface in any of the 3 axes
* Scaling detector detects whether an obstruction(along either the x or z plane) can be scaled over
* Tab key getting pressed simultaneously with a at least of the WSAD keys.
* C key getting clicked and changing the stance of the player from upright to crouching(or vice versa)

Shooting milestone:

For the arsenal of the game, I will be using this folder for my weapon sprites as it contains sprites for weapons. For now I will focus on implementing fully the shooting functionality on a single weapon type before I do so on other types. Using decomposition, the factors that must be attended to for this shooting function are:

1. Keeping the weapon fixed and parallel to the player’s view so where they move and look towards, the weapon points towards it too.
2. Have bullets to be shot out from the weapon when the shoot  button is pressed and for them to follow a trajectory that points towards where the player faces (at the moment it was shot)
3. The bullet(once shot) should disappear once it contacts a surface
4. A counter which represents the amount of bullets in the weapon remaining and it should decrease incrementally with each fired shot
5. A reloading function that occurs when the weapon is out of ammunition when the player requests to reload
6. Likewise, another value like this be used but for the ammo pool (the bullets remaining excluding those that are currently in the weapon)

Firstly, I created an empty object and assigned a gun sprite onto it.

I obtained a file containing a set of free weapon sprites (each weapon consists of the 3D model of the weapon and their corresponding skin sprite, which is like the cover pf the 3D model and provides it with the visual features)

To then achieve the first objective, I made this weapon inherit the features of my player (ultimately clipping it on) therefore making it able to move and look around correspondingly with the player. Utilizing this inheritance feature is better option than rewriting my previous camera view and movement script and assigning it to the weapon. As seen here...... the weapon remains in the same position in the player’s perspective wherever they look around



vid8

By taking an abstractive approach, rather than keeping the weapon’s bullets stored within the weapon and them exiting the weapon barrel when shooting, I can instead instantiate the bullet at the tip of the barrel of the gun so when the weapon is firing, it will still look almost visually identical. To do this, I created an invisible 3d object (so it doesn’t obstruct the player’s view) and fixed it at the tip of the barrel of the gun sprite, using the same inheritance technique as I mentioned above about clipping the gun the player’s view. This created object will act as a designated spawn point so when the player presses fire, the bullet gets instantiated at that given point (which is at the tip of the gun barrel).

When firing, bullets will constantly be reappearing from the weapon barrel and then outwards. Therefore, I can’t implement such a feature by treating these bullets as a standard GameObject( such as a box in the surroundings), instead I must use the bullet as a Prefabricated GameObject. Prefabs is a special feature that can be assigned to GameObjects which preserves all the properties of the gameObject as a reusable asset.

Here below is the bullet design and it being implemented as a PreFab, I have kept it as a small sphere as I am not concerned about its aesthetics and with the speed of the projectile, the player would barely be able to see it anyway.

A screen shot of a cellphone

Description automatically generated

In addition to that, an abstract approach I took when designing this projectile is by purposely giving it a spherical shape. By doing this, the object would not require a primary orientation and therefore I do not need to be concerned on creating code that will correct the orientation of the projectile

BULLET MILETONE:

Shooting direction:

When shooting, I must provide a given trajectory for the bullet to take once its instantiated. The direction should be towards a point at the center of the player’s perspective

A blue and purple rectangle on a black background

Description automatically generated

Here is a diagrammatic explanation of the trajectory I am trying to attain. The upper image shows the outcome if I were to center the trajectory of the bullets to be collinear with the gun itself. In this case, it will not be accurate even if the user was to overcome it, it would be difficult to adapt with. Therefore, my shooting direction should be towards the object. To achieve this I have used a function known as ray casting. Ray casting allows me to beam an invisible projection from a given point and I can obtain the position of where this beam stops when it meets a terminating condition (which would be specified).

The condition in this case that will stop the invisible ray from projecting is if it touches any object, to express this condition I defined a variable of ‘hit’ with the data structure of ‘RaycastHit’ which stores data about this ray (viewPoint) such as where it collided, what it collided, has it collided etc. In this case, I am concerned about where the ray collides and if it has collided. For the argument of the ‘Physics.raycast’ function within the if statement, I use the ‘out’ keyword to return information about the ray and the function ‘Physics.raycast’ verifies whether that ray has collided.

If it does collide, the variable ‘targetLocation’ will get assigned the position vector of where the collision occured(relative to the origin)

If there is no collision, then this variable simply gets assigned the position vector of a position that it 100 units away from the user’s centre of view.

A screenshot of a computer

Description automatically generated  
With this given position vector of where the player is pointing, I can then use it to calculate the vector to give my projectile to travel along. To do this, I subtract the position vector of the position of the tip of the gun barrel (referred to here as ‘ProjectileSpawnPoint.position’) from the position vector of the target point.



Here is a visual explanation of this process

A graph of a mathematical equation

Description automatically generated with medium confidence

Bullet creation:

Here is the variable ‘projectile’ which is assigned the data type GameObject as I wish to make it be able to interact with other objects. To generate this object, I must instantiate it which can be done by used with a predefined function in unity and provide it with 3 parameters:

1. An animate body for it to be in(in this case the bullet PreFab I defined earlier),
2. Spawning position(in this case the tip of the barrel)
3. An orientation. I want the starting orientation to be the ‘natural orientation’ which can be done by expressing the orientation with the identity quaternion.

Once the projectile is instantiated, in order to give it movement, I must refer to its rigid body component. Just like how I used rigid body for the player movement, using it on the projectile here allows me to manipulate physical aspects of it such as its weight, momentum, etc. In this case, I have used the ‘AddForce’ function to alter the force of the projectile’s rigid body. This too require requires parameters which are:

1. A direction to travel, which is calculated from the ‘Trajectory’ function returning a trajectory called ‘FiringDirection’ which is normalized (meaning it is treated as a unit vector).
2. A velocity for the object (the projectile) to move at.
3. The type of force to apply. In this case, the projectile should be projected instantly and this is done by referring to the predefined function ForceMode.Impulse

I will later on encode the functionality of the projectile hitting targets and objects and then terminating the projectile. However, I must also consider what to do if the projectile is just fired off into space, not colliding with any object. If this was to occur, the bullet would theoretically travel on forever in the 3D world. This would mean that throughout the time of a user playing the application, all the projectiles that were fired would all be present in the game and never disappear. This would be undesired as throughout the course of the gameplay, the performance of the application would slow down gradually as there are more projectiles for the user’s computer processor to process. A solution I created for this is to implement a lifetime for the projectile; where if the time period between when the projectile was fired to the present reaches the lifetime duration, it gets terminated

To enforce this, I created a Coroutine responsible for this function. Coroutine is a way to perform an encapsulated method over an elapsed period rather than instantly once called. In this case, the elapsed period is the lifetime of the bullet (which I will specify).

Below I have defined the Coroutine as a private IEnumerator; this is a special interface type which supports iteration by pausing code execution and resuming it at given points.

To begin the time delay (the ticker for the projectile lifetime), I use the syntax ‘yield’ followed by ‘return’, in which I allocate the delay duration. Once this duration is exceeded, I apply the predefined ‘destroy’ function to the projectile which is a GameObject; this simply destroys the projectile.

A screen shot of a computer

Description automatically generated  
Here is the function responsible for the projectile firing. I inserted this into a function because this operation will be repeatedly used.

A close-up of a computer screen

Description automatically generated

Firing mode:

I want to implement at least 2 different firing modes, single fire and automatic. To do this, I have created these following primary variables:

1.  Firing – an empty Boolean which will be used to flag if the weapon is currently firing

2. ReadyToFire – an empty Boolean which will indicate if the weapon is ready to shoot again

3. FireRate – a fixed float variable which is used for the fire rate intervals (this will be tested and calibrated)

4.  ReChamber – this is a boolean which is used to indicate whether the weapon can shoot again

Just as I previously allocated states of motion in an enum in the movement mechanics, I will do the same here for the different shooting modes, fully automatic and single fire.

A variable called ‘currentFireMode’ will then hold whatever current variable within the enum is getting used.

I will store these different states of firing within a special data type known as an enum. Enums are used to store a collection of related and labelled variables, in this case firing modes. I will refer to this enum to help allocate appropriate firing modes when given conditions are met (such as allocating a fully automatic weapon the fullAuto mode)

The name of this enum variable is Firemode, however I created an additional variable called currentFireMode which will hold a copy of the selected state of the enum, which I can then utilise in my firing mode functionality.

A computer screen shot of a computer code

Description automatically generated with medium confidence  
I have implemented this fire rate control mechanism in a void function with a nested if statement. This if statement issues the fully auto firing mode if its selected, otherwise the single fire mode.

Fully automatic:

This function will work when the left click is getting held down, hence the use of the term ‘Input.getKey’ meaning that the operation will only take effect if whatever button specified (left click in this case) is held down

Firstly, I created a void function called rechamber which allocates the state of ‘True’ the boolean variables ‘ReadyToFire’ and ‘reChamber’. For this operation to get enforced, I’ve put before it an invoke function. Invoke functions work by entering 2 arguments, a given function to execute and a time delay before it gets executed. The time delay in this case is the fire rate duration; this will only work if the boolean ‘reChamber’ is true(because the firing mechanism should only work if the weapon is ready to shoot)

Single firing:

This will take effect if auto firing isn’t selected. In this case, the shooting mechanism occurs per click hence the use of the term ‘Input.getKeyDown’ meaning that the operation would execute if whatever button specified (left click in this case) is clicked down. This means that the left click must be re-clicked again if the method

Following this, I created an if statement which will require the parameters of ‘ReadyToFire’ and ‘Firing’ to be true for the shooting function to be called

A computer code with text

Description automatically generated with medium confidence  
Here is the view of the inspector tab of the weapon gameObject.

A screenshot of a computer

Description automatically generated

Another great advantage of this is that I can now use polymorphism; the ‘shoot’ script now governs all possible shooting mechanics I’ll require. Therefore, for weapons that I’ll make later, I can simply assign this script onto additional weapons (make them inherit it) and then adjust these public variables to match the nature of these different weapons such as adjusting their fire rate or their bullet speed etc.

For addition convenience, I made use of Unity software and created a tracer for the projectiles that get fired out (RESEARCH SOURCE IN INDEX). This makes testing much easier, especially if I’m reviewing fast projectiles

A screenshot of a computer

Description automatically generated  
Proof:

VID 9

Here is the weapon currently in single fire mode (as assigned on the inspector tab) and with a fixed projectile speed of ‘30’. Regardless of what the fire rate is, it’s futile during single fire mode. Throughout the video, the ‘ReadyToFire’ Boolean variable seems to flicker; this is expected as the weapon will very briefly be unable to be ready to fire when it is momentarily shooting.

The ‘Firing’ Boolean variable seems to remain unchanged but it does flicker to ‘True’ very briefly due to the weapon being on single fire mode and this occurring whenever the left mouse is clicked, hence it being unrecognizable.

VID 10

Here the weapon is currently in fully auto mode (as assigned on the inspector tab) and with the same fixed projectile speed of ‘30’. This time, I assigned the fire rate to be ‘1’ (meaning there is a 1 second pause between each fired shot). Due to the fully auto mode, the ‘Firing’ Boolean variable remains true as long as the left mouse button is clicked. Furthermore, the ‘ReadyToFire’ Boolean variable remains False during this state as the weapon is already currently shooting.

VID 11

This is identical to the 2nd video however in this case, the fire rate is assigned to ‘0.05’ and as presented, the weapon fires at a much faster fire rate

VID 12

Here is proof of the ray casting mechanism for the projectile trajectory. To portray this, I utilized the fact that whenever I want to reuse the mouse (by pressing escape), it gets allocated to the centre of view of the player. The center of the screen is also the direction of the ray cast which gets shot out from the center of the player’s view.

The fact that the bullet always meets to where this mouse is proves that it works

Beside the gameplay is the hierarchy tab which shows all present GameObjects and their drop-down boxes (if they’re a parent object and have child object(s)). The projectiles that get shot out are GameObjects themselves and I didn’t assign them as a child object so when they get instantiated (when the player shoots), they appear in this tab. Furthermore, due to me assigning a lifetime to these projectiles, they get destroyed and ultimately disappear from the hierarchy tab as evident in the clip below

VID 14

Reviewing over this stage of development, below are the objectives I can claim I have met:

* The bullet should follow a straight trajectory towards where the player momentarily looked at when they fired

Below are the objectives I can claim I have partially met:

* Right click should enable a shooting animation and releases a bullet. During this state, the movement should be lowered.
* There should be a variety of weapons, with each one having a unique feature such as different fire rate etc

Below are the objectives I haven’t met:

* Left click should enable an aiming animation. The accuracy should significantly increase but the movement speed will be significantly lowered
* Pressing R should enable a reload animation and an appropriate number of bullets should be subtracted from the ammunition pool into the weapon
* The weapon should become futile once the player is out of bullets

Adjusting projectiles to interact with the environment:

Apart from projectiles exceeding their lifetime, I have no other terminating conditions on them. This is why in the clips above, the projectiles were able to bounce off objects because up to now, they still behave with physics as they have the ‘rigidBody’ component on them. Now I will create a terminating condition for when the projectile contacts either a target or a standard object within the environment.

With the unity software, I can ‘tag’ objects which can be used within code for comparative purposes

Below is a clip of me assigning 3 GameObject tags (target and game environment)

VID 13

With these tags, I can use them for referencing within the ‘projectile script’

A screen shot of a computer code

Description automatically generated

I created a function which gets initiated when a certain requirement is met; the requirement in this is when the collider of the projectile (it has a collider due to it having rigid body) gets touched by a GameObject. The tag on the GameObject gets checked if its ‘target’, in which it’ll get destroyed. If the tag isn’t ‘target’, then the compiler goes to the nested if statement to check if the GameObject that has collided with has the ‘environment object’. If the GameObject doesn’t possess neither of these tags, then the projectile would behave as a regular rigid body and simply bounce off if it has yet to expire.

Below is a clip of me testing the code above. I purposely slowed down the projectile velocity to help visualize the results and with a high projectile velocity, its very difficult to distinguish the projectile tracer before the projectile itself gets destroyed.

VID 15

Reviewing over this stage of development, below are the objectives I can claim I have met:

* When a bullet hits a target, damage should be inflicted
* When a bullet hits any object in its environment (including targets). It should disappear

Below are the objectives that haven’t been met:

* When the user has bullets in their weapon and presses to shoot, a bullet should be shot.

TARGETS MILESTONE:

I originally wished to make an array of targets of different types such as airborne targets, stationary targets etc; however due to time constraints I will not be able to do so. Therefore, I will focus on what I believe is one of the most important target types for FPS target game. This target is a hybrid between stationary and moving target. I have designed it so that it remains stationary in its place but also teleports to random locations.

Firstly, I created a script for this target’s behavior, which is for it to teleport to different locations whenever shot at.

Inserting image...  
For the teleportation mechanism, I want it to be random to make it challenging and unpredictable for the player to guess the sequence of locations the target will move to. Therefore, I’ve decided to first create a set of variables that’ll behave as coordinate boundaries for the target to teleport within. I have primarily assigned them as public because I want to test the space that’ll be convenient for the target to teleport within.

A computer screen shot of text

Description automatically generated  
To effectuate the element of randomness, I made use of the pre-defined randomization function (UnityEngine.Random.Range). This function works by calling it and then inserting 2 parameters, a lower bound and an upper bound; the function then outputs a random value within these 2 given bounds. This outputted value gets assigned to the public variables I created beforehand (horizontalShift and verticalShift).

I was aware that the plane (floor) of my game is relatively low relative to the y-axis so for the ‘verticalShift variable’, I assigned the lower bound for the y-coordinate to be ‘5’ so then the target will never teleport to a location beneath the plane (out of the player’s sight).

I intentionally didn’t assign a randomizing variable for the z-axis. This is because I only want this teleporting object to move across a plane (consisting of the y and x axis). Therefore, I’ll assign it a fixed value.

Just as I did before for the projectile, I created a function which gets initiated when a certain requirement is met; the requirement in this is when the collider of the projectile gets touched by another gameObject. In this case however, the tag that the if statement’s condition requires is the ‘target’ tag. If this condition is met, then the target will get teleported. This is done by using the pre-defined ‘transform.positon’ function, which requires 3 parameters of a value for each of the 3 axes. For the x and y coordinate parameter, I passed the ‘horizontalShift’ and ‘verticalShift’ variable respectively  This function uses a given coordinate and translates the GameObject to that position; I want this given position to always change each time so use the keyword ‘new’ meaning that each time that line of code is met, a new instance of an xyz coordinate is made for the target to get translated to.

Here is a clip below of it in action.

VID 16

Further testing:

I once again gave my interviewees the opportunity to test this additional aspect of the game, particularly the bullet velocity and the fire rate of the full-auto mode. I have made these both public variables so they can adjust it so what they believe is suitable and sensible. Here are the following settings they’re satisfied with.

A screenshot of a video game

Description automatically generated

Reviewing over this stage of development, below are the objectives I can claim I have met:

* Course of motion of target

Below are the objectives that have been partially met:

* Elimination of target

Below are the objectives that haven’t been met:

* Health deduction of target

Menu MILESTONE:

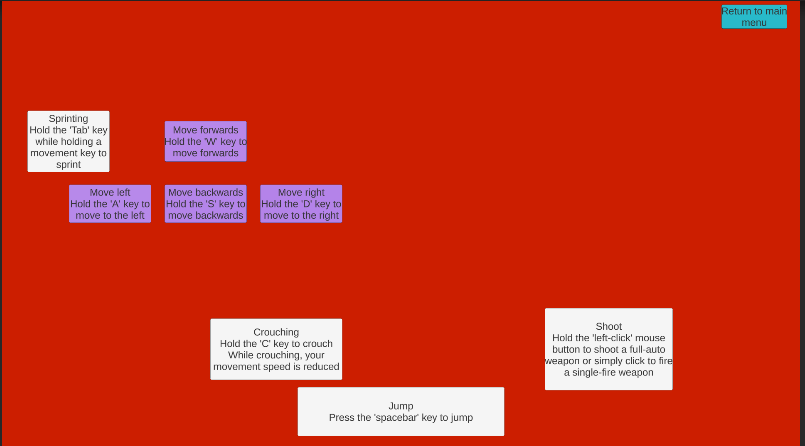
When the user first uploads the application, they’ll be presented with a main menu before them which they can interact with. The main menu will have to be in its own scene. In unity, scenes are just parts of your game which hold a collection of section(s) of the game. As an example, if a game had 3 different worlds, each of these worlds would be in their own separate scene. I will be taking an approach like this with the main menu.

Firstly, I created a new scene called main menu for the main menu and arranged these icons in the scene using the unity software. These icons are UI components are uploaded on the canvas; the canvas is a generated mesh for view and regenerates this mesh accordingly to what UI elements are added. Each of these have a text feature, which I filled accordingly to what those icons will represent

Here is an image of what the main menu scene looks like now

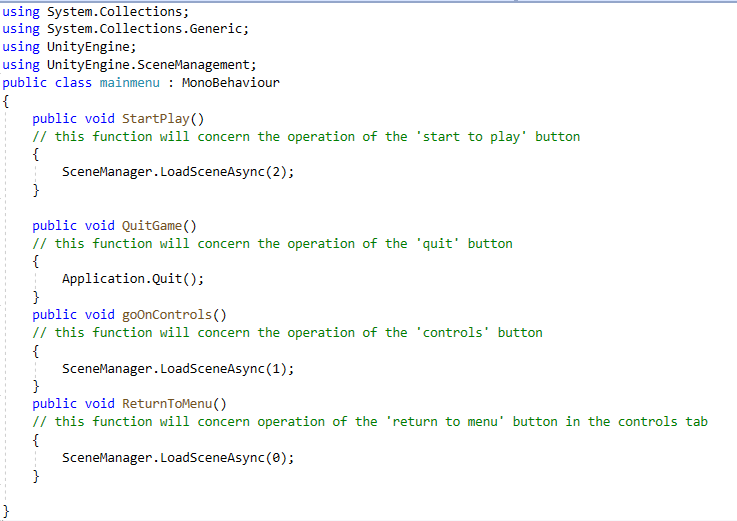
The open world is still visible to the player camera behind these icons so on the inspector tab I adjusted the background filler of the empty space of the world to be a solid color, despite the open world till being visible to the camera as displayed here

vid 18

I repeat this same process but for the controls so that they’re also in their own separate scene, then by using UI icons as I did previously with the main menu icons, I arranged them in a manner like the keyboard’s arrangement of keys. In the corner, I placed a return to menu button which I will elaborate on its function later. Here is an image below of the controls page 

Alterations between unity scenes:

Using the unity software, I allocate each separate scene in the ‘build manager’ its own reference code in numbers I can use these values to address to specific scenes to be uploaded. Here is the code that will concern the operation of switching between scenes. Here is a source I used to aid me with utilizing the unity software utilities(SEE INDEX)



I mentioned the use of ‘systems.collection’ which is a utility system that allows C# interact with the scenes of the application

With all the functions to switch between scenes set, I attached this script to the interactable buttons in controls scene and the menu scene and assigned each of these buttons with one of the specific functions in the script

Further feedback:

After presenting my interviewees with the menu and its function, they’re satisfied with it and provided no further criticism

Reviewing over this stage of development, below are the objectives I can claim I have met:

* The user left-clicking the play icon
* The user left-clicking the controls icon
* The user being able to exit the game to return to main menu
* The user left-clicking the exit icon

Below are the objectives I have partially met:

* The user left-clicking the return icon upon entering any of the 4 pages above

Below are the objectives I did meet:

* The user left-clicking the stats icon
* The user left-clicking the settings icon

HUD MILESTONE:

Firstly, I created a new script uniquely for the HUD display and included the text mesh pro library (TMPro). This library allows me to format text on the UI in more simplistic ways.

Time display:

Firstly I created 2 variables for this time display mechanism, one variable will hold the current time and the second variable will simply replicate it in a form that can be displayed on the UI



Now I need to utilize this TextMeshProUGUI variable by assigning it to a UI gameObject, which I can create and arrange appropriately in the user’s UI. Its purposely public so then I can assign it to a UI gameObject.

vid 17

vid 20

To increment the increase in time, I added used the ‘+=’ function with the argument of ‘Time.DeltaTime’; this simply is the duration of a single frame. By accumulating the duration of each frame to a total count, an accurate timer can be produced. To have this iterative process maintain such consistency (that occurring per frame), I inserted it into an ‘Update’ function so it can execute each frame. I was aware that a duration of a frame will be a fraction of a second, so I purposely defined the ‘liveTime’ variable to be a float to deal with such numbers

Once I have the numerical value of the time, I allocate it to the Timer variable (which is used to present this value on the UI). For this to happen, the variable that is to be presented must be in the form of a text (as seen with ‘Timer.text’) must be a string to get presented so I casted it into a string (as seen by the ‘ToString()’ function). Moreover, by making it a text type, this allows it to be not interactable with the user in case they are to click on it

A white background with black lines

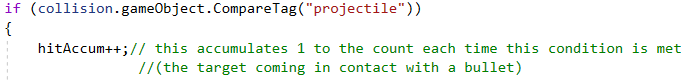
Description automatically generated  
STATS MILESTONE:

I wish to present before the user the accuracy of their performance, particularly the accuracy of their shots. To do this, I need:

* The total shots they fired; I can acquire this by creating a counter that accumulates whenever the weapon is fired.
* The total shots on target; I can acquire this by creating a counter that accumulates whenever the bullet touches the target.

The first variable can be obtained from the ‘shoot’ script as it possesses the shooting function, which I can utilize as a trigger to initiate each increment to the total shots fired.

  
  
I will take an identical approach for the variable concerning the number of shots on target.

  
  
The concern here is that the script which concerns the HUD, the one that hold the total number of shots fired and the one that hold the total number of shots on target are all on separate scripts and therefore I can’t use them all at once to calculate accuracy as I would usually do with other processes when all the variables are local. In this case, these variables aren’t local to each other and there is no way to make them local and have them to behave logically as they should.

To tackle this problem, I purposely assigned these both count variables to be public as well as their classes, as presented below



This it from the ‘target’ script



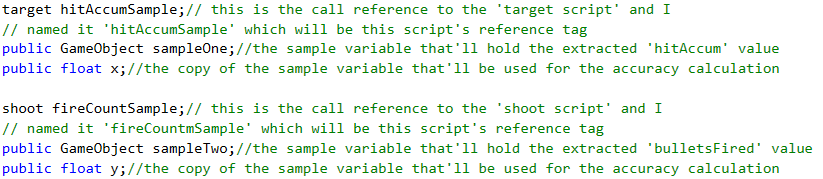
This is from the ‘shoot’ script

By making these classes and these specific variables within them public, I can make use of inheritance which will allow me to use these variables from other classes within the HUD script

To do this:

1. I must firstly make a reference to both scripts within the HUD script
2. Secondly, I’ll create 2 ‘sample’ variables (corresponding to each of the 2 scripts), these will be used to hold whatever piece of data I desire to fetch from an alternate script.
3. Another pair of empty variables (corresponding to each of the 2 scripts). These will simply act as ‘holders’, holding a copy of each of the sample variables and can be used freely within the script; in this case they’ll be used for the accuracy calculation.

Here are the following defined within the HUD script



Nevertheless, to present the accuracy I will take a similar to approach to that of the ‘timing’ where I had a variable holding the actual time and a copy of it to present on the HUD. In this case, it’ll be a variable that hold the accuracy value and a copy of it (which has a datatype of TextMeshProUGUI that is designed for UI application) used to present onto UI

Further more, as I did with the timing mechanism, I have utilize this TextMeshProUGUI variable by assigning it to a UI gameObject, which I can create and arrange appropriately in the user’s UI. Its purposely public as well so then I can assign it to a UI gameObject.

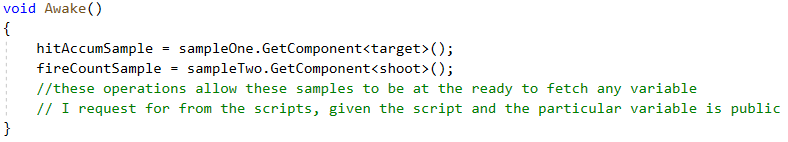
vid 19

vid 21

With all these variables defined, I can calculate the accuracy and be capable of uploading it on to the HUD.

Firstly, I must instantiate both of my sample variables by fetching the data I want to fetch from each script (‘hitAccum’ from the ‘target’ script and the ‘bulletsFired’ variable from the ‘shoot script)

I want this to occur initially before the rest of the program is executed to prevent possible logic errors if these sample variables happen to be null at first. Therefore, I inserted the instantiation operation inside an ‘Awake’ function. ‘Awake’ functions get prioritized to be executed beforehand before the rest f the code in a script gets executed.



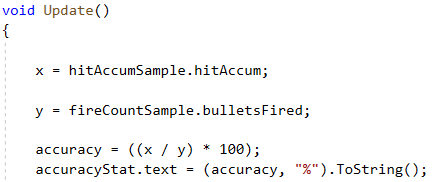
I want the provided accuracy to be real-time therefore I must also have the accuracy calculation to be done within the same ‘Update’ function as the time counter.

Secondly, I assigned the ‘holder’ variables x and y the specific variables I want fetched from the inherited script so I can directly implement them to calculate accuracy.

Thirdly, I assigned the ‘accuracy’ variable the percentage accuracy, this is calculated by dividing the hits on target (hitAccum) by the total shots fired (bulletsFired) and then multiplying it by 100.

Then to upload it onto the UI, I take the same approach as I did with displaying the time

Here is the code below:

  
Here is a clip below of the accuracy presentation in effect.

vid 22

Interviewee feedback:

After presenting this to my interviewees, they replied with 2 criticisms.

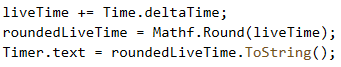
1. They did not like the arrangement of both the live clock and live accuracy stats. They told me they preferred to have them next to each other in the top left corner of the HUD. Therefore, I made such rearrangements on the unity software as presented below
2. Although they were appreciative of the degree of accuracy of accuracy of the displayed (as previously shown having the precision down to many decimal places), they suggest that I lowered their accuracy so that they don’t obstruct the HUD too much mainly. Nevertheless, the time increasing per frame rate and with such accuracy meant that it was constantly increasing, and its increase was visible and visually distracting (due to the quickly changing digits).

To deal with this second issue, I first decided to tackle the accuracy statistic. I returned to where the ‘accuracy’ variable is defined and applied the ‘Mathf.Round’ function. This function takes in a numeric parameter (in this case the accuracy value) and rounds it to the nearest whole number. Ultimately, the uploaded string presenting the accuracy on the HUD is much shorter (at most 3 digits) and therefore less obstructed, which is what’s desired by my interviewees. Below is the code presenting the amendments I made for this specific problem



Secondly, I will tackle the concern with the timing display on the HUD. However, I realized I can’t implement the same approach as I did above for accuracy in which I simply rounded the accuracy value. This is because I designed it to increment with the duration of each elapsed frame (a very small duration). Therefore, if I were to apply the round function to it, the time count will always round down and never make it past zero despite time elapsing. As a solution, I created an additional empty variable that’ll act as an intermediate carrier for the time count. I'll have the live time counter as it is to maintain its accuracy and I then assign this ‘carrier’ variable this live time value but rounded. By doing this, I preserve the accuracy of the time counter as well as providing a rounded time duration on the HUD (to the nearest second). Below is the ammended code for this specific problem.





Here is a video below of the HUD icons in action after the amendments.

vid 23

EXIT BUTTON:

To add an exit button, I followed a very similar procedure to with how I managed the HUD of the main menu etc. I then allocated it interactable HUD button with the ‘main menu’ script which I previously made during the main menu development and this script governs the functionality of swapping between scenes throughout this application. I intend that when this button is clicked, the user is returned to the main menu. This makes it (in terms of functionality) identical to the exit button on the controls window. Therefore, I can reuse the function that I created for that button to this one as well. Here is a video of me testing this mechanisms.

vid 24

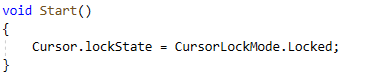
After testing, it has appeared that the function of exiting the main game doesn’t work despite no syntax error appearing on the IDE nor the unity software.

I recognized that this is logic error is due to the mouse concealment feature I implemented

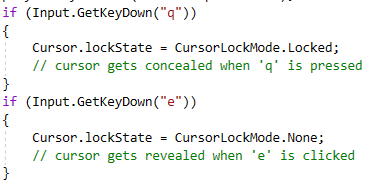
When I was coding the camera view mechanics and I wanted to hide the mouse from the player’s view. This feature revealed to be incongruous with this HUD feature as this ‘quit’ button requires a click input.

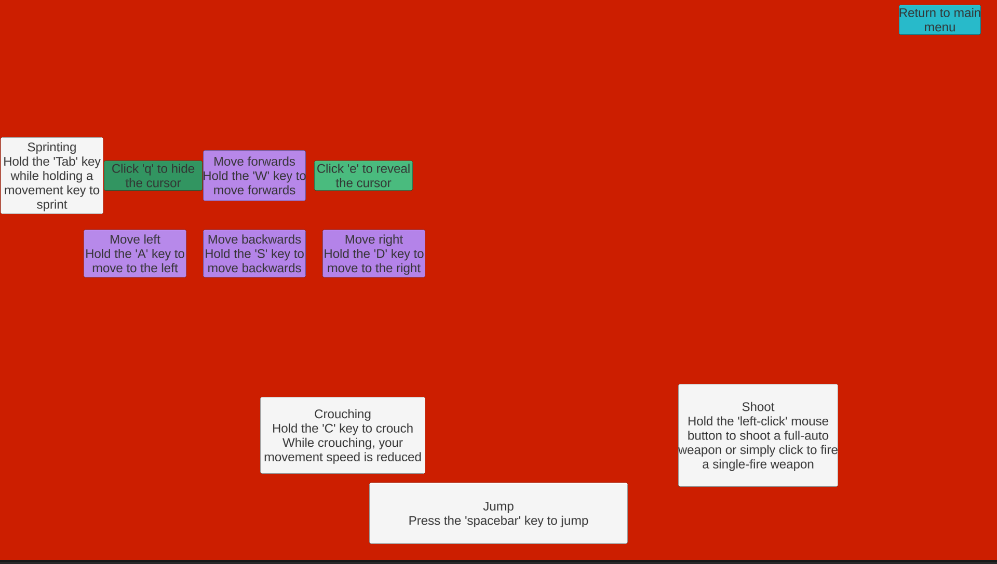
To solve this, I pursued with some research and discovered an alternate method of concealing and revealing the cursor in a way that will be congruous with the ability to click on the ‘quit’ button (link provided at index)

I completely removed this previous code which governed the concealment and revealing the cursor



I then replaced it with this code. This code operates in a more manual manner, and I allocated the keys ‘q’ and ‘e’ for concealing and revealing the cursor respectively. This functionality is enforced by using a pair of if statements that will apply these specific changes to the cursor’s profile depending on what was inputted.

  
Due to me adding these 2 additional key controls (‘q’ and ‘e’), I updated the controls page that can be accessed from the main menu according to these new additional features. I added these additional UI icons to portray the function of key ‘q’ and ‘s’ in an identical way as I did when doing so for the rest of the controls. Here is an image below of what the controls guide window looks like



Reviewing over this stage of development, below are the objectives I can claim I have met:

* A counter to accumulate the number of bullets fired by the player throughout the entire match.
* A counter to accumulate the number of bullets fired on target.

Below are the objectives I have partially met:

* A counter to accumulate the number of targets eliminated.
* Statistical calculations produced from gathered values
* A timer to begin once the player begins a match, it should display to a resolution of seconds on the player's HUD, but also to its max resolution(i.e to the millisecond).

Below are objectives I haven’t met:

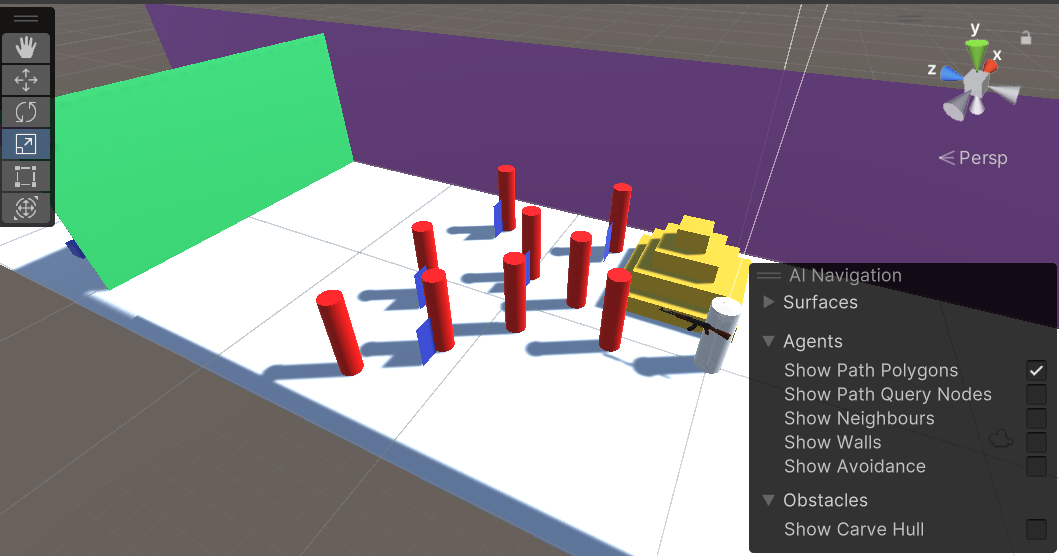
* A counter to accumulate the number of headshots landed.

Another matter that was raised by my interviewees which I haven’t planned is obstacles to place in the environment so I scattered some objects in the environment that the user can traverse along if they wish to while practicing their aim. I asked my interviewees and here are their suggestions below:

* Haphazardly arranged beams about the width of the player
* Small fences that the player can crouch upon
* stairs that the player can climb up

Below is the scene of the world now containing the features that my interviewees desired. As I did previously with all 3d objects in the world, I have given all these objects an identifier tag of ‘environment object’ (as I did with the floor and the green target board). This is so none of them interact with incoming projectiles and that projectiles disappear when they come in physical contact with them.

In addition, I added further barriers around the plane to prevent the player from falling or jumping off the world if they are to move near the edge. Below is an image of what the world now looks like.



# D. Evaluation

EVALUATION:

Below is the video of the entire program getting run

vid 25

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Aspect to be tested and according inputs for functions** | **Expected outcome** | Timestamp | Pass/fail/partial |
| 1 | The user left-clicking the play icon | The play page is uploaded, with a given option to return to main menu | 0:23 | pass |
| 2 | The user left-clicking the stats icon | The stats page is uploaded, with a given option to return to main menu | - | fail |
| 3 | The user left-clicking the settings icon | The settings page is uploaded, with a given option to return to main menu | - | fail |
| 4 | The user left-clicking the controls icon | The controls page is uploaded, with a given option to return to main menu | 0:09 | Pass |
| 5 | The user left-clicking the return icon upon entering any of the 4 pages above | The original main menu page gets presented to the user | 0:20 | Partial pass |
| 6 | Movement of the user's mouse should orientate the player's camera view accordingly with their input and the sensitivity | An appropriate change in the player's camera orientation with regards to their input | 0:27 | pass |
| 7 | There should be no limit the horizontal change in orientation of the user's camera view (they should be able to view 360 degrees horizontally) | No matter how far the mouse is moved horizontally, the user's camera view should also change accordingly. | 0:31 | pass |
| 8 | There should be a vertical orientation limit of 90 degrees upwards and a 90 degrees downwards | Regardless of the user's vertical mouse input, it should be ignored if its exceeding one of these vertical orientation limits | 0:42 | pass |
| 9 | The player should be able conceal the cursor by pressing ‘q’ | The mouse disappears when ‘q’ is clicked | 0:27 | pass |
| 10 | The player should reveal the cursor by pressing ‘e’ | The mouse reappears when ‘e’ is clicked | 4:08 | pass |
| 11 | A and D key should move the player left and right(respectively) | An appropriate change of positioning(along the x axis) according to the user's input | 0:27 | pass |
| 12 | W and S key should move the player front and back(accordingly) | An appropriate change of positioning(along the z axis) according to the user's input | 0:52 | pass |
| 13 | Space key should make the player jump and temporarily elevate them | An appropriate change of positioning(along the y axis) according to the user's input | 2:17 | pass |
| 14 | Moving while airborne (jumping). This test allows me to observe if the mechanics of gravity | While jumping and moving should result in a parabolic course of motion rather than a | 2:21 | pass |
| 15 | Body collider detector when in contact with a surface in any axes | Stops the player from proceeding to move any further along an axis (if obstructed upon so) | 1:20 | pass |
| 16 | Scaling detector detects whether an obstruction(along either the x or z plane) can be scaled over | Allows the player to scale over any obstructing objects under a given height. | 1:25 | pass |
| 17 | Tab key getting pressed simultaneously with a at least of the WSAD keys. | Allows the player to proceed in the same direction but with a greater speed(sprinting) | 2:00 | pass |
| 18 | C key getting clicked and changing the stance of the player from upright to crouching(or vice versa) | Changes the stance of the player, their profile will be reduced and so will their speed, they shouldn’t be able to jump | 1:45 | partial |
| 19 | Moving while in the state of crouching | The player should be able to move around freely while crouching and at a reduced speed | 1:47 | pass |
| 20 | When the user has bullets in their weapon and presses to shoot, a bullet should be shot. | A small projectile exits the gun and travels in a straight course relative to where it was shot from. | 2:48 | pass |
| 21 | Lifetime of the bullet. I can prove this by firing vertically upwards | The projectile should travel upwards but not return downwards | 3:00 | pass |
| 22 | When a bullet hits any object in its environment (including targets). | The bullet should completely disappear (not just become invisible as then it'd still behave as a bullet). | 2:48 | pass |
| 23 | Progression of the time counter | The incrementation of the time should increase accurately and I can authenticate it by matching it with timing of the recorded clip | 0:23 | pass |
| 24 | Accurate display of accuracy statistic of player’s performance | The amount of shots the player has fired and the shots they have on target should be used together to provide an accuracy percentage. I will display live accuracy on screen to validate the accuracy. | 4:20 | patrial |
| 25 | The target repositioning after each time its shot at | The target should teleport to a random spot upon the green screen | 3:13 | pass |
| 26 | Jumping nature | While mid – air, the player should be incapable of jumping as I repetitively attempt to jump | 2:12 | pass |
| 27 | When the player clicks on the quit button, they exit the application. | The application closes when the quit button is clicked | 4:27 | pass |

Usability testing:

In terms of usability testing, I specifically done this at this time stamp (0:09). In this test I tested the functionality of the controls page, which is there to help and guide players on how to use the application.

ROBUSTNESS TESTING:

In addition to my testing, I performed various robustness tests such as:

1. Erroneous data testing: I presented this by attempting to look up more than 90 degrees upwards and downwards 90 degrees; regardless of my mouse’s persistent vertical input towards these vertical angular limits, the player camera doesn’t go beyond these. (time interval) Another way I enforced this method of testing was throughout me testing the movement, I collided with a variety of objects to prove that they behave as an obstruction and that the player doesn’t permeate any object. I also purposely raised the walls to prevent the user from using the obstacles as a means of jumping off. Furthermore, I effectuated this method of testing with the UI icons on the controls page and the time and accuracy statistics on the player’s HUD in the main game. I continuously clicked on them to ensure that they don’t have any abnormal behaviors. (0:43, 1:00, 2:30, 0:10) These are the timestamps for the instances I described above respectively
2. Simultaneous actions: Throughout testing, I intentionally performed a set of actions simultaneously (such as moving and jumping while shooting) to prove that the application works smoothly and accordingly, even with a variety of different actions all occurring simultaneously. (03:37)

CRITERIA COMPARISON:

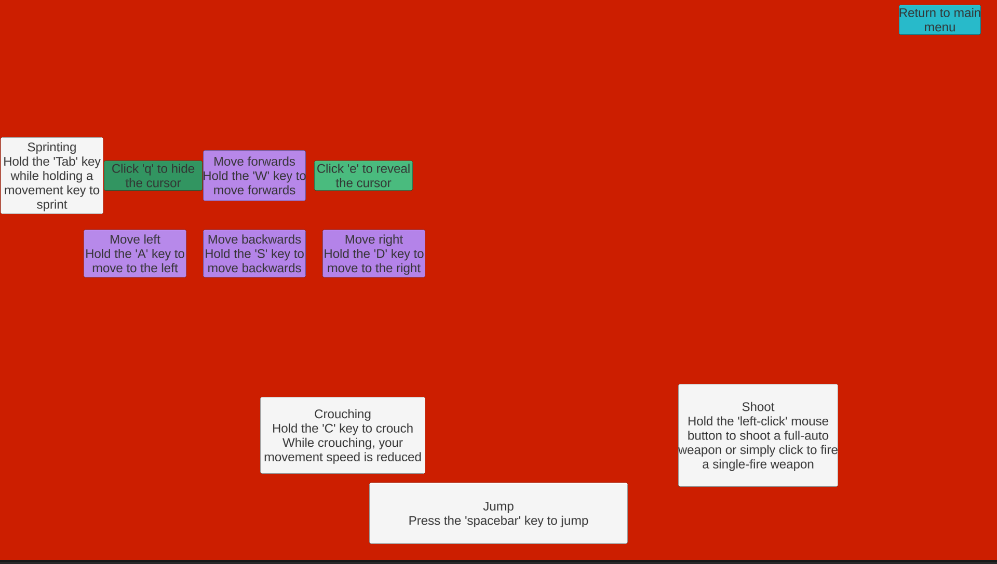
|  |  |  |  |
| --- | --- | --- | --- |
| No. | Criteria | success | Comments |
| 1 | To make a main menu that has icons such as play, settings, controls, quit | Partially met | The only feature I haven't implemented is the tab of statistics and settings. This was due to time constraints. However, to compensate for the absence of the stats tab, I uploaded the user’s stats on their HUD while they’re on the main game |
| 2 | Player can adjust certain features such as properties of their UI (FOV, mouse sensitivity etc) and features of targets (e.g. target speed, number of targets) | Not met | Due time constraints, I was unable to create a settings tab nor the functionality for adjustable settings |
| 3 | Player being able to do basic FPS movement and look around with their camera | Fully met | The player can move around and look around just as they would in any other first-person game |
| 4 | Player can choose from a variety of different weapons | Partially met | Although I created a superclass that contains all adjustable weapon features, I couldn’t implement them to multiple weapons due to time constraints |
| 5 | Player can view their performance in an array of statistical presentations such as rate of elimination, shot accuracy etc (all on the stats tab from the main menu) | Partially met | I uploaded the user’s stats on their HUD instead of on a stats tab. However, due to time constraints, I could only upload an accuracy stat rather than multiple others such as rate of eliminations etc |
| 6 | There are a variety of targets of different natures that the player can choose from to shoot at. | Partially met | Due to time constraints, I didn’t create and code various targets so I instead made a hybrid of a stationary and a moving target |
| 7 | For the player to be able to reload their weapon and aim with their weapon (which should increase the accuracy of their shot and reduce their movement speed) | Not met | After discussion with my interviewees, they informed me that reload mechanics would be futile in this application as the focus here is accuracy rather than the conversion of bullets, which would only be a concern when playing against other enemies |

BETA TESTING:

I offered both of my original interviewees Mo and Jay the opportunity to now test this application in its entirety. This will be beneficial to me as not only does it provides me a second opinion to what is this applications ‘key’ audience but also a chance for any additional bugs and flaws to be pointed out if any are present. To do this, I interviewed each one of them and asked for their opinion on each aspect of the application (SEE INDEX). I have summarized their comments and categorized them

|  |  |  |
| --- | --- | --- |
|  | Mo | Jay |
| Menu | Satisfied | Pointed out that when they left the main game and went to the main menu and then returned to the main game. Everything would restart so no data would be saved which some may dislike. |
| Movement | Pointed out that the weapon deforms while the player is crouching.  They disliked the fact that the player is capable of sprinting in any direction, this is because they believe that a player should only be able to sprint forwards and that all other movements (e.g. side-to-side) should be done at normal speed. This is because they believe these other movements are essential for limbing obstacles and adjusting positioning, therefore it would be unfavorable to do so at a high speed (sprinting). | Pointed out that the weapon deforms while the player is crouching |
| HUD | Disliked the color of the HUD text because it was hard to see the text when orientated towards a bright background, black font would've been preferable | Satisfied, although pointed out that they would’ve liked a feature where the controls could be immediately accessed while in the main game. |
| Targets | Satisfied with the random moving target as it focuses on how well a player can swap focus for their targets | Satisfied with the nature of the targets however would’ve much preferred airborne targets as then the skill of leading shots can take in effect. They also think the teleporting target should be able to move across 3d space rather than 2d |
| Shooting | Although they didn’t request it, they found that the tracers left by the projectiles are very useful for knowing where shots land. | Although I didn’t include crosshairs, Jay believed that the absence of crosshairs allows one to adapt and intuitively centralize their target on screen without assistance. |

Usability features:



The purpose of the control panel is to provide users with a guide on how to use the application. This is done by presenting them a topologically accurate layout of the control keys of the keyboard and the function. Furthermore, when necessary, the label on the control keys address how they’re particularly used e.g the crouch key label on controls page states that it must be held whereas the spacebar for jumping only needs to be clicked. The user can also access this controls page before and after playing so they’re not just limited to only accessing it before playing.

However, a flaw of this usability feature is that I provided description of how to use full-auto firing (by holding left-click). Although I did create this feature and proved its success, I didn’t implement it due to time constraints . Another concern is that I only provided usability features of features that I have implemented in the final model. This means due to me not implementing the other features I included in the design, I couldn’t provide usability features for those absent features. Ultimately, the controls page that I designed during the development stage is different to my true final controls page. Below is a copy of my designed controls page which can be compared to with the copy of my current controls page.

A screenshot of a computer screen

Description automatically generated

In conclusion, for the current features that I have implemented in my application, the usability feature (controls page) provides guidance on how to use all the existing features

Further development for usability features:

For each feature I have yet to include elsewhere in the application, I would focus on developing them primarily. Afterwards once I’m satisfied with its function, I will then add to the controls page a new icon concerning the usability of said new feature in an appropriate manner. As an example, in the future I may focus on and implement the feature of sliding (a relatively common maneuver in FPS games). Once I implement such feature I could then create a new icon in the controls page addressing the user on how to use this ‘sliding’ mechanic and any additional information about it. Below is an example of this controls icon I would add as an additional usability feature (corresponding to the sliding mechanic example I mention above)

SLIDING:

While sprinting FORWARDS, double-click the key X in rapid succesion to slide

(There is a 5 second cooldown)

Analysis of partially unmet criteria revealed within the testing and proposed future development for them:

Following the chronological order of my development, I developed the crouching mechanic before the shooting mechanic (particularly clipping the weapon onto my user). Clipping the weapon onto the player allowed it to ultimately inherit all features of the player such as moving along with the player, jumping with them etc. Inevitably, this resulted in the weapon possessing the feature of crouching. The crouching mechanic compresses/minimizes the vertical (y-axis) scale of the player to reduce their profile; consequently, this effect also occurs to the weapon as well as presented at this time stamp of the clip (1:45). This, of course, wasn’t a feature I was expecting nor is it desired.

I believe that this could be solved by making a separate piece of code that governs the behavior of the camera and the weapon when the player crouches. Rather than using a vertical compression of a given scale on the entire (which is what I did), I could instead only apply it to the body of the player and then I can vertically translate (move a given position in a 3D space to a new set of coordinates) the position of the player’s weapon and camera downwards

Another flaw I found with my testing which I didn’t observe during development was the fact that when my user approaches a wall and is very close to it, the weapon partially permeates through the wall. This is an undesired feature I didn’t foresee.

I pursued research to find if it’s possible to overcome this issue and below is a solution I found.

This solution utilized the layer transparency system in the unity software. This system can be used to add unique controls to how certain aspects of a game scene can render before the user and to how light can interact with certain objects in front of the player’s view.

To summarize, by creating an exclusive new transparency layer and assigning it to the player weapon, you can then go on the player’s camera settings and adjust the culling mask (this allows you to manipulate the appearance of certain objects in the player’s view e.g whether light reflects of it etc). One can adjust this ‘culling mask’ setting so that the weapon in the player’s view doesn’t

EVALUATION OF ADDITIONAL PARTIALLY MET CRITERIA:

* Player can view their performance in an array of statistical presentations such as rate of elimination, shot accuracy etc (all on the stats tab from the main menu)  (FROM MY CRITEREA IN THE ANALYSIS STAGE PAGE 21-22)

The crucial role of this application is to provide users with a quantitative display of their performance. However, due to time constraints, I couldn’t enforce this feature in the same fullness as I intended to in my design. Therefore, I implemented a provisional form of statistical representation in which the stats of the player were immediately uploaded before them rather than on a stats tab that could be accessed within the main menu. Moreover, I was only capable of presenting the accuracy statistic to the user as I believed that was the most crucial and foremost statistic in FPS games. As a result, I was also incapable of implementing the mechanics of statistical comparisons which is another feature I intended to implement to provide users a display on how they performed on different occasions and how in particular they differed etc

HOW I WOULD GO ABOUT THIS MATTER IN FUTURE DEVELOPMENT:

In future development, I will proceed with constructing these mechanics that I designed and intended to construct during the design stage while maintaining correspondence with my interviewees for their valuable input such as how I should arrange the statistical icons on the stats tab that can be accessed through the main menu.

However, from my current application, although the immediate uploading of the user’s statistic was a intended as provisional feature I believe that it’s an ideal feature for the user to have some form of immediate update on their performance before them. I could present just a narrow variety of live statistics before on the player on their HUD while they’re in game as presenting all/too much of their statistics could be visually obstructive and possible unfavourable. The player would then be able to view the entirety of their statistical performance and comparisons between their performances etc via the stats icon on the menu.

* Player can choose from a variety of different weapons  (FROM MY CRITEREA IN THE ANALYSIS STAGE PAGE 21-22)

During my coded solution stage when I was constructing the shooting mechanics, I intended to implement a variety of features apart from simply shooting such as fire rate, movement speed while holding certain weapons (STARTING FROM PAGE 62) etc. However, due to time constraints I focused on just implementing the single additional feature of firing mode (semi-automatic and fully automatic). Nevertheless, although this adjustable shooting mechanic functioning correctly with proof alongside it, I was unable to implement as a feature due to further time constraints. As a result, users can only use a single type of weapon.

HOW I WOULD GO ABOUT THIS MATTER IN FUTURE DEVELOPMENT:

In future development, I would proceed on constructing the same features I desired to construct in my design while maintaining correspondence with my interviewees for their valuable input such as weapon sprites of their preferences for additional different weapons I will include in future development

* There are a variety of targets of different natures that the player can choose from to shoot at.  (FROM MY CRITEREA IN THE ANALYSIS STAGE PAGE 21-22)

Due to time constraints, I was unable to implement a variety of different targets of different natures, so I decided to create a provisional solution. This was a target that had a hybrid nature of motion (AS SHOWN ON PAGE 72) which remained stationary rest and randomly teleported as well whenever shot at. Furthermore, due to time constraints, I wasn’t able to implement the feature of target heath and health deduction due to bullet damage. Therefore, due to the absence of health mechanics, I took a makeshift measure of just making the target react instantly once shot at (it repositioning whenever shot at)

HOW I WOULD GO ABOUT THIS MATTER IN FUTURE DEVELOPMENT:

I would proceed on constructing other types of targets with their unique natures as I intended to during my design while maintaining correspondence with my interviewees for their valuable input such as speed of targets and their courses of motion such as circular paths of motion or spiral etc.

However, although the current target in my application was supposedly a makeshift one, I believe that it will be ideal keeping it within the application as its teleporting behaviour creates the opportunity for players to practice their skills such as flicking (MENTIONED IN PAGE 6)

EVALUATION OF UNMET CRITERIA:

* Player can adjust certain features such as properties of their UI (FOV, mouse sensitivity etc) and features of targets (e.g. target speed, amount of targets)  (FROM MY CRITEREA IN THE ANALYSIS STAGE PAGE 21-22)

Due to time constraints I was incapable of implementing these features, another factors that contributed to this was that for a feature like this, it couldn’t be partially implemented neither. What is mean by this is that if I were to include these adjustment features on the controls page, they’d be futile without the implementation of the actual adjustment mechanic and ultimately being a significant loss in efficiency given the time frame I had.

HOW I WOULD GO ABOUT THIS MATTER IN FUTURE DEVELOPMENT:

In future development, I will work on implementing this feature, maintaining correspondence with my interviewees for their valuable input such as where I could arrange these calibration features on the controls page etc.

* Different maps/environment the player can choose from  (FROM MY CRITEREA IN THE ANALYSIS STAGE PAGE 21-22)

Although a common and desirable feature within FPS training applications, it is more of a supplementary feature rather than a crucial one for the fundamental mechanics of this application therefore I had in mind to implement this feature last. However, due to time constraints I was incapable of creating several environments for the user to choose from

HOW I WOULD GO ABOUT THIS MATTER IN FUTURE DEVELOPMENT:

In future development I will work on creating on at least one other different map that the user can choose to play with. The user can be provided the selection of these different maps through the main menu with addition to a thumbnail(s) of what the map looks like before they click on it. I will also be maintaining correspondence with my interviewees for their valuable input such as how I could display these additional icons onto the menu etc.

* For the player to be able to reload their weapon and aim with their weapon (which should increase the accuracy of their shot and reduce their movement speed) (FROM MY CRITEREA IN THE ANALYSIS STAGE PAGE 21-22)

Due to time constraints, I was unable to begin to construct the aiming mechanics and weapon reloading mechanics. In future development, I will proceed on the aiming mechanics as I designed during my design the stage. I will also be maintaining correspondence with my interviewees for their valuable input such as what degrees of magnification I could use for the aiming of different weapons etc

HOW I WOULD GO ABOUT THIS MATTER IN FUTURE DEVELOPMENT:

However, after careful consideration of my current application, I concluded that the need for a weapon chamber and weapon reloading system is unnecessary. I believe this because the concept of conserving bullets in the scenario of an FPS game is more of a tactical matter rather than a matter that concerns weapon skill. As an example, a player who is in a scenario that they’re very low on ammo, they’re likely to conserve the ammo for the case of self-defence rather than for aggression. This matter is more of a tactical/strategic one than a matter that concerns weapon skill. in conclusion, I believe that there would be no need to include and implement ammunition features in future development

Maintenance issues and how they can be dealt with:

In my final testing video, I tested all features within the span of just a few minutes, this means that I didn’t get to witness the behavior of the program after a long period of time (a form of boundary testing). This is important because users will not use this application for just the duration of my test video (a few minutes) but rather many minutes/hours. Due to this, I am not certain that the program will perfectly run after prolonged usage as errors can appear such as an overflow with variable holding the number of fires shot etc.

This can be dealt with by running the application and using it for a relatively long time (> 4hours). Throughout running it, I can note down any abnormalities (if any) that occur and pursue on finding what caused it/them and how they can be amended

Another maintenance issue about this application concerns how it runs on different devices with different operating systems. This is crucial for application development as it’s imperative that created applications run as they should on all operating systems, they’re designed to run on In my final test video, I ran the application on a windows computer. This application is a unity application therefore it can be run on other operating systems such as Linux etc. Due to this, I am not certain that this application will run just as it did on my Windows computer as it would on these other operating systems.

This can be dealt by running the application on each operating system either through on a separate computer that holds the alternative operating system or using a virtual machine on my current computer that’ll run on the alternative operating system(s). Then while running the application, I can note down any abnormalities (if any) that occur and pursue on finding what caused it/them and how they can be amended

LIMITATIONS AND POTENTIAL CHANGES:

I desired that the application can be run on computers with 4GB of RAM at least as well as other specs that tend to be are on par with this such as graphical processing specifications etc. Due to this, I didn’t prioritize the visually aesthetical aspect of the application because it comes at the cost of requires processing power so most of the objects within the application are that of simplistic 3d shapes like cylinders etc. Furthermore, the Unity template I used to create this application is purposely structured to not support the best of graphics, so I was incapable of making use of advanced graphing tools such as complex rendering etc. This can displease users. Moreover, my application lacks the implementation of varying graphics intensity and resolution, which can be considered as crucial features especially if the application is intended to be ran on a variety of computers (which tends to be favorable)

As a resolution, in future development I would focus on making use of these graphical calibration features. I may keep my current game scene (the environment) as an option for user’s with computers with relatively low processing power. On the other hand, I can construct a game scene that makes use of the advanced graphic features (e.g. resolution amplification) and implement more visually aspects such use more complex sprites in the environment rather than the basic shapes provided by the Unity software.

Another limitation is that after prolonged usage, collected data values will theoretically tend to a large number (such as shots fired). This will lead to the results of statistical values such as accuracy converging to a certain value. Although these converged value could be deemed to be accurate quantitative representations of the player’s performance, it can also be undesirable. This is because once there are large values within these statistical results they’re insensitive to small instantaneous changes. Below is an example of this limitation.

shots on target

accuracy = --------------------- × 100

shots fired

As an example if the player currently fired 10000 on target and fired 20000 in total

then their accuracy would be (10000/2000) × 100 = 50%

new accuracy = (1020/2020) × 100 = 50.4950495%

After 20 successive and successful shot on target

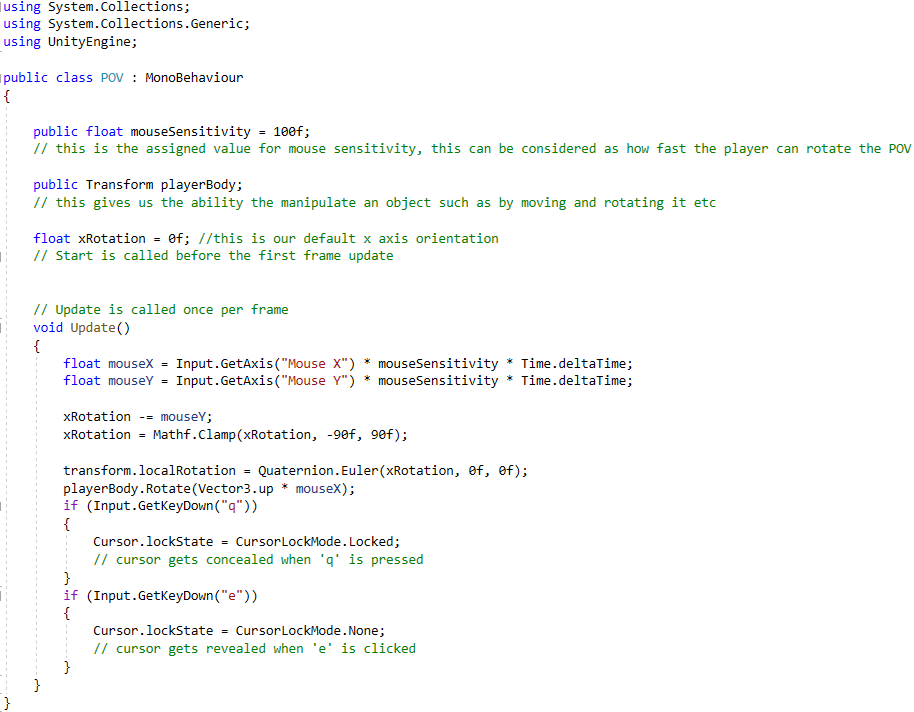
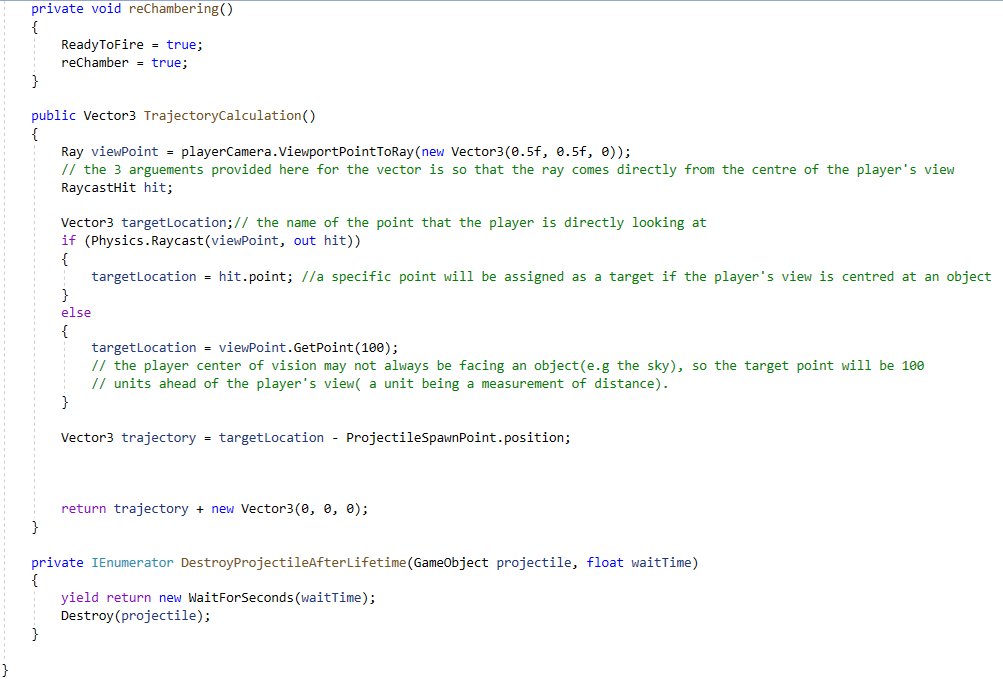
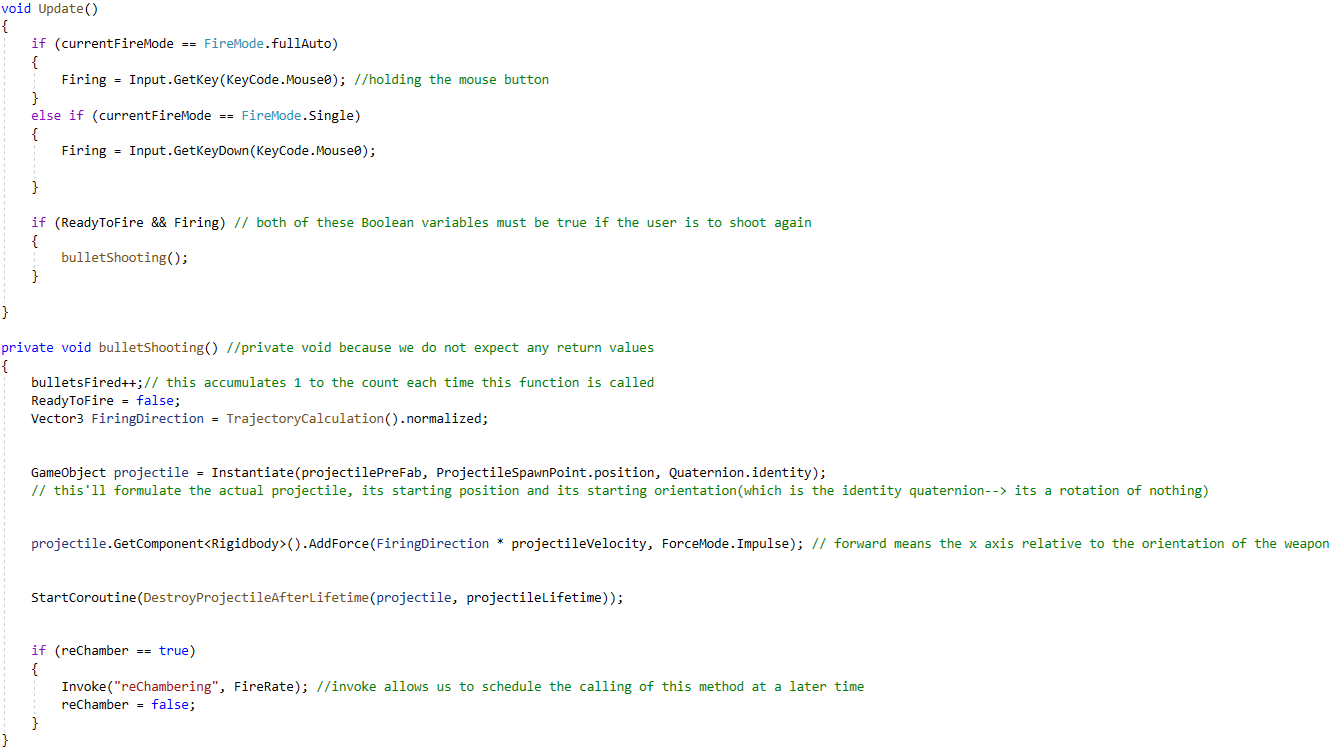
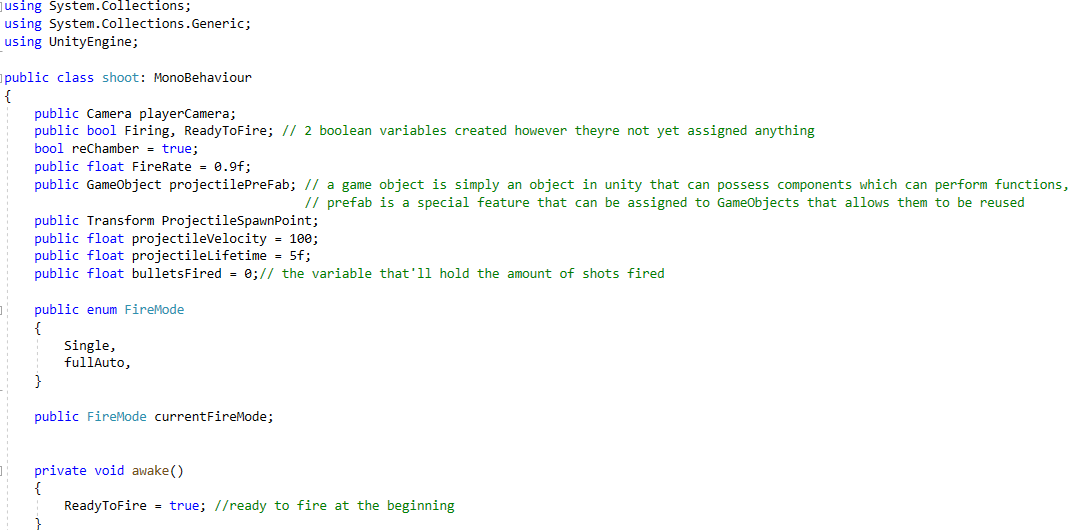
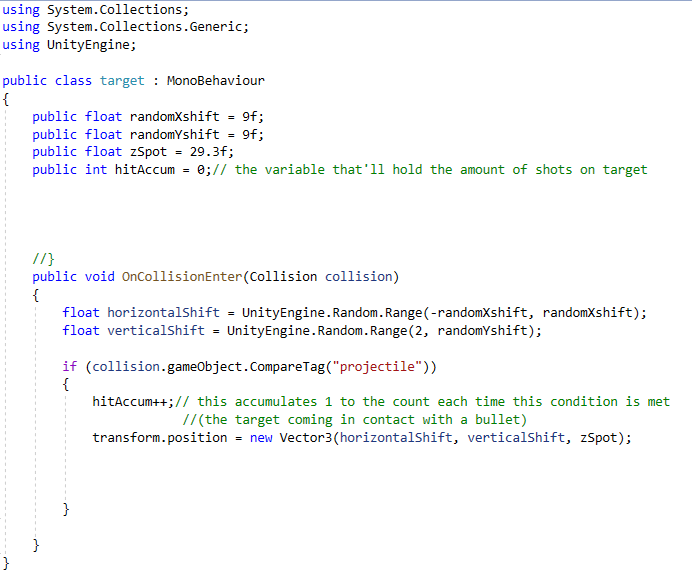
Above is a theoretical and sensible scenario that a user could be in after a prolonged duration of being on the application. The blue arrow represents the player landing 20 consecutive shots on target. Considering the time frame in which these 20 shots were fired, the user maintained an excellent accuracy of 100% (20/20 × 100 = 100%), however when accumulated to their total accuracy it presents am almost negligible difference (<0.5% increase). This wouldn’t accurately portray the user’s true quantitative accuracy in their latest performance which can be viewed as undesirable.

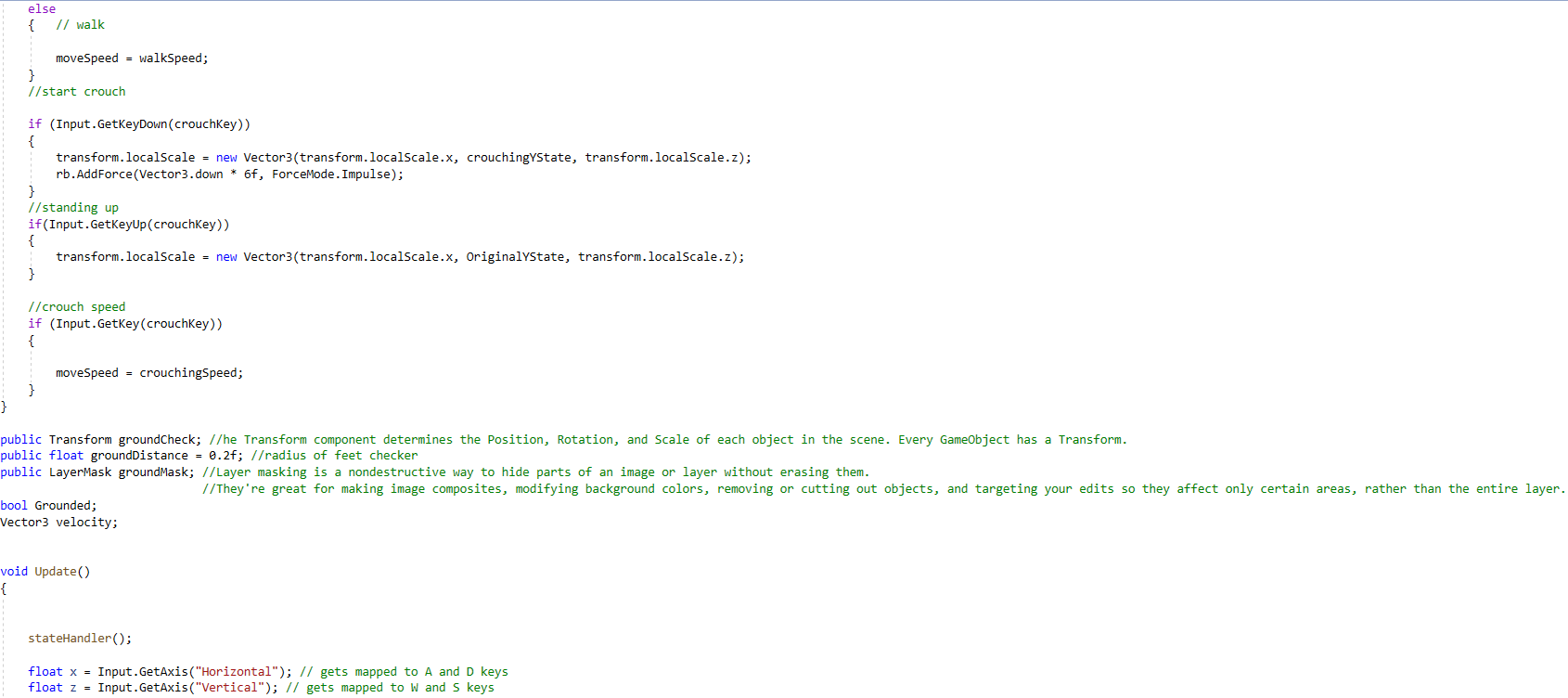
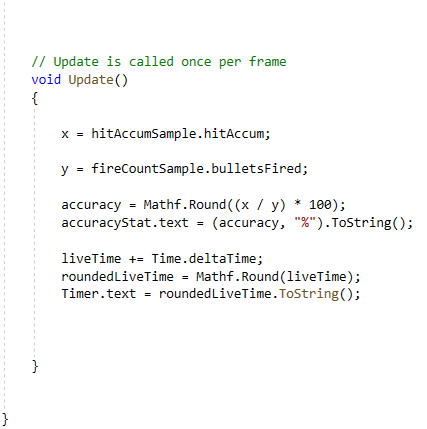
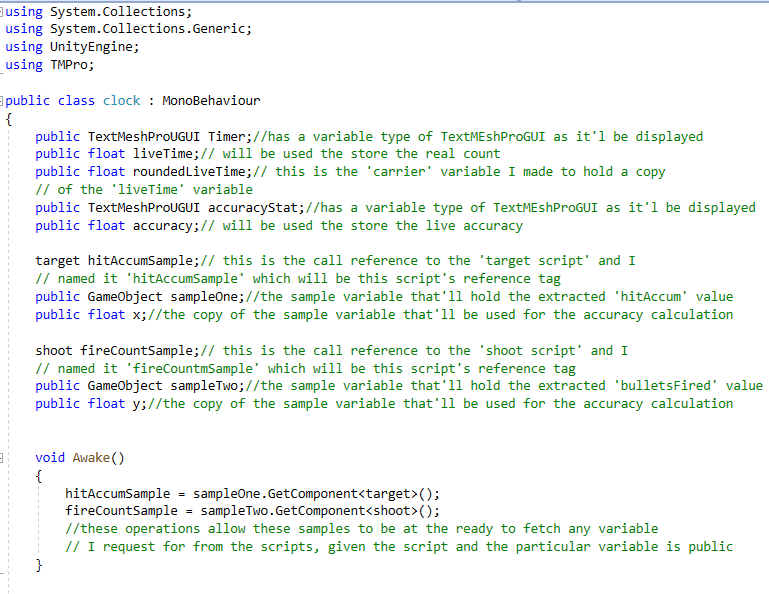
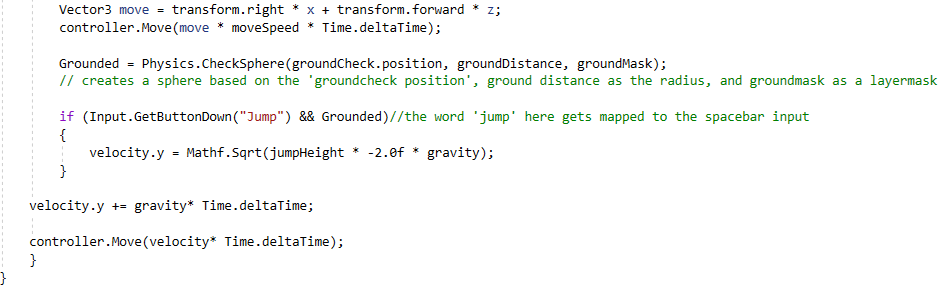
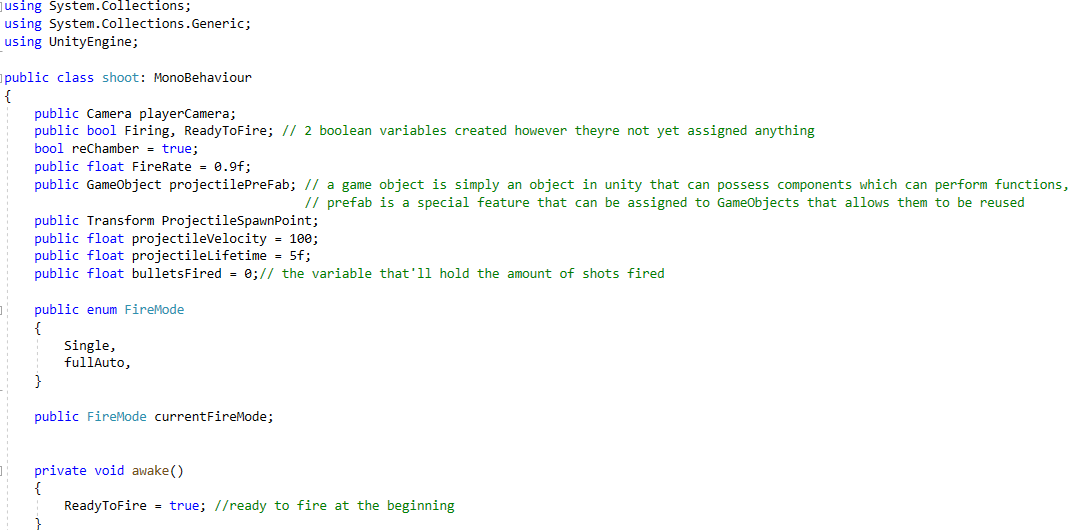
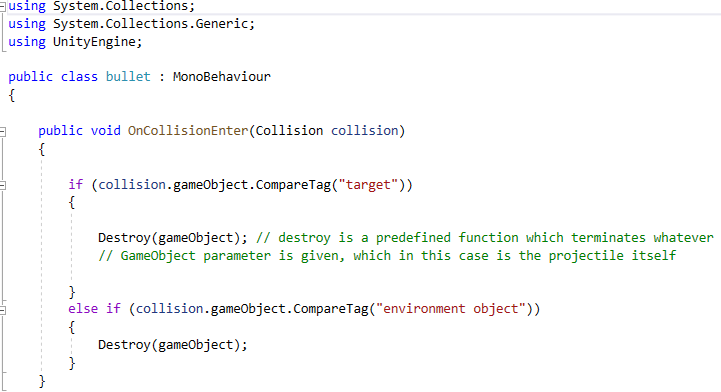
A potential amendment that I could make in the future with regards to this matter is that I can implement timed sessions e.g hour long training lessons. This would result the significance decrease of statistical values converging/tending to a fixed value and therefore provide players with a more accurate and ‘up to date' outlook on the user’s performance.

An alternative potential amendment is that I can instead present the user with their statistical performance values in a periodic manner. That is, I can make a separate copy of the gathered data (that are to be used in statistical calculations) that renews periodically such as every 25 minutes. From this, the user can be provided with their latest quantitative performance statistics.

Furthermore, a copy can be made of each of the resultant statistic values after each period and then presented before the user for comparison

Below are the screenshots of my entire code:

pti  
  


# Project Appendixes

Insert as many project appendixes as you need for your project.

These might include, but are not limited to:

* Interview notes/interviewee feedbacks:

P5-6, P15-16, P57, P63, P74-75, P77, P81-82, P83, P94

* THE INTERVIEW WITH MY INTERVIEWEES AFTER THEY CONDUCTED THE BETA TESTING:(SEE PAGE 94)

POST BETA TEST WITH MO:

Hey Mo, what are your thoughts about the menu:

I’m fine with the menu because it’s pretty easy to interact with and I can quickly hop on the game and hop off.

What do you think of the movement?

It was all fine until I decided to start shooting while crouching; I noticed that the gun sort of squashed and it was very noticeable as well, but I don’t think it effected the shooting though. Another thing I noticed is that I was able to sprint in all directions as well, which I’m not a fan of. It’s just not realistic to be able to sprint backwards and side to side because then if that was a thing in FPS games, then it’d be much harder to shoot players and just in general it’s not a realistic maneuver. On top of that, lots of players such as me move sideways to adjust their shots and tend to keep a finger hovering over the sprint button just in case. So, while I was shooting, I found myself accidently slightly to the side, completely throwing off my aim which I’m not happy about. I highly advise you to solve this problem in future development ASAP.

What are your thoughts on the HUD?

I’m alright with the HUD and I’m a fan of how the stats are in the corner so they don’t pose as a physical obstruction. I wouldn’t say I’m a fan of the color of these stats though being the bright color of white. This is because the scenery itself is full of bright colors especially the sky, so I sometimes find it hard to distinguish fully the stats on the HUD

What’s your opinion on the targets?

Although I’m aware of the time constraints, I think that the target is pretty good. This is because it’s like a hybrid between a stationary target and a moving target when it teleports. I also like the randomness of its movement as it really does test the user how fast they can switch targets.

Finally, what do you think of the shooting?

I’m satisfied with the shooting however I did notice that you implemented these bullet trails. I am a fan of these because they let me know where my bullets are going, which is pretty hard to tell as it does obviously travel really fast.

POST BETA TEST WITH JAY:

Hello Jay, I’d like to ask you about our views on the menu.

I’m happy with the simplicity of the menu and its ease of use, however there’s no way I can pause the game before returning to menu. This was a bit annoying as I was playing for a while and I wanted to quickly look at the controls, but when I returned to the game I started from the beginning with the timer and accuracy stat being restarted.

Thanks for the input there, now what do you think of the movement?

I found it smooth and consistent however when I crouched, I noticed my weapon deforming as well as if its crouching with me which of course is quite odd

Thanks for the input Jay, in fact, Mo also informed me of the same issue as well!

What is your opinion on the HUD?

I’m happy with the HUD but relating to the issue I raised in the previous question, I think it would have been more suitable if a user like me could simply access the controls page immediately from in game as well as from the menu just like in the game CS:GO. I believe this would make things more convenient and simpler.

What are your thoughts on the targets?

I like the target that has been implemented however I think it would be even better if the target or if there was a target that could fly rather than teleport so then a user like me could practice leading shots. I also think that it would be better if the teleporting target could teleport across the 3 dimensions

And finally, what do you have to say about the shooting?

I’m satisfied with the shooting however I did obviously notice the absence of crosshairs. I believe that this is a nice feature that should be kept as it really does let a player test how intuitively they can determine where the exact centre of their site is. This is critical especially in dire situation in game where the player gets distorted in game by the enemy such as from an emp grenade, which scrambles the HUD of opponents temporarily.

* LINKS OF EXTERNAL SOURCES USED THROUGHOUT THIS PROJECT (e.g diagrammatic software, pieces of code)

1. https://www.googleadservices.com/pagead/aclk?sa=L&ai=DChcSEwix2Na18deFAxUumlAGHTf3DYsYABAAGgJkZw&ae=2&gclid=EAIaIQobChMIsdjWtfHXhQMVLppQBh039w2LEAAYASAAEgJ4wPD\_BwE&ohost=www.google.com&cid=CAASJuRoeLzy6uJCyvyygv4uLJqBQsqU8\_izw3JtW-NVNLw7wa9lgSt8&sig=AOD64\_1HOob9Kchd-a3gskYSz6vHXJ4Bnw&q&adurl&ved=2ahUKEwiGzNG18deFAxVWUEEAHRXdBYAQ0Qx6BAgGEAE / contains the software used to construct all the flowcharts used
2. desmos.com / the software used for the diagram on P56
3. <https://www.google.com/imgres?q=2d%20man%20shooting%20gun&imgurl=https%3A%2F%2Fi.pinimg.com%2Foriginals%2Fa9%2Fa7%2Fdf%2Fa9a7df2bebe0772511c6052afaab5047.gif&imgrefurl=https%3A%2F%2Fin.pinterest.com%2Fpin%2Fquick-saves--794252084296897419%2F&docid=OBV70zjtKGxSRM&tbnid=TJcKKWVZISUDwM&vet=12ahUKEwiZ4_uZ8deFAxWjW0EAHQa7D2EQM3oECFEQAA..i&w=400&h=300&hcb=2&ved=2ahUKEwiZ4_uZ8deFAxWjW0EAHQa7D2EQM3oECFEQAA> / the source of the shooting man I used in the diagram P66
4. <https://www.google.com/url?sa=i&url=https%3A%2F%2Fjordansequillion.wordpress.com%2F2011%2F12%2F07%2Fscoring%2F&psig=AOvVaw2eIhU2CdppDtOXDzlSlkVP&ust=1713946342967000&source=images&cd=vfe&opi=89978449&ved=0CBAQjRxqFwoTCPjC7Onx14UDFQAAAAAdAAAAABAE> / the source of the target I used in the diagram P66
5. <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwihyuau8teFAxUHR0EAHX1pCUIQFnoECBYQAQ&url=https%3A%2F%2Fwww.physicsclassroom.com%2FPhysics-Interactives%2FVectors-and-Projectiles%2FVector-Addition%2FVector-Addition-Interactive&usg=AOvVaw1yk-0Kl46I61_A_KJGFKif&opi=89978449> / Software used for the vector simulation P67
6. <https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbmpMUE51LWFhaFRHRnJfbDFPMXpMVTVxZmJaUXxBQ3Jtc0trTk10YkpMYWQyZ1llcVJUc0xkNjRBeGUzcFNxNGVLMHVoblZDQWptSklLUzd4YU9wY0Rjdi1KSmd4enV0cTlmOGE3TnpxNEJUbWs2NHg4VVBRYzliaGxNT08yWlpuSFhrS0l4c3hmRFB0MFdxNEtXUQ&q=http%3A%2F%2Fdevassets.com%2Fassets%2Fmodern-weapons%2F&v=UK57qdq_lak> / Weapon sprites
7. <https://www.youtube.com/watch?v=TjwI3kqWQWw> / (mouse concealment and revealing code source for both concealing it(the issue I came across P57) and the issue of revealing it again (the issue I came across page P82))
8. <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=video&cd=&cad=rja&uact=8&ved=2ahUKEwj8w4KF5-mFAxW4UkEAHQOtB84QtwJ6BAgLEAI&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DuMtse2rN-zU&usg=AOvVaw343lLp5kGST0A4rtdPZX9w&opi=89978449> , [How to Fix Weapon Clipping in Unity (youtube.com)](https://www.youtube.com/watch?v=64HrAte6KD8) , https://www.youtube.com/watch?v=94xGeEavbW4&pp=ygURaW5oZXJpdGFuY2UgdW5pdHk%3D / sources I used in evaluation research for weapon clipping (P100)
9. [Inheritance - Unity Learn](https://learn.unity.com/tutorial/inheritance) , <https://www.youtube.com/watch?v=Y7pp2gzCzUI&pp=ygU5aG93IHRvIG1ha2Ugc2NyaXB0cyB1c2UgdmFyaWFibGVzIGZyb20gb3RoZXIgc2NyaXB0c3VuaXR5> / sources used to aid me with OOP inheritance in the project (P82)
10. <https://www.3daimtrainer.com/> , / source of 3d aimer photos
11. <https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSp1WcJcjrwdh9kIpuRitkpG4mCuDfSDWBoml6chvZY-w&s> , photo of aimlab
12. <https://www.catalog.update.microsoft.com/Search.aspx?q=Mouse+driver> , systematic hardware research
13. <https://docs.unity3d.com/Manual/system-requirements.html> , source for specs needed to run unity
14. <https://learn.microsoft.com/en-us/visualstudio/releases/2019/system-requirements> , source for Visual studio specs
15. <https://www.youtube.com/watch?v=C2bK0fkM0Bc> , projectile trail research
16. <https://www.youtube.com/watch?v=DX7HyN7oJjE&pp=ygUQdW5pdHkgc2NlbmUgbWVudQ%3D%3D> , source for unity software utility tools for scene mechanics
17. <https://www.studocu.com/en-gb/document/claremont-high-school/accounting-and-finance-in-business/documenting-defold-programming-projects-ocr-h446-games-projects/63433227> , exemplar template to aid me in structuring, for inspiration and advice
18. [Unity - Scripting API: MonoBehaviour.StartCoroutine (unity3d.com)](https://docs.unity3d.com/ScriptReference/MonoBehaviour.StartCoroutine.html) ,[Unity - Scripting API: Mathf (unity3d.com)](https://docs.unity3d.com/ScriptReference/Mathf.html), source for unity tools research
19. <https://www.3daimtrainer.com/> , source of aim trainer comparison research
20. <https://stackoverflow.com/questions/58911786/where-to-initialize-a-static-variable-in-unity> source of research on how to use features from one script in other one(s)
21. <https://www.youtube.com/watch?v=DX7HyN7oJjE&pp=ygUOdW5pdHkgbWVudSB0dXQ%3D> , <https://www.youtube.com/watch?v=pcyiub1hz20&pp=ygUXdW5pdHkgbWVudSBmcHMgdHV0b3JpYWw%3D> , sources used for research with menu construction
22. <https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/mechanics/kinematics/equations-of-motion.html> research for movement mechanics

* Complete Code Listing :

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class POV : MonoBehaviour

{

public float mouseSensitivity = 100f;

// this is the assigned value for mouse sensitivity, this can be considered as how fast the player can rotate the POV

public Transform playerBody;

// this gives us the ability the manipulate an object such as by moving and rotating it etc

float xRotation = 0f; //this is our default x axis orientation

// Start is called before the first frame update

// Update is called once per frame

void Update()

{

float mouseX = Input.GetAxis("Mouse X") \* mouseSensitivity \* Time.deltaTime;

float mouseY = Input.GetAxis("Mouse Y") \* mouseSensitivity \* Time.deltaTime;

xRotation -= mouseY;

xRotation = Mathf.Clamp(xRotation, -90f, 90f);

transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);

playerBody.Rotate(Vector3.up \* mouseX);

if (Input.GetKeyDown("q"))

{

Cursor.lockState = CursorLockMode.Locked;

// cursor gets concealed when 'q' is pressed

}

if (Input.GetKeyDown("e"))

{

Cursor.lockState = CursorLockMode.None;

// cursor gets revealed when 'e' is clicked

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class target : MonoBehaviour

{

public float randomXshift = 9f;

public float randomYshift = 9f;

public float zSpot = 29.3f;

public int hitAccum = 0;// the variable that'll hold the amount of shots on target

//}

public void OnCollisionEnter(Collision collision)

{

float horizontalShift = UnityEngine.Random.Range(-randomXshift, randomXshift);

float verticalShift = UnityEngine.Random.Range(2, randomYshift);

if (collision.gameObject.CompareTag("projectile"))

{

hitAccum++;// this accumulates 1 to the count each time this condition is met

//(the target coming in contact with a bullet)

transform.position = new Vector3(horizontalShift, verticalShift, zSpot);

}

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class shoot: MonoBehaviour

{

public Camera playerCamera;

public bool Firing, ReadyToFire; // 2 boolean variables created however theyre not yet assigned anything

bool reChamber = true;

public float FireRate = 0.9f;

public GameObject projectilePreFab; // a game object is simply an object in unity that can possess components which can perform functions,

// prefab is a special feature that can be assigned to GameObjects that allows them to be reused

public Transform ProjectileSpawnPoint;

public float projectileVelocity = 100;

public float projectileLifetime = 5f;

public float bulletsFired = 0;// the variable that'll hold the amount of shots fired

public enum FireMode

{

Single,

fullAuto,

}

public FireMode currentFireMode;

private void awake()

{

ReadyToFire = true; //ready to fire at the beginning

}

// Update is called once per frame

void Update()

{

if (currentFireMode == FireMode.fullAuto)

{

Firing = Input.GetKey(KeyCode.Mouse0); //holding the mouse button

}

else if (currentFireMode == FireMode.Single)

{

Firing = Input.GetKeyDown(KeyCode.Mouse0);

}

if (ReadyToFire && Firing) // both of these Boolean variables must be true if the user is to shoot again

{

bulletShooting();

}

}

private void bulletShooting() //private void because we do not expect any return values

{

bulletsFired++;// this accumulates 1 to the count each time this function is called

ReadyToFire = false;

Vector3 FiringDirection = TrajectoryCalculation().normalized;

GameObject projectile = Instantiate(projectilePreFab, ProjectileSpawnPoint.position, Quaternion.identity);

// this'll formulate the actual projectile, its starting position and its starting orientation(which is the identity quaternion--> its a rotation of nothing)

projectile.GetComponent<Rigidbody>().AddForce(FiringDirection \* projectileVelocity, ForceMode.Impulse); // forward means the x axis relative to the orientation of the weapon

StartCoroutine(DestroyProjectileAfterLifetime(projectile, projectileLifetime));

if (reChamber == true)

{

Invoke("reChambering", FireRate); //invoke allows us to schedule the calling of this method at a later time

reChamber = false;

}

}

private void reChambering()

{

ReadyToFire = true;

reChamber = true;

}

public Vector3 TrajectoryCalculation()

{

Ray viewPoint = playerCamera.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));

// the 3 arguements provided here for the vector is so that the ray comes directly from the centre of the player's view

RaycastHit hit;

Vector3 targetLocation;// the name of the point that the player is directly looking at

if (Physics.Raycast(viewPoint, out hit))

{

targetLocation = hit.point; //a specific point will be assigned as a target if the player's view is centred at an object

}

else

{

targetLocation = viewPoint.GetPoint(100);

// the player center of vision may not always be facing an object(e.g the sky), so the target point will be 100

// units ahead of the player's view( a unit being a measurement of distance).

}

Vector3 trajectory = targetLocation - ProjectileSpawnPoint.position;

return trajectory + new Vector3(0, 0, 0);

}

private IEnumerator DestroyProjectileAfterLifetime(GameObject projectile, float waitTime)

{

yield return new WaitForSeconds(waitTime);

Destroy(projectile);

}

}

MOVE

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class movement : MonoBehaviour

{

public CharacterController controller;

Rigidbody rb;// abbreviative purpose

public float gravity = -0.8f;

public float jumpHeight = 3f;

private float moveSpeed;

public float walkSpeed;

public float runSpeed;

public float crouchingSpeed;

public KeyCode runKey = KeyCode.LeftShift;

public KeyCode crouchKey = KeyCode.C;

public float crouchingYState;

private float OriginalYState;

private void Start()

{

OriginalYState = transform.localScale.y;

}

private void stateHandling()

{ // sprinting mode

if(isGrounded && Input.GetKey(runKey))

{

moveSpeed = runSpeed;

}

else

{ // walk

moveSpeed = walkSpeed;

}

//start crouch

if (Input.GetKeyDown(crouchKey))

{

transform.localScale = new Vector3(transform.localScale.x, crouchingYState, transform.localScale.z);

rb.AddForce(Vector3.down \* 6f, ForceMode.Impulse);

}

//standing up

if(Input.GetKeyUp(crouchKey))

{

transform.localScale = new Vector3(transform.localScale.x, OriginalYState, transform.localScale.z);

}

//crouch speed

if (Input.GetKey(crouchKey))

{

moveSpeed = crouchingSpeed;

}

}

public Transform groundCheck; //he Transform component determines the Position, Rotation, and Scale of each object in the scene. Every GameObject has a Transform.

public float groundDistance = 0.2f; //radius of feet checker

public LayerMask groundMask; //Layer masking is a nondestructive way to hide parts of an image or layer without erasing them.

//They're great for making image composites, modifying background colors, removing or cutting out objects, and targeting your edits so they affect only certain areas, rather than the entire layer.

Vector3 velocity;

bool isGrounded;

void Update()

{

stateHandling();

float x = Input.GetAxis("Horizontal");

float z = Input.GetAxis("Vertical");

Vector3 move = transform.right \* x + transform.forward \* z;

print(isGrounded);

controller.Move(move \* moveSpeed \* Time.deltaTime);

isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

// creates a sphere based on the 'groundcheck position', ground distance as the radius, and groundmask as a layermask

if (Input.GetButtonDown("Jump") && isGrounded)//the word 'jump' here gets mapped to the spacebar input

{

velocity.y = Mathf.Sqrt(jumpHeight \* -2.0f \* gravity);

}

velocity.y += gravity\* Time.deltaTime;

controller.Move(velocity\* Time.deltaTime);

}

}

HUD

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class clock : MonoBehaviour

{

public TextMeshProUGUI Timer;//has a variable type of TextMEshProGUI as it'l be displayed

public float liveTime;// will be used the store the real count

public float roundedLiveTime;// this is the 'carrier' variable I made to hold a copy

// of the 'liveTime' variable

public TextMeshProUGUI accuracyStat;//has a variable type of TextMEshProGUI as it'l be displayed

public float accuracy;// will be used the store the live accuracy

target hitAccumSample;// this is the call reference to the 'target script' and I

// named it 'hitAccumSample' which will be this script's reference tag

public GameObject sampleOne;//the sample variable that'll hold the extracted 'hitAccum' value

public float x;//the copy of the sample variable that'll be used for the accuracy calculation

shoot fireCountSample;// this is the call reference to the 'shoot script' and I

// named it 'fireCountmSample' which will be this script's reference tag

public GameObject sampleTwo;//the sample variable that'll hold the extracted 'bulletsFired' value

public float y;//the copy of the sample variable that'll be used for the accuracy calculation

void Awake()

{

hitAccumSample = sampleOne.GetComponent<target>();

fireCountSample = sampleTwo.GetComponent<shoot>();

//these operations allow these samples to be at the ready to fetch any variable

// I request for from the scripts, given the script and the particular variable is public

}

// Update is called once per frame

void Update()

{

x = hitAccumSample.hitAccum;

y = fireCountSample.bulletsFired;

accuracy = Mathf.Round((x / y) \* 100);

accuracyStat.text = (accuracy, "%").ToString();

liveTime += Time.deltaTime;

roundedLiveTime = Mathf.Round(liveTime);

Timer.text = roundedLiveTime.ToString();

}

}

BULLET

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class bullet : MonoBehaviour

{

public void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.CompareTag("target"))

{

Destroy(gameObject); // destroy is a predefined function which terminates whatever

// GameObject parameter is given, which in this case is the projectile itself

}

else if (collision.gameObject.CompareTag("environment object"))

{

Destroy(gameObject);

}

}

}