[1 Project overview 8](#_Toc131611145)

[2.0 Analysis 11](#_Toc131611146)

[2.1 Existing application research 11](#_Toc131611147)

[2.1.1 Application 1: Aim Lab 11](#_Toc131611148)

[2.1.2 Application 2: KovaaK's 13](#_Toc131611149)

[2.2 Features 14](#_Toc131611150)

[2.2.1 Feature 1: Data and Feedback 14](#_Toc131611151)

[2.2.1.1 Accuracy 14](#_Toc131611152)

[2.2.1.2 Error Size 15](#_Toc131611153)

[2.2.1.3 Reaction Time 16](#_Toc131611154)

[2.2.2 Feature 2: Graph making 16](#_Toc131611155)

[2.2.2.1 Line Graph 16](#_Toc131611156)

[2.2.2.1.1 Choosing which metrics to display on the graph 17](#_Toc131611157)

[2.2.2.1.2 Distance between each point on the graph 17](#_Toc131611158)

[2.2.2.1.3 2D example using unity 17](#_Toc131611159)

[2.2.2.2 spider graph 18](#_Toc131611160)

[2.2.2.2.1 creating the regular cyclic polygon 18](#_Toc131611161)

[2.2.2.2.2 creating the points within the cyclic polygon 18](#_Toc131611162)

[2.2.2.2.3 creating lines between the points 18](#_Toc131611163)

[2.2.3 Feature 3: Line Pathing 18](#_Toc131611164)

[2.2.3.1 Spawning the targets 18](#_Toc131611165)

[2.2.3.2 Rendering Lines between targets 18](#_Toc131611166)

[2.2.3.3 Getting statistics from the line 18](#_Toc131611167)

[2.2.3.4 Removing targets from the array when hitting targets 19](#_Toc131611168)

[2.2.4 Feature 4: Settings 19](#_Toc131611169)

[2.2.4.1 Controls 19](#_Toc131611170)

[2.2.4.2 Audio 19](#_Toc131611171)

[2.2.4.3 Crosshair 19](#_Toc131611172)

[2.2.5 Feature 5: Game sensitivity 19](#_Toc131611173)

[2.2.5.1 Conversion table 19](#_Toc131611174)

[2.2.5.2 Converting from my app to other games 19](#_Toc131611175)

[2.2.6 Feature 6: Target Spawning 19](#_Toc131611176)

[2.2.7 Feature 7: Player Tasks 20](#_Toc131611177)

[2.2.8.1 choosing what task to play 20](#_Toc131611178)

[2.2.8.2 what each task does 20](#_Toc131611179)

[RandShot 20](#_Toc131611180)

[Massclick 20](#_Toc131611181)

[Linetrace 20](#_Toc131611182)

[Gridshot 20](#_Toc131611183)

[2.2.8.3 layout of tasks 20](#_Toc131611184)

[2.2.8.4 results of each task 20](#_Toc131611185)

[2.2.8 Feature 8 Storing Player Data 20](#_Toc131611186)

[2.3 Stakeholders 20](#_Toc131611187)

[2.4 Requirements 21](#_Toc131611188)

[2.4.1 Hardware Requirements 21](#_Toc131611189)

[2.4.2 Software requirements 21](#_Toc131611190)

[2.5 Limitations 21](#_Toc131611191)

[2.5.1 Feature 1: Game physics and guns 21](#_Toc131611192)

[2.5.2 Feature 2: Game Maps 22](#_Toc131611193)

[2.5.2 Feature 3: AI controlled targets 22](#_Toc131611194)

[2.6 Success criteria 22](#_Toc131611195)

[3.0 Design 23](#_Toc131611196)

[3.1 Decomposition 23](#_Toc131611197)

[3.1.1 Data Storage 23](#_Toc131611198)

[3.1.2 View 24](#_Toc131611199)

[3.1.3 Controller 24](#_Toc131611200)

[3.2 Test Data 25](#_Toc131611201)

[4.0 prototype 1 25](#_Toc131611202)

[4.1 introduction 25](#_Toc131611203)

[4.1.1 Target Spawning 25](#_Toc131611204)

[4.1.2 Settings 25](#_Toc131611205)

[4.1.3 Data and feedback 26](#_Toc131611206)

[4.1.4 Line Pathing 26](#_Toc131611207)

[4.2 UI Design 26](#_Toc131611208)

[4.2.1 Game screen 26](#_Toc131611209)

[4.2.1 Settings 27](#_Toc131611210)

[4.3 Class diagram 27](#_Toc131611211)

[4.3.1 Classes 28](#_Toc131611212)

[4.3.1.1 TargetSpawner 28](#_Toc131611213)

[4.3.1.2 Target 28](#_Toc131611214)

[4.3.1.3 Shooting 28](#_Toc131611215)

[4.3.1.4 PlayerMovement 28](#_Toc131611216)

[4.3.1.5 MouseLook 28](#_Toc131611217)

[4.3.2 Methods 28](#_Toc131611218)

[4.4 Data Structure 29](#_Toc131611219)

[4.5 Algorithms Design 32](#_Toc131611220)

[4.5.2 Generating random coordinates for spawning targets 33](#_Toc131611221)

[4.5.3 Looping through the array of targets to render lines between them 33](#_Toc131611222)

[4.5.4 Taking health off the target when shot and destroying it when health reaches 0 33](#_Toc131611223)

[4.5.5 calculating error size and accuracy 34](#_Toc131611224)

[4.6 Implementation 35](#_Toc131611225)

[4.6.1 Mouse\_Look code 35](#_Toc131611226)

[4.6.2 TargetSpawner Code 39](#_Toc131611227)

[4.6.3 Shooting Code 41](#_Toc131611228)

[4.6.4 PlayerMovement Code 41](#_Toc131611229)

[4.6.5 Target Code 43](#_Toc131611230)

[4.7 Testing 44](#_Toc131611231)

[4.8 User Feedback 50](#_Toc131611232)

[4.9 Remedial Action 50](#_Toc131611233)

[4.10 Evaluation 50](#_Toc131611234)

[5.0 Prototype 2 51](#_Toc131611235)

[5.1 Introduction 51](#_Toc131611236)

[5.2 UI design 51](#_Toc131611237)

[5.2.1 Graph Screen 51](#_Toc131611238)

[5.2.2 Line path Screen 52](#_Toc131611239)

[5.2.3 Gridshot screen 52](#_Toc131611240)

[5.2.4 Results Screen 53](#_Toc131611241)

[5.2.5 Main Menu Screen 53](#_Toc131611242)

[5.2.6 Main Settings Screen 53](#_Toc131611243)

[5.2.7 Controls Menu Screen 54](#_Toc131611244)

[5.2.8 Crosshair Menu Screen 54](#_Toc131611245)

[5.3 Class Diagram 54](#_Toc131611246)

[5.3.1 Classes 54](#_Toc131611247)

[5.3.1.1 GridshotController 54](#_Toc131611248)

[5.3.1.2 LinePathController 54](#_Toc131611249)

[5.3.1.3 CameraController 54](#_Toc131611250)

[5.3.1.4 MovementController 54](#_Toc131611251)

[5.3.1.5 Graph 55](#_Toc131611252)

[5.3.1.6 UIController 55](#_Toc131611253)

[5.3.1.7 Target 55](#_Toc131611254)

[5.3.1.8 Options 55](#_Toc131611255)

[5.3.1.9 LineHover 55](#_Toc131611256)

[5.3.1.10 MainMenu 55](#_Toc131611257)

[5.3.1.11 DisplayGraphPointValue 55](#_Toc131611258)

[5.3.2 Methods 55](#_Toc131611259)

[5.3.2.1 GridshotController Methods 55](#_Toc131611260)

[5.3.2.1.1 start() 55](#_Toc131611261)

[5.3.2.1.2 update() 55](#_Toc131611262)

[5.3.2.1.3 GenerateAccuracyGraphPositions() 55](#_Toc131611263)

[5.3.2.1.4 EndTask() 56](#_Toc131611264)

[5.3.2.1.5 SpawnTargets() 56](#_Toc131611265)

[5.3.2.1.6 Shoot() 56](#_Toc131611266)

[5.3.2.1.7 get\_paused() : bool 56](#_Toc131611267)

[5.3.2.1.8 get\_pregame() : bool 56](#_Toc131611268)

[5.3.2.1.9 get\_taskEnded() : bool 56](#_Toc131611269)

[5.3.2.1.10 get\_timer() float 56](#_Toc131611270)

[5.3.2.1.11 get\_graphPoints() : vector2[] 56](#_Toc131611271)

[5.3.2.1.12 get\_taskTime() : float 56](#_Toc131611272)

[5.3.2.1.13 set\_gunVolume(float value) 56](#_Toc131611273)

[5.3.2.2 LinePath methods 56](#_Toc131611274)

[5.3.2.2.1 start() 56](#_Toc131611275)

[5.3.2.2.2 update() 56](#_Toc131611276)

[5.3.2.2.3 spawnTargets() 57](#_Toc131611277)

[5.3.2.2.4 generateLines() 57](#_Toc131611278)

[5.3.2.2.5 generateHoverTimeGraphPositions 57](#_Toc131611279)

[5.3.2.2.6 endTask() 57](#_Toc131611280)

[5.3.2.2.7 shoot() 57](#_Toc131611281)

[5.3.2.2.8 get\_paused() 57](#_Toc131611282)

[5.3.2.2.9 get\_pregame() 57](#_Toc131611283)

[5.3.2.2.10 get\_taskEnded() 57](#_Toc131611284)

[5.3.2.2.11 get\_graphPoints() 57](#_Toc131611285)

[5.3.2.2.12 get\_Timer() 57](#_Toc131611286)

[5.3.2.2.13 get\_errorSize() 57](#_Toc131611287)

[5.3.2.2.14 get\_reactionTime() 57](#_Toc131611288)

[5.3.2.2.15 get\_shotsTotal() 58](#_Toc131611289)

[5.3.2.2.16 get\_targetsHit() 58](#_Toc131611290)

[5.3.2.2.17 get\_avgPercent() 58](#_Toc131611291)

[5.3.2.2.18 get\_taskTime() 58](#_Toc131611292)

[5.3.2.3 LineHover methods 58](#_Toc131611293)

[5.3.2.3.1 start() 58](#_Toc131611294)

[5.3.2.3.2 update() 58](#_Toc131611295)

[5.3.2.3.3 onMouseEnter() 58](#_Toc131611296)

[5.3.2.3.4 onMouseExit() 58](#_Toc131611297)

[5.3.2.3.5 get\_totalTime() 58](#_Toc131611298)

[5.3.2.3.5 get\_hoverTime() 58](#_Toc131611299)

[5.3.2.4 CameraController methods 58](#_Toc131611300)

[5.3.2.4.1 start() 58](#_Toc131611301)

[5.3.2.4.2 update() 58](#_Toc131611302)

[5.3.2.4.3 set\_FOV(float value) 59](#_Toc131611303)

[5.3.2.4.4 set\_sens(float value) 59](#_Toc131611304)

[5.3.2.4.5 set\_ADS(float value) 59](#_Toc131611305)

[5.3.2.5 Graph methods 59](#_Toc131611306)

[5.3.2.5.1 update() 59](#_Toc131611307)

[5.3.2.5.2 generateXpositions() 59](#_Toc131611308)

[5.3.2.5.3 generateGraphPoints() 59](#_Toc131611309)

[5.3.2.5.4 generateGraphLines() 59](#_Toc131611310)

[5.3.2.6 Options methods 59](#_Toc131611311)

[5.3.2.6.1 start() 59](#_Toc131611312)

[5.3.2.6.2 update() 59](#_Toc131611313)

[5.3.2.6.3 goToMainFromCrosshair() 59](#_Toc131611314)

[5.3.2.6.4 goToMainFromControls() 59](#_Toc131611315)

[5.3.2.6.5 goToControlsFromMain 59](#_Toc131611316)

[5.3.2.6.6 goToCrosshairFromMain() 59](#_Toc131611317)

[5.3.2.6.7 get\_paused() 60](#_Toc131611318)

[5.3.2.6.8 get\_pregame() 60](#_Toc131611319)

[5.3.2.6.9 FOVInput(string value) 60](#_Toc131611320)

[5.3.2.6.10 SensInput(string value) 60](#_Toc131611321)

[5.3.2.6.11 ADSInput(string value) 60](#_Toc131611322)

[5.3.2.6.12 DotSizeInput(String value) 60](#_Toc131611323)

[5.3.2.6.13 CenterGapInput(string value) 60](#_Toc131611324)

[5.3.2.6.14 LineWidthInput(string value) 60](#_Toc131611325)

[5.3.2.6.15 LineHeightInput(string value) 60](#_Toc131611326)

[5.3.2.7 UIcontroller methods 60](#_Toc131611327)

[5.3.2.7.1 start() 60](#_Toc131611328)

[5.3.2.7.2 update() 60](#_Toc131611329)

[5.3.2.7.3 set\_dotSize(float value) 60](#_Toc131611330)

[5.3.2.7.4 set\_centerGap(float value) 60](#_Toc131611331)

[5.3.2.7.5 set\_lineWidth(float value) 60](#_Toc131611332)

[5.3.2.7.6 set\_lineHeight(float value) 60](#_Toc131611333)

[5.3.2.8 Target methods 61](#_Toc131611334)

[5.3.2.8.1 update() 61](#_Toc131611335)

[5.3.2.8.2 takedamage(float value) 61](#_Toc131611336)

[5.3.2.8.3 die() 61](#_Toc131611337)

[5.3.2.9 MovementController methods 61](#_Toc131611338)

[5.3.2.9.1 update() 61](#_Toc131611339)

[5.3.2.10 DisplayGraphPointValue methods 61](#_Toc131611340)

[5.3.2.10.1 update() 61](#_Toc131611341)

[5.3.2.10.2 awake() 61](#_Toc131611342)

[5.3.2.10.3 showValue(string value) 61](#_Toc131611343)

[5.3.2.10.4 hideValue() 61](#_Toc131611344)

[5.3.2.11 MainMenu methods 61](#_Toc131611345)

[5.3.2.11.1 update() 61](#_Toc131611346)

[5.3.2.11.2 goToGridshot() 61](#_Toc131611347)

[5.3.2.11.3 goToLinePath() 61](#_Toc131611348)

[5.4 Data Structure 61](#_Toc131611349)

[5.4.1 GridshotController key attributes 61](#_Toc131611350)

[5.4.2 LinePath key attributes 62](#_Toc131611351)

[5.4.3 LineHover attributes 63](#_Toc131611352)

[5.4.4 CameraController attributes 63](#_Toc131611353)

[5.4.5 Graph attributes 63](#_Toc131611354)

[5.4.6 Options attributes 63](#_Toc131611355)

[5.4.7 UIcontroller attributes 63](#_Toc131611356)

[5.4.8 Target attributes 64](#_Toc131611357)

[5.4.9 MovementController attributes 64](#_Toc131611358)

[5.4.10 DisplayGraphPointValue attributes 64](#_Toc131611359)

[5.5 Algorithm Design 64](#_Toc131611360)

[5.5.1 gridshot target spawning algorithm 64](#_Toc131611361)

[5.5.2 linePath target spawning algorithm 65](#_Toc131611362)

[5.5.3 linePath line spawning algorithm 65](#_Toc131611363)

[5.5.4 generating graph y positions 66](#_Toc131611364)

[5.5.5 generating graph x positions 66](#_Toc131611365)

[5.5.6 gridshot target shooting algorithm 66](#_Toc131611366)

[5.5.7 linepath target shooting algorithm 67](#_Toc131611367)

[5.5.8 generating graph points 67](#_Toc131611368)

[5.5.9 generating graph lines 68](#_Toc131611369)

[5.6 Implementation 69](#_Toc131611370)

[5.6.1 GridshotController Code 69](#_Toc131611371)

[5.6.2 LinePathController Code 76](#_Toc131611372)

[5.6.3 LineHover Code 82](#_Toc131611373)

[5.6.4 CameraController Code 84](#_Toc131611374)

[5.6.5 Graph Code 87](#_Toc131611375)

[5.6.6 Options Code 89](#_Toc131611376)

[5.6.7 UIcontroller Code 96](#_Toc131611377)

[5.6.8 Target Code 99](#_Toc131611378)

[5.6.9 MovementController Code 100](#_Toc131611379)

[5.6.10 DisplayGraphPointValue Code 102](#_Toc131611380)

[5.6.11 MainMenu Code 104](#_Toc131611381)

[5.7 Testing 106](#_Toc131611382)

[5.8 User feedback 109](#_Toc131611383)

[5.9 remedial action 109](#_Toc131611384)

[5.10 Evaluation 109](#_Toc131611385)

[6.0 Prototype 3 109](#_Toc131611386)

[6.1 introduction 109](#_Toc131611387)

[6.2 UI design 109](#_Toc131611388)

[6.2.1 Game Sensitivity Screen 109](#_Toc131611389)

[6.2.2 Feedback screen 109](#_Toc131611390)

[6.2.3 Updated Main Settings Screen 110](#_Toc131611391)

[6.2.3 Updated Results Screen 110](#_Toc131611392)

[6.2.4 Updated Controls Settings Screen 110](#_Toc131611393)

[6.3 Class diagram 111](#_Toc131611394)

[6.3.1 Classes 111](#_Toc131611395)

[6.3.2 Methods 111](#_Toc131611396)

[6.3.2.1 convertSensitivity() 111](#_Toc131611397)

[6.3.2.2 returnFeedback() 111](#_Toc131611398)

[6.3.2.3 saveControlsSettings() 111](#_Toc131611399)

[6.3.2.4 saveCrosshairSettings() 111](#_Toc131611400)

[6.4 Data structure 111](#_Toc131611401)

[6.5 Algorithm design 111](#_Toc131611402)

[6.5.1 Sensitivity conversion algorithm 112](#_Toc131611403)

[6.6 Implementation 112](#_Toc131611404)

[6.7 Testing 112](#_Toc131611405)

# 1.0 Project overview

The aim trainer will be developed using the unity game engine and the C# programming language. To produce the aim trainer, I must be able to use unity and be able to code and understand C#. In order to learn unity, I must use many resources at my disposal

Resources:

* + Videos
    - <https://www.youtube.com/watch?v=pwZpJzpE2lQ>
      * This video is 2 hours long and first teaches us how to use the unity editor and showing us around the GUI before explaining how to make and use basic scripts within the editor to apply code to certain game objects. The finished result is a basic 2D platformer with 3D graphics which includes respawning, coin collection and physics
  + Websites
    - [https://learn.unity.com](https://learn.unity.com/)
      * This is an official website made by unity which teaches people the basics of unity and how to work their way around the editor as well as basic 3D modelling withing the editor and basic scripts using C# as the language.

The unity game engine is a good fit for the aim trainer since it doesn’t require very high specs to run as well as using a programming language that is acceptable within the spec. However, the simplicity and ease of unity comes at a cost. While the game developer using unity must do most of the coding themselves, there are aspects of the game where the game engine does it for them such as physics simulations, 3D rendering and others. This means that for the coursework to get any marks, I must make my game to have multiple levels of complexity and I plan to achieve this in a couple of ways.

* + Method 1: multiple levels/tasks
    - In order to add a level of complexity to the game it must include multiple levels and tasks. This also helps the game be more successful in what it needs to achieve by having a selection of different scenarios for the player to choose from. The game will include multiple levels which are tailored to either different games or different skills that are needed for a player to improve their performance within a shooting game.
    - Examples of games:
      * Valorant
      * CS: GO
      * Apex Legends
      * Fortnite
      * PUBG
    - The games listed above are popular games that have large player bases and have huge competitive scenes. Games having big and popular competitive scenes are important because seeing pro player play at the top level of the game inspires average players to try to improve to hopefully reach that level one day
      * Note: For both apex legends, PUBG and Fortnite, due to their larger map compared to the other games there will be longer range battles in which the guns have bullet drop. This would mean that I would have to code the bullet drop that use physics respective to their games and guns.
    - Examples of improvable skills:
      * Flicking
      * Tracking
      * Speed
      * Precision
      * Reaction time
      * Audial detection
    - The skills listed above are important skills that a player needs to improve in order to get better at the games such as the ones listed above. Splitting the games into certain categories is important because it allows players from each game to improve in skills that they feel are weak in or improves skills that are needed more in one game than the other. For example, Apex Legends, which is a battle royal one a big map and a longer time to kills would require more tracking due to its faster movement and pace than a game such as Valorant, which would require more flicking and precision due to its precise gunplay elements which mean accuracy and aim makes or breaks a player.
  + Method 2: metrics and feedback
    - In order for an aim trainer to be good and achieve its purpose successfully, it must be able to store metrics and data of the player and give feedback to the player on how they can improve their aim.
  + These metrics may change based on what skill the task is made to focus on
  + Example 1:
    - After a task made for flicking:
      * Reaction time (time between target hits)
      * Accuracy (shots on target / total shots)
      * Precision (how close to centre of target)
      * Which side of the screen is their strength / weakness (up, down, left, right)
  + Example 2:
    - After a task made for tracking:
      * Time the crosshair has spent on the target
      * Accuracy (time spent on target / time spent overall)
  + Method 3: simple AI
    - While just using stationary targets may be a decent way to train ones aim, in order to fully maximise the accuracy and usefulness of an aim trainer, it must be able to simulate online players, or at least how the move and act. To do this, I would need to code an AI, it doesn’t have to be very complex but it can't be as simple as following a set path. The function of the AI would be to simulate how a player moves and acts such as fast left to right movements called "strafing".
      * "Strafing" overview:
        + Strafing is a movement technique that is very popular in almost every single shooter. Strafing consists of the player moving left and right in quick succession repeatedly in order to make it harder for the player aiming at them to hit them. To simulate strafing, it would need for a target either spherical or shaped like a player to randomly move left and right quickly to simulate what it would be like shooting at another player that is strafing. However, it is possible to add more depth and complexity to this AI and we can do this by introducing a reactive element to the task where the game will locate the position of the crosshair relative to the target and move the target accordingly. For example, if the crosshair is to the left of the target and moving towards the target, the target will move further to the left to avoid getting hit by the player. This introduces further difficulty in the task and the result will be the player improving their ability to react to strafing movements.
    - Another use of AI within the aim trainer would be reactive targets that change size and/or shape depending on the performance of the player within the task. I will call this "Reactive training".
      * "Reactive training" overview:
        + Reactive training is a useful tool to make sure that the players improvement never hits a plateau due to the game not introducing any more difficulty or change to the task. This is done by changing the size and/or shape of the targets based on player performance. For example, if the player is consistently hitting shots with high accuracy, the targets will get smaller and smaller and stop at a minimum size, however if the player is not hitting shots and has a low accuracy, the targets will increase in size until they hit a max size. This dynamic target size would keep the user engaged in the task as the changing size of the targets makes sure they are focusing on their accuracy to ensure that they don’t miss the targets when the size changes. This increases the skill ceiling of the task and allows for much further improvement to the players aim after they have become confident with the normal training.

# 2.0 Analysis

## 2.1 Existing application research

### 2.1.1 Application 1: Aim Lab

When thinking of aim trainers, Aim Lab is the very first that comes to mind due to its popularity amongst players of all games. Aim Lab came out in February 2018 and has amassed over 25 million players to date, making it one of the most popular aim trainers of all time. Its popularity combined with its effectiveness as an aim trainer makes Aim Lab one of the best existing apps I could use as research to develop my own.

Not only has Aim Lab improved exponentially over the years to improve effectiveness as an aim trainer, they have also worked hand in hand with the developers of the popular shooters such as Valorant and Rainbow Six Siege to develop tasks that are official and tailored to that game specifically to ensure that the player of that game have tasks that are specialised to help them improve in their respective game

Aim Lab link: <https://store.steampowered.com/app/714010/Aim_Lab/>

|  |  |
| --- | --- |
| Feature | Why it would be suitable for my app |
| Data and Feedback | Every task has its own unique analytics which are focused on a particular area that is specific to that task. This combined with the data visualisations in the form of line and spider graphs allows you to understand your performance and adjust and improvements accordingly.  -----------------------------------------------------------------  [1]: This figure points to the area of the results page where the player stats are displayed such as accuracy, score, reaction time are all calculated by the game based on the raw data that is gathered during the time the player is doing the task.  [2]: This figure points towards the spider diagram on the results page where they game has taken multiple player stats and put it on a spider diagram to visualize the players performance relative to other statistics. This is effective because it visualizes the players strengths and weaknesses for example the player can see how good their accuracy is compared to their speed.  [3]; This figure points towards the area where the players performance is compared to their average over the last 100 plays of that task.  -----------------------------------------------------------------  [1]: This figure points towards the 3 statistics represented as line graphs where the y axis is time(s) and the x axis is the name of the graph. This visualizes the statistic and how it changes as time goes on within the task to help the player see their consistency withing the task.  [2]: this figure points towards a graph that splits the screen into 8 sections and represents the performance in that area using green, amber or red. Green means that the performance in that sector is good, amber means that the performance is average and red means the performance is bad  -----------------------------------------------------------------  This screen shows the strengths and weaknesses of the player to ensure the player knows what to improve on as well as tips to improve that weakness and turn it into a strength. |
| Custom crosshairs / Crosshair editor | Crosshairs, just like sensitivity is completely dependent on the player. A crosshair that one player likes may not be one that another player likes. So, in order for the player to be able to train efficiently, Aim Lab has a system where the player can either design their crosshair using the in-game crosshair customizer that has sliders for different features such as length, width, centre dot as well as a feature that allows for the import of an image to use as a crosshair in case the crosshair design is not able to be made by the crosshair maker. |
| Changeable FOV scale | The field of view (FOV) is the extent of the observable world that is seen at any given moment. It is one of the most key settings in any game that is crucial to being a better player since a player with a higher FOV can see more of the world than a player with low FOV. This feature is important because it is crucial that the player’s game within the aim trainer has identical FOV as in game to be able to train efficiently especially since a change in FOV also means a change in sensitivity. |

### 2.1.2 Application 2: KovaaK's

KovaaK’s aim trainer was released in April 2018 and is also one of the most popular aim trainers alongside Aim Lab. KovaaK’s is made on unreal engine 4 which is a game engine made and developed by epic games in 2014.

Kovaak’s is a paid aim trainer so is not as popular as free ones such as Aim Lab, however it contains many more features than the ones that are free to play and download such as in-depth map customization and identical weapon physics and movement.  
Kovaak’s Link: <https://store.steampowered.com/app/824270/KovaaKs/>

|  |  |
| --- | --- |
| Feature | Why it would be suitable for my app |
| Practice scenarios | Kovaks contains multiple practice scenarios which are tailored to help certain skills and make the training feel more like the real game. This allows the player to have more efficient training since the similarity of the practice scenario to the game that they are training for allows them to improve drastically |
| Custom Crosshair | Explained in Application 1 |
| Changeable FOV scale | Explained in Application 1 |

## 2.2 Features

### 2.2.1 Feature 1: Data and Feedback

The data and feedback feature can be implemented using calculation algorithms that take data input from the player metrics and output the result of the calculations as data and gives feedback based on that.

Examples of metrics that will be used to calculate aim score:

* Accuracy
* Error Size
* Reaction Time

#### 2.2.1.1 Accuracy

Accuracy is how many shots are on target relative to total shots fired. Accuracy is an important metric for aim score because aim score is deducted when you miss shots and have low accuracy.

= *Accuracy (%)*

= *Number of targets hit*

*= Total shots taken*

Example code of how accuracy would be calculated:



*targetsHit* is a variable that is incremented every time a target is destroyed

*shotsTotal* is a variable that is taken every time a shot is fired

#### 2.2.1.2 Error Size

Error size is the distance from the exact center of the target. A higher error value would mean that the player is hitting the target far away from the center of the target and a lower error value would mean that the player is hitting closer to the center of the target. A lower error value would contribute to a higher aim score.

= *Error Size*

*= Total Error Size*

*= Total targets destroyed*

Example code of how error size would be calculated:

//Calculates errorsize by getting the x and y distance from the shot position to the center for the target then adds it to  
//the errorsize variable then divides it by the number of targets hit to get the average errorsize  
shotPosition = ShotHit.point;  
errorSize += shotPosition.x - ShotHit.transform.position.x;  
errorSize += shotPosition.y - ShotHit.transform.position.y;

This line of code takes the position of the shot on the target then takes the distance from the shot to the center of the target then adds it to the variable errorsize.

This line of code then takes the errorsize variable from the script that calculates the errorsize of the targets then divides it by the number of targets hit to give the average error size for each target hit.

Example:

A total error size of 4.56 over 4 targets would give an average error size of 1.14

This is calculated by substituting the values into the error size equation which will look like

#### 2.2.1.3 Reaction Time

Reaction time is the average time it takes for the player to hit the target. This would be measured by starting a timer from when the target spawns in all the way to when the player hits, adding all the times and dividing the number by the number of targets that have been hit.

*= Reaction time (ms)*

*=Total time targets have spent spawned in (ms)*

*= Total targets destroyed*

Example code of how Reaction Time would be calculated:



This line of code uses the variable t to store the time passed since the start of the task



This line of code then divides the time since the start of the task by the number of targets hit which will gives the average time that it takes to hit each target.

Example:

10.34 seconds have passed since the start of the task and the player has hit 45 targets which gives an average of 0.229 seconds per target.

#### 2.2.1.4 Line Hover Time

#### 2.2.1.5 Calculating Overall Score

One way to track the user’s overall performance in all of the fields would be to have an overall score which is calculated using all of the data that is collected over the entirety of the task. This includes all of the above statistics.

To get an overall score, we must first normalise all the values between 0 and 1.

To do this we use the equation:

After all the values are normalised, we can use the equation:

= The value you want to normalise

= The minimum your value can be

= The maximum your value can be

The value is multiplied by 25,000 in order to get a number between 0 and 100,000 where 100,000 is a complete perfect score and 0 is the worst possible score.

### 2.2.2 Feature 2: Graph making

Graph making is a feature that involves taking raw data values that are taken from the players performance such as time and accuracy and uses those data values to plot graphs of the players performance.

This feature is needed to give the player a visual representation of their performance as well as the raw data. This visual representation helps the player to truly understand their results and allow them to work on parts of their gameplay which they are weakest in.

I found that this feature is very important to implement because of the existing app research I did where I found that the data that was given to me in the form of a graph allowed me to better understand my results as compared to the raw data values that were given after the task. One such example is where after the task a graph was displayed that plotted my accuracy against time and I found that as there was less and less time remaining in the task, my accuracy had started to decrease, which allowed me to understand that my accuracy is worse under pressure than it is when not under pressure which helps me to improve my overall score by practicing under situations that put pressure on me.

#### 2.2.2.1 Line Graph

One of the best ways to display the player’s metrics in a fashion that they understand would be to put the values into a line graph. This allows the player to see the relation between 2 values such as accuracy and time

#### 2.2.2.1.1 Choosing which metrics to display on the graph

When making a graph, the very first thing to consider is what values you are plotting against each other to generate the graph.

There are many different metrics collected from the task that we can plot on the graph such as:

* Accuracy
* Reaction Time
* Error Size

However, some tasks focus on different gameplay aspects and metrics more than others for example: 3x3 gridshot would focus a lot more on reaction time and flicking than line trace. This means that the values that we plot on the graph can differ between tasks.

#### 2.2.2.1.2 Distance between each point on the graph

Since we are generating many points to display on the graph based on how many seconds have passed, we can’t just make the size of the graph based off the number of points/how many seconds the task is, since each task might have a different length in terms of time. Therefore, a better solution would be to adjust the distance between each point in the graph in order for it to fit into the bounds of the graph. We can do this by dividing the length that we want the entire graph to be by the number of points.

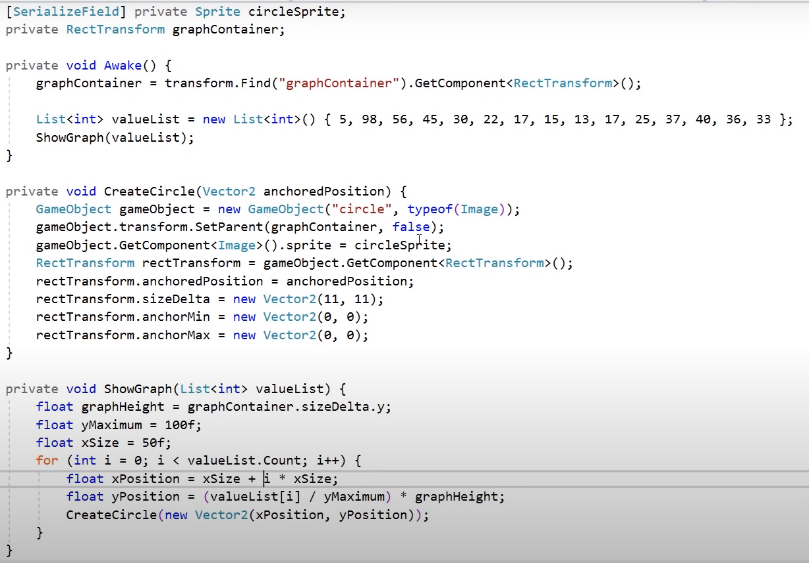
Where:

= Distance between each point in the graph

= Total length of the graph

= Number of points in the graph

#### 2.2.2.1.3 2D example using unity



Above is example code taken from a tutorial made on YouTube [1] on how to make a graph in unity.



Above is what the code for the graph outputs

[1] = <https://www.youtube.com/watch?v=CmU5-v-v1Qo>

#### 2.2.2.2 spider graph

##### 2.2.2.2.1 creating the regular cyclic polygon

##### 2.2.2.2.2 creating the points within the cyclic polygon

##### 2.2.2.2.3 creating lines between the points

### 2.2.3 Feature 3: Line Pathing

Line pathing feature includes generating lines between the last target that had just been destroyed by the player and the new target that has just spawned in. This line is there to guide the player to the next target in a straight line because straight lines are the shortest path to the next target. This will help the player destroy targets much quicker since they are taking the shortest path there which reduces the time between destroying each target.

#### 2.2.3.1 Spawning the targets

At any one time in this task there will be one target spawned in at a random position within the bounds of the spawner. To spawn the targets, we must:

* Make sure there are no more than one targets spawned in
* Generate a random position to spawn the target

#### 2.2.3.2 Rendering Lines between targets

To render the lines between the targets, we must first get the position of the currently spawned in target and when the target is destroyed, we then spawn a cube in between the position of the target that has just been destroyed and the target that has just spawned. We then adjust the length of the cube to be equal to the distance between the 2 targets which creates a line between 2 targets.

#### 2.2.3.3 Getting statistics from the line

The statistic that I will be getting from the line is how long the player spends with their cursor hovering over the line compared to how long the line has been spawned in for. This will give the player an idea of how straight their cursor moves from target to target which will help them see if they need to improve their cursor movement between target to target because if they are spending less time on the line, it means that they are not moving their cursor straight which means they are wasting time between targets which can greatly reduce the total targets that they destroy overall in a task.

#### 2.2.3.4 Removing targets from the array when hitting targets

### 2.2.4 Feature 4: Settings

#### 2.2.4.1 Controls

#### 2.2.4.2 Audio

#### 2.2.4.3 Crosshair

### 2.2.5 Feature 5: Game sensitivity

This feature involves being able to change the sensitivity in game to allow the player to play just how they would in their normal game as well as match sensitivity settings to their respective games.

This feature is important to implement because it allows the player to control more their aim training experience to match it as close to their actual game as possible. This will include an option to enter the game that they play and what sensitivity they play on so that they can play on that exact sensitivity on the aim trainer to make their aim training more effective.

This will be done by calculating a ratio between the unity game engine base sensitivity of the camera and the game on its default sensitivity. Once a multiplier of the game engine to the game’s sensitivity has been found, we can use that to change the sensitivity of the game engine to match the game.

The algorithm will take the user input on what game they’d like the sensitivity to be from then use the multiplier of that game to then match the sensitivity.

#### 2.2.5.1 Conversion table

https://aiming.pro/mouse-sensitivity-calculator#/

This conversion table was a test to confirm that it was possible to accurately convert between different game sensitivities using one game as a standard. This table was made by first converting from Aiming.pro on sensitivity **1.0** and then converting that to other games.

This allowed me to make a connection between sensitivity values of all games using Aiming.pro as a medium. For example, I converted from Aiming.pro sensitivity **1.0** to both valorant and apex. This gave me values of **2.604354** and **0.818511** respectively. We then divided the larger number by the smaller number (**2.604354 / 0.818511**) which gave us a value of **3.18181918**. We then use this value to then try to convert from one value to another, using the Aiming.pro conversion website to confirm the conversion is accurate. To test conversion, we multiply **2.604354** by 2 to get **5.208707** then divide that value by the conversion value (**3.18181918**) to get **1.637022001** which we can compare to the value given by the Aiming.pro conversion website (<https://aiming.pro/mouse-sensitivity-calculator#/>) Of **1.637022**.

This gives an accuracy within ± 0.001 of the converted value.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Aiming.pro | Apex Legends | Valorant | CS: GO | Overwatch 2 | Rainbow 6 |
| 1.0 | 2.604354 | 0.818511 |  |  |  |
| 2.0 | 5.208707 | 1.637021 |  |  |  |
| 5.0 | 13.021768 |  |  |  |  |
| 10.0 | 26.04354 |  |  |  |  |

#### 2.2.5.2 Converting from my app to other games

To convert from my app to other games, I will be using a similar table as the 2.2.5.1 conversion table

### 2.2.6 Feature 6: Target Spawning

This feature involves the spawning of targets that the player will be shooting. Targets are used to measure the ability of the player based on several categories. Target sizes can vary depending on the task and the skill that the player is trying to improve for example, a task to practice flicking will include bigger targets than a task for accuracy. This is because different target sizes can help train different skills for example smaller targets help train accuracy and precision because it forces the player to concentrate more on where they are placing their crosshair rather than how fast they can place their crosshair there.

Targets will be spawned considering different conditions:

* How many targets are already on screen
* Max number of targets that can be on the screen at any one time
* Spawn position that the target will spawn in
* Size of the target

### 2.2.7 Feature 7: Player Tasks

This feature involves the player task system and how the application calculates the player’s score based on their performance and how those results are used to give useful feedback to the player in order to improve their skills. Player tasks are what the entire application is based off and are the feature that will feed into all the other features

#### 2.2.8.1 choosing what task to play

#### 2.2.8.2 what each task does

##### RandShot

##### Massclick

##### Linetrace

##### Gridshot



#### 2.2.8.3 layout of tasks

#### 2.2.8.4 results of each task

### 2.2.8 Feature 8 Storing Player Data

## 2.3 Stakeholders

My target stakeholders are competitive online gamers, specifically those who play shooter games. Competitive online gamers refer to those who play games online against other players and the specific types of online gamers are those who play shooter games. My app does apply to a lot of competitive gamers since in this article about the best competitive games [1], 12 out of the 20 games on the list are shooter games.

These players would be specifically looking to gain a competitive edge against the other online players by using my aim trainer to improve aspects of their game which will allow them to perform better. My app allows the players to see their strengths and weaknesses when it comes to their aim, this may allow a player to make an informed decision on how they approach their gameplay for example, a player who is consistently able to hit slow moving or stationary smaller targets with a high accuracy but is unable to hit fast moving larger targets with the same accuracy and consistency might want to play a sniper class due to their ability to hit smaller targets with high accuracy.

As well as helping the player to make informed decisions about how they approach their game, my app will also allow the player to practice the areas of gameplay where they are weaker for example, if a player struggles to switch between shooting one target and another, they can use the line pathing feature in order to improve the way that they switch between targets which will overall help them improve in that area.

## 2.4 Requirements

|  |  |  |
| --- | --- | --- |
| User Story | Priority | Story Score |
| Data and feedback | High | 40 |
| Graph making | High | 80 |
| Line Pathing | Low | 60 |
| FOV slider | Low | 10 |
| Game sensitivity | High | 70 |
| Player Movement | High | 30 |

### 2.4.1 Hardware Requirements

CPU: Must have X64 architecture with SSE2 and instruction set support

GPU: Must be DX10, DX11 and DX12 compatible

### 2.4.2 Software requirements

Windows 7(SP1 +) and windows 10 (64 bit) versions only

The hardware and software requirements are taken directly from the unity documentation website [1] and are the minimum requirements to run unity.

[1] = <https://docs.unity3d.com/2020.1/Documentation/Manual/system-requirements.html>

## 2.5 Limitations

### 2.5.1 Feature 1: Game physics and guns

This feature includes implementing the exact same physics and guns that a game would include which will allow the player to feel more like they are practicing that game rather than a general aim practice.

I did not include this feature since there are far too many popular shooter games for me to implement as well as it being difficult to simulate the physics since I am making my app on unity and not every game would have been made on the unity game engine which would make their physics extremely hard to implement.

### 2.5.2 Feature 2: Game Maps

This feature would have included implementing the exact map models from certain games to make the player feel like they are playing the game that they are practicing for.

I did not implement this feature since the map models would be hard to replicate and implement into my app using the physics from my app as well as the models making the app size larger which may make the game less accessible to players who have limited disk space.

### 2.5.2 Feature 3: AI controlled targets

This feature would have included targets that would be controlled by AI and would move based on the players crosshair. For example, the targets would move relative to the position of the player crosshair in order to make it much harder for the player to hit the target which can help them become better at aiming at quick moving targets.

I did not implement this because coding an AI that would move based on the players crosshair would be very demanding and might not work with the rest of the app.

# 2.6 Success criteria

|  |  |  |
| --- | --- | --- |
| Success criteria | How it will be measured | justification |
| The data and feedback will result in player improvement | Seeing a rise in the players aim score over time | The data and feedback that the app will give will allow the player to see their own strengths and weaknesses which will allow them to work on different areas in order to improve |
| The graphs will help the player visualize their results | The player will be able to know exactly where their weaknesses are. | The graphs will help the player to see exactly where their weaknesses are by showing them exactly which aspect of their aim, they are weak in. For example, a graph of accuracy by time can show whether they are weak under time pressure by seeing if their score goes down when time is low |
| The FOV slider will help the player to train effectively | The player will be able to match their FOV in the aim trainer to the FOV in game | This helps because it allows the player to train closer to how their actual gameplay will feel in game which will make the aim training more effective since the FOV will be exactly as they play in game |
| Being able to change sensitivity will help the player train effectively | The player will be able to change their sensitivity to what they feel more comfortable | Sensitivity plays a really big role when it comes to how a player plays in a first person shooter game. This means that a lot of player spend time perfecting what sensitivity to play on so the ability to change the sensitivity will help massively with helping the player train better |
| The gridshot task will allow the player to improve their flicking | Seeing an improvement in reaction time | Gridshot is a task where the targets spawn in a 3x3 grid meaning that the targets spawn closer together than the other tasks. This means that the player is able to make very small adjustments called “flicks” onto the targets that spawn in quick succession. An improvement in this skill can be seen where there is an improvement in reaction time where a quicker reaction time means that they are shooting more targets meaning they are doing more flicks. |
| Being able to change the crosshair will allow the player to train effectively | The player will be able to change their crosshair to match their ingame one | Being able to match the crosshair in the aim trainer to the one in their game will massively help them improve in their game as it reduces the differences between the aim trainer and the game which allows the player to improve better. |
| being able to store settings will improve the overall user experience | The player will be able to access their settings even after closing out of the trainer and going back onto it | This will improve the overlal user experience because it means that the user will not have to enter their settings every time they open the trainer and instead its already stored there so they don’t have to enter it again |
| The linetrace class will allow the player to improve their overall score | The player will be able to path between the targets more effectively | This will improve the overall score because it will help them to path between the targets more effectively which will reduce the time between each target being destroyed and increase wih the number of targets destroyed in the task since the path between the targets is in a straight line it’s the shortest path which means the player can destroy more targets in the same amount of time. |
| The ability to change ADS multiplier will help the player to train more effectively | The player will be able to change their ADS multipleier to match the in game one | This will help them train more effectively because it creates another similarity between the game and the trainer which means that when they are training it feel more like the acrual game and the trainer which will alow them to train better than if it felt more different from the game. |
|  |  |  |
|  |  |  |
|  |  |  |

# 3.0 Design

## 3.1 Decomposition

I have broken the application down into 3 categories: Data Storage, View and Controller. This is because those 3 sections all cover the core aspects of the application which includes how it stores the data (Data Storage), how the application interacts with the player (View) and how the application uses and applies logic (Controller).

### 3.1.1 Data Storage

Total Score Storing which is where the application stores the total score of the player so that the player can see their improvement from past training sessions.

Storing game stats includes the application storing the statistics of the player such as time spent training, training tasks complete and overall accuracy.

### 3.1.2 View

User input includes features where the player directly interacts with the application using inputs with both the keyboard and mouse to control the in-game character.

* Movement = Using the keyboard to move the in-game character
* Aiming = Using the mouse to aim the gun at the targets
* Gun controls = using the keyboard/mouse to

User output includes features where the application outputs UI elements in order to directly interact with the player.

* Main Menu = First UI element the player will see, allows the player to navigate between the different features of the game e.g., different modes, viewing statistics.
* Crosshair = UI element that is shown in the middle of the screen in order to show the user where the bullets from the gun will hit.
* Outputting graphs = Showing the graphs of the players statistics to the player.
* Outputting player stats = Showing the player the raw value statistics of their performance.

### 3.1.3 Controller

* Generating lines between targets = allows the player to click between targets faster because it is a straight lune
* Calculating graphs = Uses the player results from the task to display a graph that can be easily understood compared to raw data
* Spawning in targets = calculates random coordinates to spawn targets in so the player does not know where they will spawn in order to help with reaction time and decision making.

Calculating statistics includes features that are used to calculate metrics that determine how well the player has performed in the task both overall and in different skills.

* Calculating accuracy = calculates the players accuracy (percentage of total shots taken that have hit the target)
* Calculating Gain = Calculates how far away from the target the player shoots when they miss
* Calculating error size = Calculates how far away from the center the player hits on average to show how precise the player is.
* Calculating reaction time = Calculates the average time taken between the spawning of a target to when the player hits it to determine how quickly the player can react to a target being spawned in.
* Calculating overall score = combines the values of all the other data values in order to produce an overall number that is used to determine how well the player performed in that specific task.

## 3.2 Test Data

Most inputs will be made through keys on the keyboard as well as buttons on the screen and inputs on the mouse:

* Movement keys
  + Character moves in the correct direction relative to the button pressed
  + Character can only jump when they are touching the ground
  + Character falls accordingly following calculations using suvat equation
* Buttons on screen
  + Must change color when hovering over it with the mouse
  + Must do the correct thing when pressed
  + Must only perform action when the mouse Is within the bounds of the button
* Gun control buttons
  + Must fire once per button press
  + Must only destroy target objects
  + Must change FOV when ADS
  + Must change Sensitivity when ADS
* Player inputs
  + Must be within range of value
* Menu controls
  + Buttons on menu must switch to correct corresponding screen
  + Game must pause in the background when options menu is open

# 4.0 prototype 1

## 4.1 introduction

The focus of the prototype will be to ensure that the system designed to spawn and destroy targets as well as some parts of data collection and settings work correctly.

The features I plan to add are:

* Target Spawning
* Settings (partial)
* Data and Feedback (partial)
* Line pathing (partial)

### 4.1.1 Target Spawning

This feature will include spawning 3 targets in random positions within a predetermined range to ensure the targets spawn within the player’s line of sight. I will use 4 invisible objects in the shape of a square to set the maximum and minimum values to be used to calculate the spawn position of each target.

### 4.1.2 Settings

This feature will include 3 options that the player can choose to change to be able to personalize the experience to the way that they play the actual games. The 3 settings are FOV, Sensitivity and ADS multiplier. I have chosen these 3 settings because I felt like they have the most impact on the player as well as being very easy to implement initially.

### 4.1.3 Data and feedback

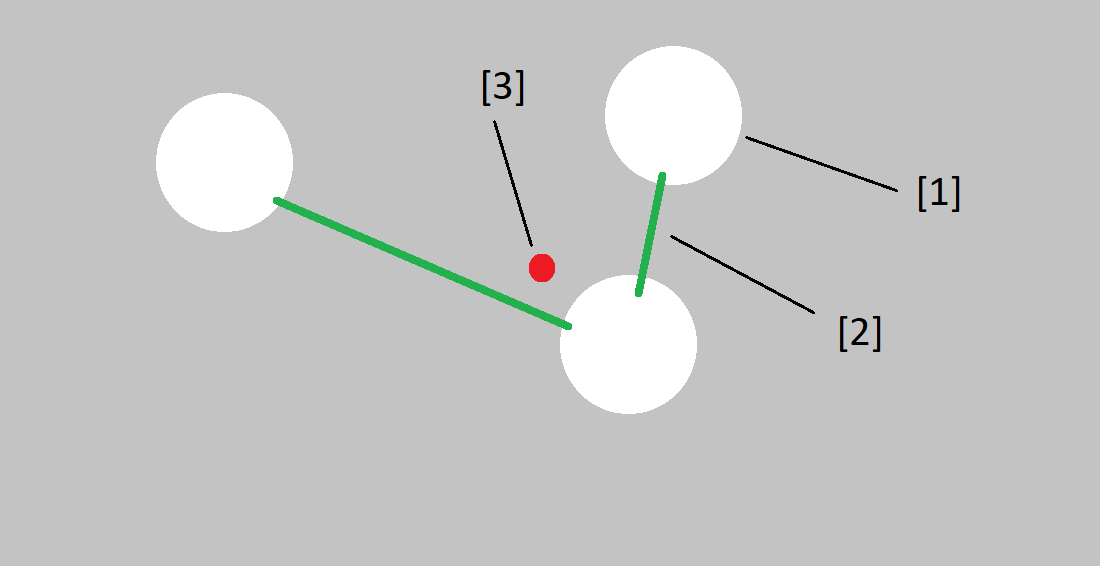
This feature will just include the calculations of the core data values that will be given as feedback to the player in later prototypes. I am keeping this feature simple in this prototype because I want to focus more on the core gameplay rather than the data side in this prototype.

### 4.1.4 Line Pathing

This feature will be kept very simple as it is just exploring a method of generating lines between targets. This is not the final method I will use and was just a test to see how the line generation would work in future prototypes and if id use this same method of generating lines again.

## 4.2 UI Design

### 4.2.1 Game screen



The way the user will interact with this screen is that they can use th W,A,S,D keys to move the player around the scene, they can use the space bar to jump and the shift key to sprint. They can look around the screen using the mouse and the left mouse button shoots the gun. They can use the escape button to open the options menu.

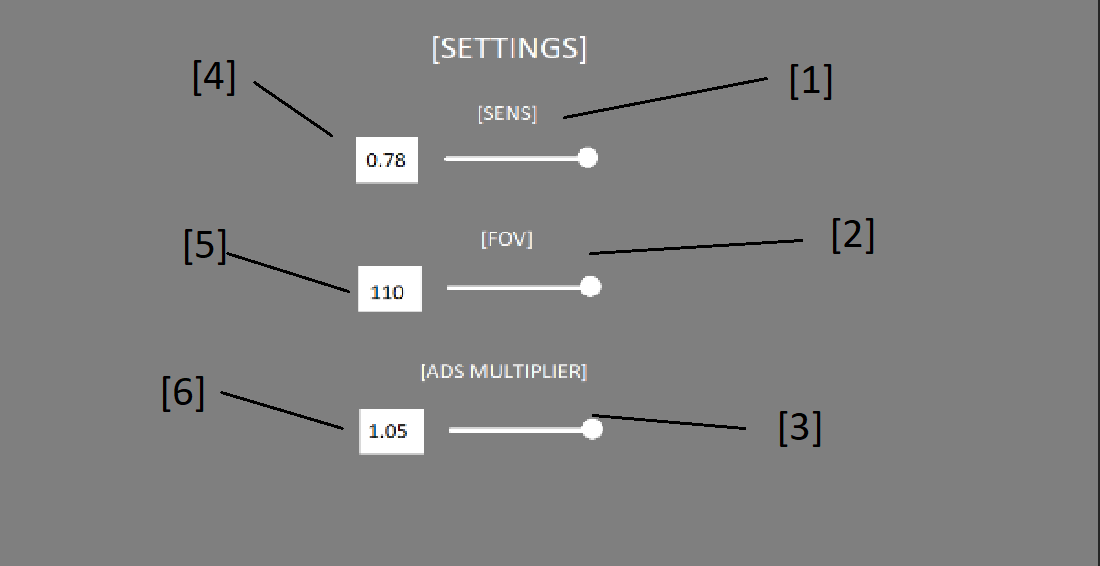
This screen is the main screen that you will see when you are doing a task. Some elements on this screen will change based on which task you are doing however, the main elements that will always be there are the crosshair and the targets on screen.

[1] This is the target object that the player will be shooting, the target’s spawn location is randomly generated using the dimensions of the spawn object to get max and minimum values for where the targets can spawn.

[2] This is the line that is rendered between the targets to guide the player to the other target in the straightest line possible because it is the quickest path between the 2 targets.

[3] This is the crosshair; it is a small red circle in the middle of the screen that indicates where the bullet is going to hit on the target and allows the player to aim to different targets.

### 4.2.1 Settings



This screen is the settings screen that the player will go to when they want to change their settings such as sensitivity, FOV and ADS sensitivity.

The way the player can interact with this screen is that they can use the mouse to move the cursor around the screen and the left mouse button to interact with the sliders and input fields. To use the input field, they click the input field to select it then use the keyboard to input the values they wish then the enter key to submit the values into the input field.

[1] is a slider that allows the player to change the sensitivity of the player camera

[2] is a slider that allows the player to change the FOV of the player camera

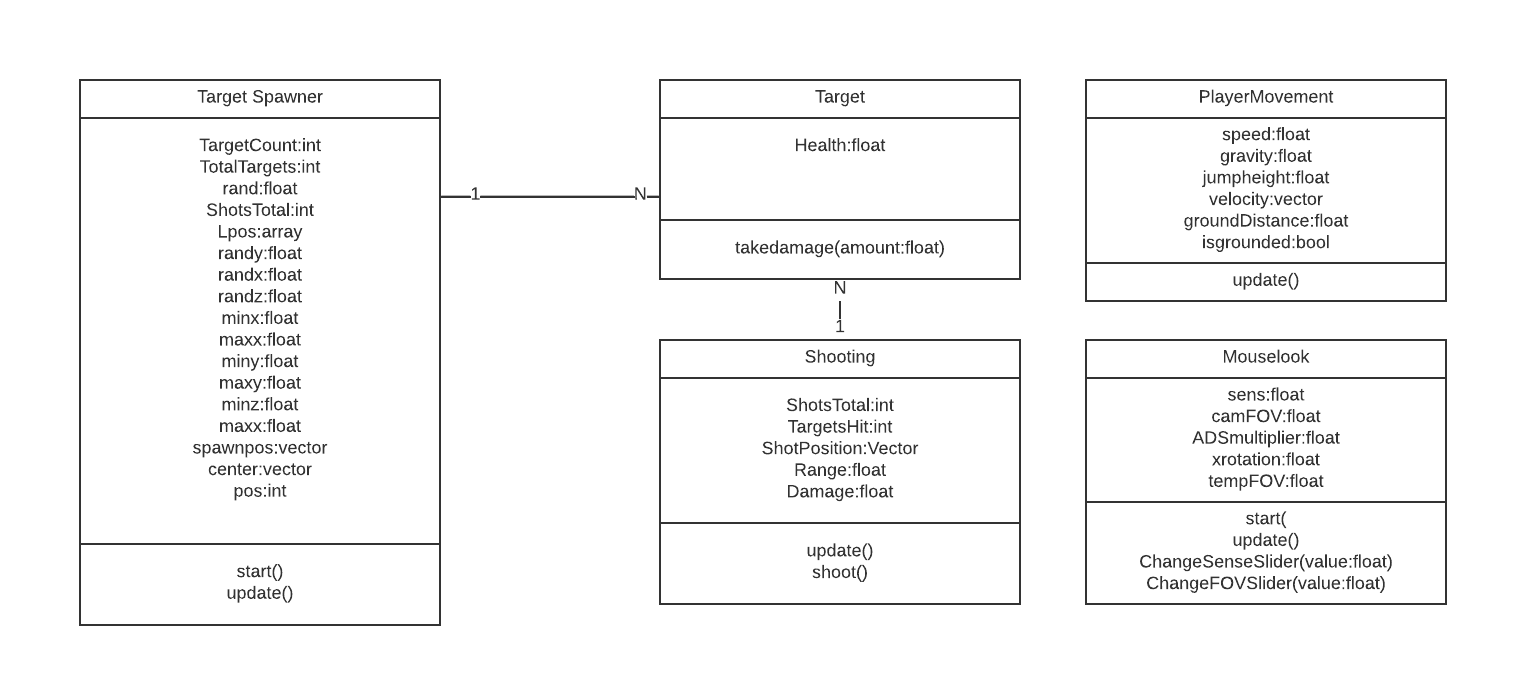
[3] is a slider that allows the player to change the sensitivity of the player camera when aiming in

[4] is an input box that allows the player to change the sensitivity of the player camera

[5] is an input box that allows the player to change the FOV of the player camera

[6] is an input box that allows the player to change the sensitivity of the player camera when aiming in

## 4.3 Class diagram



### 4.3.1 Classes

#### 4.3.1.1 TargetSpawner

The **TargetSpawner** class is the class that handles a large portion of the game operations. It calculates the target spawn positions then spawns them as well as gets and stores the target positions in an array and then renders lines between them.

#### 4.3.1.2 Target

The **Target** class assigns a health value to the target object that is instantiated using the **TargetSpawner** class then destroys the target object when the health reaches 0.

#### 4.3.1.3 Shooting

The **Shooting** class handles the calculation of where the gun is going to shoot using raycasts from the player camera to the middle of the screen. It also handles the destruction of targets by reducing the health of the target by a set value (damage) as well as has a range value. It also handles the calculations of both error size and accuracy

#### 4.3.1.4 PlayerMovement

The **PlayerMovement** class handles the calculations for moving the player character based on the player input using the keyboard. This includes walking, sprinting, jumping and gravity calculations.

#### 4.3.1.5 MouseLook

The **MouseLook** class handles the movement of the player character based on the player input using the mouse, this includes moving the player body horizontally to look left and right as well as moving the player camera vertically to look up and down and locks the mouse in the middle as well as makes it invisible. It also handles the calculation of the ADS sensitivity based on the value provided by the player input in settings and also changes the sensitivity and player camera FOV values based on the player input using the settings.

### 4.3.2 Methods

TargetSpawner - Start(): This method is run at the start of the code to calculate the minimum and maximum values of the box collider that the targets spawn in, set the target frame rate and set the number of line positions in the line renderer to the number of targets that will be spawned.

TargetSpawner – Update(): This method is run once per frame and is used to get and update the array of targets, calculate the random spawn positions of the targets and spawn them in those positions as well as render the lines between the targets.

Die(): It reduces the health of the target by the amount of damage that the gun does. When the health value reaches 0 then the target is destroyed.

Shoot(): It casts a raycast from the position of the camera into the middle of the screen where the crosshair is. If the object that has been hit contains a target script, it will reduce the health of the target by the value damage. It also calculates error size by taking the average distance away from the center in each target.

Shooting – update(): It runs the shoot() method every time the shoot button is clicked (default left click) and increments the value shotstotal by 1.

PlayerMovement – update(): Moves the player character according to the inputs given by the keyboard and checks whether the player is on the ground before allowing them to jump in the air. It also calculates the gravity that is applied to the player every single frame.

MouseLook – start(): Locks the cursor in the middle of the screen and makes it invisible.

Mouselook – update(): It rotates the player camera and player body according to the inputs given by the player by applying quaternion rotation vectors to both objects. It also changes the sensitivity and the FOV when the player aims down sights.

ChangeSenseSlider(): It changes the sensitivity of the player camera based on the value that is passed through from the slider in the options menu.

ChangeFOVSlider(): It changes the FOV of the player camera based on the value that is passed through from the slider in the options menu.

ChangeADSValue(): Changes the value of the ADS multiplier based on the value that the player inputs using a slider on the options menu

OnSensInput(): Changes the value of sens to the value that is input by the player using an input box in the options menu

OnFOVInput(): Changes the value of camFOV to the value that is inputted by the player using an input box in the options menu

OnADSInput(): Changes the value of ADS to the value input by the player using an input box in the options menu

## 4.4 Data Structure

4.4.1 TargetSpawner

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Justification |
| Lpos | Array | Stores the currently spawned target game objects in order to see how many are spawned in as well as render the lines between the targets by looping through the array. |
| TotalTargets | integer | Number of targets that are currently spawned in which is used to control how many targets are spawned in. |
| TargetCount | integer | The max number of targets that can be spawned in at one time. |
| pos | integer | A counter to control the while loop that is used to loop through the Lpos array in order to render the lines between the targets |
| randy | float | The random value that determines the y position of where the target us going to be spawned. |
| randx | float | The random value that determines the x position of where the target us going to be spawned. |
| randz | float | The random value that determines the z position of where the target us going to be spawned. |
| minx | float | The minimum value of x that is used to generate the spawn position using random.range. This value is made by taking away the x length of the spawn collider from the x position of the center of the collider to give the smallest x value that the target is able to spawn in. |
| miny | float | The minimum value of y that is used to generate the spawn position using random.range. This value is made by taking away the y length of the spawn collider from the y position of the center of the collider to give the smallest y value that the target is able to spawn in. |
| minz | float | The minimum value of z that is used to generate the spawn position using random.range. This value is made by taking away the z length of the spawn collider from the z position of the center of the collider to give the smallest z value that the target is able to spawn in. |
| maxx | float | The minimum value of x that is used to generate the spawn position using random.range. By adding the x length of the spawner to the x position of the center to produce the biggest x value that the target can spawn in. |
| maxy | float | The minimum value of y that is used to generate the spawn position using random.range. By adding the y length of the spawner to the y position of the center to produce the biggest y value that the target can spawn in. |
| maxz | float | The minimum value of z that is used to generate the spawn position using random.range. By adding the z length of the spawner to the z position of the center to produce the biggest z value that the target can spawn in. |
| spawnpos | vector | Combines the randomly generated x, y and z coordinates in to a vector that is used as an instantiation parameter to spawn the targets in the randomly generated position. |
| center | vector | The position of the center of the spawner that is used to calculate the maximum and minimum x, y and z values that the targets can spawn in. |
| health | float | The health of the target that is used to determine when to destroy the target when it reaches 0. |
| damage | float | The damage that the gun does to the target. The higher the value, the more damage it does to the target and the lower the value, the lower damage it does to the target |
| range | float | How far away the target can be to register a hit, the higher the value, the further away you can shoot the target and the smaller the value, the closer the target has to be to hit it. |
| shotPosition | vector | The real world coordinated that the player has shot |
| shotsTotal | integer | The total number of shots the player has taken. This number is incremented by 1 every time the mouse is clicked. |
| targetsHit | integer | The total number of targets that the player has hit and destroyed. This number is incremented by 1 every time the target health that the player has hit reaches 0. |
| accuracy | float | The percentage of shots the player has taken that have hit a target. This value is calculated by dividing **targetsHit** by **shotsTotal** then multiplying that value by 100 to produce a percentage. |
| speed | float | How fast the player moves when walking normally |
| gravity | float | The real-world gravity value of 9.18 which is made negative to calculate the effect of downwards gravity on the player. |
| jumpheight | float | How high the player jumps when they press the jump input button. |
| velocity | vector | Used to change the player character’s position by applying the gravity and jump calculations to the vector. |
| groundDistance | float | The distance from the ground the player hovers when touching the ground. |
| isgrounded | bool | Used to verify whether the player is touching the ground or not to prevent them from double jumping. This is made true or false based on if the groundcheck object is touching the ground. |
| sens | float | The sensitivity of the camera to dictate how fast it moves. |
| ADSmultiplier | float | A number to change the sens value when aiming in based on the player preferences. |
| xrotation | float | The value that dictates the angle at which the camera gets locked at to prevent the player from looking too far up or down |
| camFOV | float | The value that dictates the FOV of the camera. |

## 4.5 Algorithms Design

4.5.1 Calculating the min and max x, y and z values that the targets can spawn in  
  
minx = corner1.transform.position.x;

maxx = corner2.transform.position.x;

miny = corner3.transform.position.y;

maxy = corner1.transform.position.y;

minz = corner1.transform.position.z;

maxz = corner1.transform.position.z;

This is the algorithm that is used to calculated the minimum and maximum values of x, y and z that the targets are able to spawn in. These values are calculated using the size of the collider that the targets spawn in and then using the length and width as well as the center in order to calculate the values. For example, minx is calculated by taking the position of the center of the collider then taking that away from the length of the collider divided by 2 which gives you the very left edge of the collider which is the smallest value of x that a target can spawn in. This same logic is applied to the max values but instead of taking away from the center position, it is added to the center position in order to give the opposite side of the minimum to give the maximum.

### 4.5.2 Generating random coordinates for spawning targets

WHILE TotalTargets < TargetCount:  
 randy = random.range(miny,maxy)  
 randx = random.range(miny,maxy)  
 spawnpos.y = randy  
 spawnpos.x = randx  
 spawnpos.z = minz  
  
 clone = Instantiate(Target, spawnpos, tspawn.transform.rotation)  
  
 TotalTargets += 1  
 END WHILE

This is the algorithm that is used to generate the random coordinates of where the target is spawned. It consists mainly of a counter controller while loop that uses the total targets that are currently spawned in as the counter and the max number of targets that can be spawned at one time as the control. It works by generating 2 random numbers (x and y) then applying those numbers to the x and y values of the spawn position vector. For the z value it just spawns it in the z value of the center position because in this specific task all the targets must be in the same z position. Once the **spawnpos** vector has been fully changed with the random values, we use the **instantiate** function of unity in order to make clones of the target using the parameters(**object, position, rotation)**.

### 4.5.3 Looping through the array of targets to render lines between them

TotalTargets = Lpos.Length;  
  
IF Lpos.length == TargetCount:  
 pos = 0  
 WHILE pos < Lpos.length:  
 LR.SetPosition(pos, Lpos[pos].transform.position);  
 pos += 1  
 END WHILE  
END IF

This is the algorithm that loops through the array that stores all of the target gameobjects then uses their positions to render lines between them. The **IF** statement is validation to make sure that all the targets are spawned in before rendering the lines between them to prevent and error of a null object reference. Then we use a counter controlled while loop to loop through the array and spawn the lines between the positions using the **linerenderer** component function **SetPosition** using the parameters**(index, position)**. This algorithm is used for the LinePathing feature.

### 4.5.4 Taking health off the target when shot and destroying it when health reaches 0

function takedamage(amount)  
 health -= amount  
 *if* health <= 0:  
 die()  
 endif  
  
 function die():  
 Destroy(gameobject)  
 endfunction  
endfunction

This is the algorithm that takes the health away from the target when it gets shot using the **amount** parameter which is the damage value from the **shooting()** class. First it takes away the damage amount from the health of the target then goes through an if statement that calls the **die()** function when the health reaches 0 or below. The **die()** function destroys the gameobject that the script is stored on which in this case is the target when it Is called.

### 4.5.5 calculating error size and accuracy

IF target != null  
 shotposition = hit position  
 errorsize += shotposition[1] - target.position[1]  
 errorsize += shotposition[0] - target.position[0]  
 targetshit += 1  
 errorsize = errorsize / targetshit  
  
 accuracy = (targetshit / shotsTotal) \* 100  
END IF

This is the algorithm that calculates both error size and accuracy. First it starts by validating that the object that the player has hit is a target by making sure that it contains a target script. It then assigns the position of where the player has shot as shotposition. To calculate the errorsize we take away the x and y values of the center of the target from the x and y values of where the player shot to get the distance between the position of the shot and the center of the target. The reason why there is no z distance calculated is because the targets are spherical in shape therefore it is thickest in the center as opposed to the edges so if you shoot towards the center, which is where you should be shooting for the best precision, it means a further z distance than shooting the edges therefore it throws off the errorsize. To get the average error size over all targets hit we just divide the error size value by the number of targets hit. Accuracy is calculated by dividing the number of targets hit by the total number of shots taken then multiplying that value by 100. This algorithm is used for the data and feedback feature.

## 4.6 Implementation

### 4.6.1 Mouse\_Look code

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class old\_mouselook : MonoBehaviour

{

//Taken from brackeys tutorial on fps character controller https://www.youtube.com/watch?v=\_QajrabyTJc

//Vlaue for how fast the player wants the look sensetivity

private float sens = 125f;

private float tempSense;

private float tempFOV;

private float tempADS;

private float ADSmultiplier = 0.75f;

private bool paused;

//Options vars

[SerializeField] private TMP\_InputField sensInputField;

[SerializeField] private TMP\_InputField fovInputField;

[SerializeField] private TMP\_InputField adsInputField;

[SerializeField] private GameObject optionsMenu;

//End Options vars

//Transform of the physical player body to rotate the player body along the Y axis to make sure that the player is always moving in the direction they are facing

[SerializeField] private Transform playerbody;

//Value of xrotation to allow the camera to rotate along the X axis to look up and down

private float xrotation = 0f;

private float CamFOV = 60f;

void Start()

{ //Code runs at the very start to lock the cursor in the middle of the screen

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible = true;

optionsMenu.SetActive(false);

//sets the text within the input field to the value of the variable that it changes

sensInputField.text = sens.ToString();

fovInputField.text = CamFOV.ToString();

adsInputField.text = ADSmultiplier.ToString();

}

//called when the SensSlider value is changed

public void ChangeSenseSlider(float value)

{

sensInputField.text = value.ToString();

sens = value;

}

//called when the FOVslider value is changed

public void ChangeFOVSlider(float value)

    {

fovInputField.text = value.ToString();

CamFOV = value;

    }

//called when the ADSslider value is changed

public void ChangeADSValue(float value)

    {

adsInputField.text = (Mathf.Round(value \* 100) / 100).ToString();

ADSmultiplier = value;

    }

//called when the sens inputfield is changed and entered

public void OnSensInput()

{

//checks if the input contains numbers or characters, if true then it changes the sens to the value

if (float.TryParse(sensInputField.text, out tempSense) == true)

{

//makes sure the entered value is in range of the sensitivity

if (tempSense >= 10 && tempSense <= 500)

{

sens = tempSense;

}

//if false then it promps the player to enter a numerical value between the min and max

else

{

Debug.Log("please enter a value between 10 and 500");

}

}

//if false then it promps the player to enter a numerical value

else

{

Debug.Log("please enter a number value");

}

}

//called when the ADS input field value is changed and entered

public void OnADSInput()

{

//checks if the input contains numbers or characters, if true then it changes the sens to the value

if (float.TryParse(adsInputField.text, out tempADS) == true)

{

//makes sure the entered value is in range of the ads

if (tempADS >= 0.01f && tempADS <= 1)

{

ADSmultiplier = tempADS;

}

//if false then it promps the player to enter a numerical value between the min and max

else

{

Debug.Log("please enter a value between 0 and 1");

}

}

//if false then it promps the player to enter a numerical value

else

{

Debug.Log("please enter a number value");

}

}

//called when the FOV input field is changed and entered

public void OnFOVInput()

{

//checks if the input contains numbers or characters, if true then it changes the sens to the value

if (float.TryParse(fovInputField.text, out tempFOV) == true)

{

//makes sure the entered value is in range of the fov

if (tempFOV >= 60 && tempFOV <= 110)

{

CamFOV = tempFOV;

}

//if false then it promps the player to enter a numerical value between the min and max

else

{

Debug.Log("please enter a value between 60 and 110");

}

}

//if false then it promps the player to enter a numerical value

else

{

Debug.Log("please enter a number value");

}

}

// Update is called once per frame

void Update()

{

if (!paused)

        {

//Gets the X input and multiplies it by the sens for the speed and Time.deltaTime to make the camera movement frame independent

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

//Gets the X input and multiplies it by the sens for the speed and Time.deltaTime to make the camera movement frame independent

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

//Takes x rotation from mouseY because if it was equal, the rotation of the camera would be flipped compared to the Y input since it is rotating around the X axis

xrotation -= mouseY;

//Clamps the rotation of the camera around the X axis to make sure that the player cant look too far behind or too far down

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

//Rotates the camera using quaternion and euler angles around the X axis to allow the camera to look up and down

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

//Applies a rotation transformation on the up vector(y vector) of the playerbody to rotate it around the y axis to look left and right

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

}

//runs when the right mouse button is pushed down

if (Input.GetMouseButtonDown(1))

        {

//sets the fov to be 0.75x the current fov to zoom in

CamFOV \*= 0.75f;

//multiplies the sens by the ads multiplier that the player can change in the settings

sens \*= ADSmultiplier;

        }

//runs when the right moust button goes up after being pressed down

if (Input.GetMouseButtonUp(1))

{

CamFOV /= 0.75f;

sens /= ADSmultiplier;

}

//runs when the escape button is pressed on the keyboard

        if (Input.GetKeyDown(KeyCode.Escape))

        {

//flips the value of paused

paused = !paused;

}

        if (!paused)

        { //While not paused

optionsMenu.SetActive(false);

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible = false;

        }

        else

        { //While Paused

optionsMenu.SetActive(true);

Cursor.lockState = CursorLockMode.None;

Cursor.visible = true;

        }

//changes the main camera's FOV

Camera.main.fieldOfView = CamFOV;

}

}

### 4.6.2 TargetSpawner Code

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using UnityEngine.SceneManagement;

using TMPro;

public class targetspawner\_randshot : MonoBehaviour //dan was here!!!

{

[SerializeField] private GameObject Target;

[SerializeField] private GameObject tspawn;

[SerializeField] private LineRenderer LR;

private GameObject[] Lpos;

private int TotalTargets = 0;

private int TargetCount = 3;

private float t;

private int pos;

private float randy;

private float randx;

private float randz;

[SerializeField] private GameObject corner1;

[SerializeField] private GameObject corner2;

[SerializeField] private GameObject corner3;

[SerializeField] private GameObject corner4;

private float minx;

private float maxx;

private float miny;

private float maxy;

private float minz;

private float maxz;

private Vector3 spawnpos;

private Vector3 center;

void Start()

{

//calculates min and max values based on the positions of the corners of the spawn area

minx = corner1.transform.position.x;

maxx = corner2.transform.position.x;

miny = corner3.transform.position.y;

maxy = corner1.transform.position.y;

minz = corner1.transform.position.z;

maxz = corner1.transform.position.z;

// taken from YouTube video on how to calculate spawn positions of targets https://www.youtube.com/watch?v=fQTaJIUFI0A

//sets the framerate of the game to 100

Application.targetFrameRate = 100;

//Sets the length of the line render array to the target count

LR.positionCount = TargetCount;

}

// Update is called once per frame

void Update()

{

Lpos = GameObject.FindGameObjectsWithTag("target");

TotalTargets = Lpos.Length;

//if all 3 targets are on screen

if(Lpos.Length == TargetCount)

        {

pos = 0;

//loops through the array of targets

while(pos < Lpos.Length)

            {

//sets the position of the line renderer to be between each value in the array

LR.SetPosition(pos, Lpos[pos].transform.position);

//increments the counter

pos += 1;

            }

        }

        //creates a clone gameobject

        GameObject clone;

//Loops as long as there are less targets than the target count

while (TotalTargets < TargetCount)//rohannnn:)

{

//generates a random number between the minimum and maximum values for each axis

randy = Random.Range(miny, maxy);

randx = Random.Range(minx, maxx);

//sets the spawnposition of each axis to the random number that has been gemerated for that spawn position

spawnpos.y = randy;

spawnpos.x = randx;

spawnpos.z = center.z;

//clones the target gameobject in the position and rotation of the spawnposition

clone = Instantiate(Target, spawnpos, tspawn.transform.rotation);

//adds a target tag to the clone to be added to the array

clone.tag = "target";

//increments total targets by 1 to prevent infinite while loop

TotalTargets += 1;

}

}

}

### 4.6.3 Shooting Code

### 4.6.4 PlayerMovement Code

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

//Taken from brackeys tutorial on fps character controller https://www.youtube.com/watch?v=\_QajrabyTJc

//allows the referencing of the character controller in the code

[SerializeField] private CharacterController controller;

//How fast the player wants to move

private float speed = 12f;

//Standard value for gravity to be used in suvat and jumping equations

[SerializeField] private float gravity = -19.62f;

//The value for how high the player wants to jump

private float jumpheight = 3.0f;

//Makes a vectory3 called velocity to allow the change of the players velocity in both the jumping calculations and the gravity calculations

private Vector3 velocity;

//This transform(position) is placed at the bottom of the player mesh and is used to check wether the player is touching the ground or not

[SerializeField] private Transform groundcheck;

//Distance the player is from the ground when touching it

private float groundDistance = 0.4f;

//this layermask allows the player and the ground to be on the same layer to make sure that the code works if the player is touching a ground object

[SerializeField] private LayerMask groundMask;

//Either true or false on wether the player is touching the ground based on certain parameters

private bool isgrounded;

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

//creates a tiny sphere at the bottom of the player and checks if anything is colliding with the ground mask and changes the bool "isgrounded" accordingly

isgrounded = Physics.CheckSphere(groundcheck.position, groundDistance, groundMask);

//Assigns a value to x and z based on the vertical and horizontal inputs from the player to be used in the movement code

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

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

//if the player is grounded AND the player has no Y velocity, the y velocity of the player is removed. this is to make sure that the gravity pull is not being increased

//while the player is grounded so that when the player is grounded for a long time, the y velocity resets to make sure that gravity acts normal on the next fall

if(isgrounded && velocity.y < 0)

        {

velocity.y = -2f;

        }

//takes the directions that the player is facing and multiplies it by it by the value taken from the player input and places it into a vector3(x,y,z)

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

//When the left shift button is pressed, it multiplies the movespeed by 2 to allow the player to sprint

        if (Input.GetKeyDown(KeyCode.LeftShift))

        {

speed = (speed \* 2);

        }

//When the shift button is released, the movespeed is divided by 2 to stop the player from sprinting when the shift button is not pressed

if (Input.GetKeyUp(KeyCode.LeftShift))

{

speed = (speed / 2);

}

//Uses the Move function with the move vector3 and multiplies it by the player speed and time to allow the player to move

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

//uses the physics equation v = sqrt(h \* -2 \* g) to calculate the amount of velocity needed to jump a certain height

if(Input.GetKeyDown(KeyCode.Space) && isgrounded)

        {

velocity.y = Mathf.Sqrt(jumpheight \* -2f \* gravity);

        }

//Uses the suvat equation DeltaY = (1/2 \* g) \* sqrt(t) to calculate the change in the y velocity of the player due to the gravitational pull

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

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

//hello

}

}

### 4.6.5 Target Code

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class TargetScript : MonoBehaviour

{

    //sets the health of the target

    private float health = 10f;

    //function called when the target is shot

    public void takedamage(float amount)

    {

        //removes the amount passed through takedamage() when its called and removes that value from health

        health -= amount;

        //if the target has no health left

        if(health <= 0)

        {

            //calls the die function

            Die();

        }

        void Die()

        {

            //destroys the target/object this script is attached to when the health reaches 0

            Destroy(gameObject);

        }

    }

    //getter used to return the health that the target has when called

    public float get\_health()

        {

            return health;

        }

}

## 4.7 Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Expected result | Actual result | Proof | Works as intended? | Resolved? |
| Player movement | The user should be able to use W,A,S,D to move around, SPACE to jump and shift to sprint | All of the movement systems work as intended |  | Yes, as you can see in the images to the left, the user is able to move from the left side of the cube to the right side of the cube using the movement controls | No bug to resolve |
| Camera controls | The user should be able to use the mouse to move the camera around to look around the map area and use the right click to aim down the sights | All of the camera movement is working as intended along with the aim down sights |  | Yes, as you can see in the images to the left, the user is able to use the mouse to look from one of the targets to the target above. | No bug to resolve |
| Target functionality | The user should be able to click the left mouse button while looking at a target and that target should be destroyed while the other targets stay in place and a new target is generated in a random position | The targets are destroyed when the user clicks while the crosshair is over the target |  | Yes, as you can see in the proof images, in the first image the target is there but in the second image after I had shot the target, the target dissappeared and a new target spawned in a random position while the 2 other targets remained in the same positions | No bug to resolve |
| Full Options Menu functionality | The options meny should open when the user clicks the escape key, the user should be able to change the value of FOV, sensitivity and ADSmultiplier by using both the sliders and the input fields. | The options meny does open when the excape key is pressed and the values can be changed using both the sliders and the input field | settings for the above image    settings for the above image | Yes, as you can see in the proof images, when I change the value of the FOV in the options menu, when I back out of the settings the FOV does change. This result is the same for all of the settings available to change in the options menu. | No bug to resolve |
| Line generation between targets | There should be lines generated between the targets when they spawn in. The lines should then generate between the targtes that are newly generated as well. When a target is destroyed the line connecting to that target should be destroyed | The lines do generate between targets fine,  The entire system is working functionally. |  | Yes- as you can see in the proof images, when I destroy the target, the lines connecting to it get destyoyed and new lines are generated between the 2 remaining targets and the new target that just got generated | No bugs to resolve |

## 4.8 User Feedback

## 4.9 Remedial Action

## 4.10 Evaluation

# 5.0 Prototype 2

## 5.1 Introduction

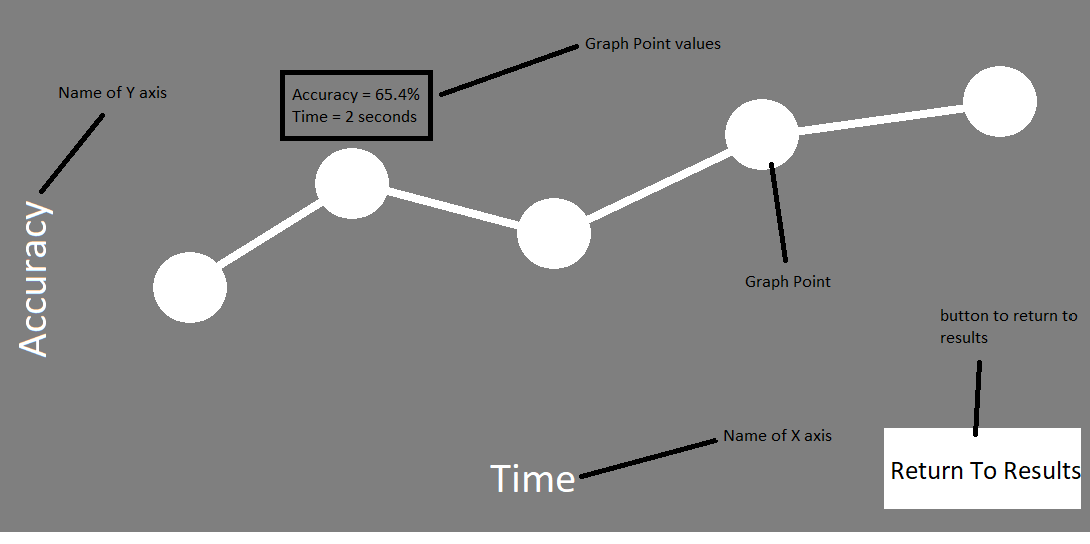
Features I will be adding or completing:

* Adding Graph Making
* Completing Data and statistics
* Completing Line pathing
* Partially Completing settings
* Adding Player Tasks

The main focus of this prototype is to start to start building up more on the analytical side of the application where I will be using different techniques to convey the player’s performance data in a way where they can understand what they mean exactly and to give them an idea of what they could be improving on based on what statistics are shown. I will also start to introduce other player tasks to this prototype in order to be able to see how different data can be collected from different tasks that utilize completely different skills than other tasks

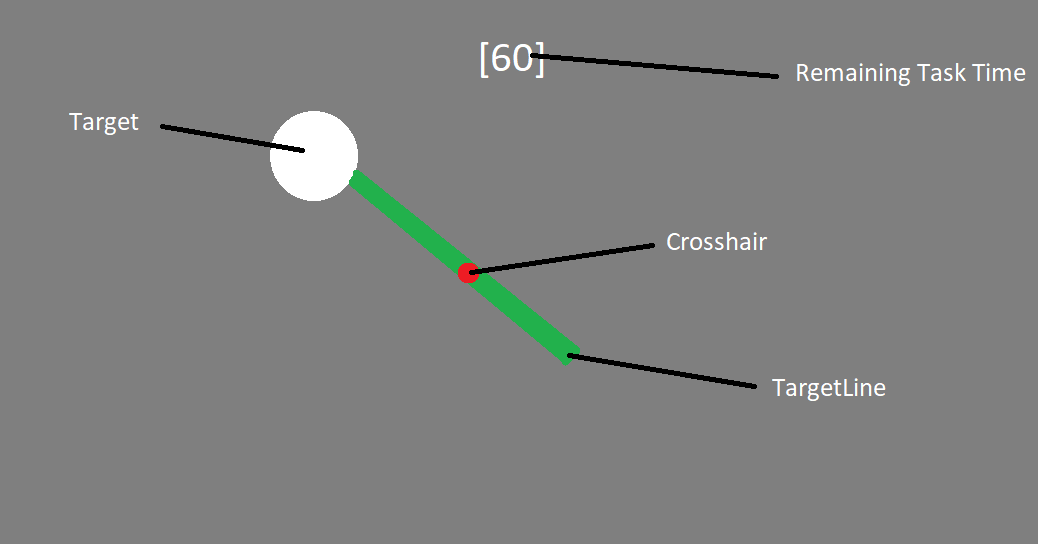
## 5.2 UI design

### 5.2.1 Graph Screen



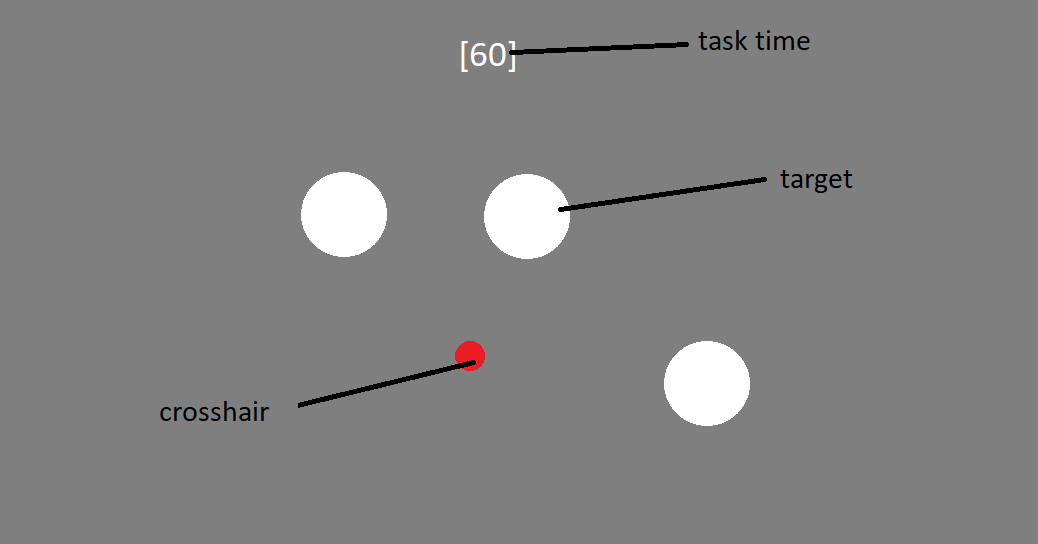
The way the user will interact with the screen would be moving the mouse and when the mouse hovers over one of the graph points, a box will pop up on screen where the mouse is and will display the x and y values of that point in the graph. This will be for every point on the graph.

### 5.2.2 Line path Screen



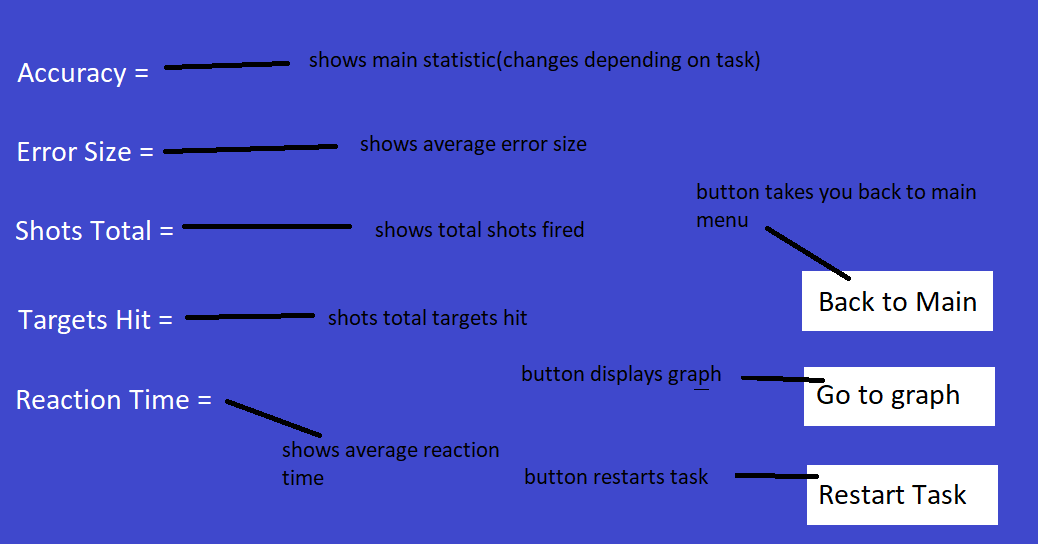
The way the user can interact with this screen is they can use the W, A , S, D keys to move the player character around and the SPACE key to jump in the air. They can also use the MOUSE to move the camera around and use the LEFT MOUSE BUTTON to shoot the gun. If the gun is shot when the crosshair is on the target, the target will be destroyed. When the crosshair hovers over the line connected to the target, the line will turn green. The player can use the ESCAPE key to open the options menu.

### 5.2.3 Gridshot screen



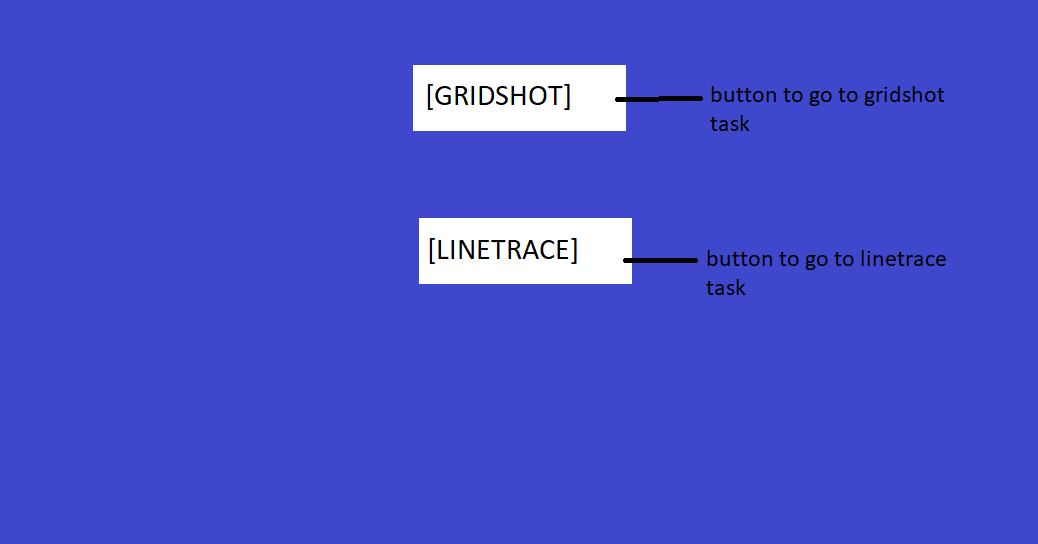
The way the user will interact with this screen will be that they will use the mouse to move the camera around to aim, use left mouse button to shoo the gun, the right mouse button to aim down the sights. They will use the W,A,S,D keys to move around the scene and the space bar to jump and the left shift button to sprint. They will use the escape button to open the options menu.

### 5.2.4 Results Screen



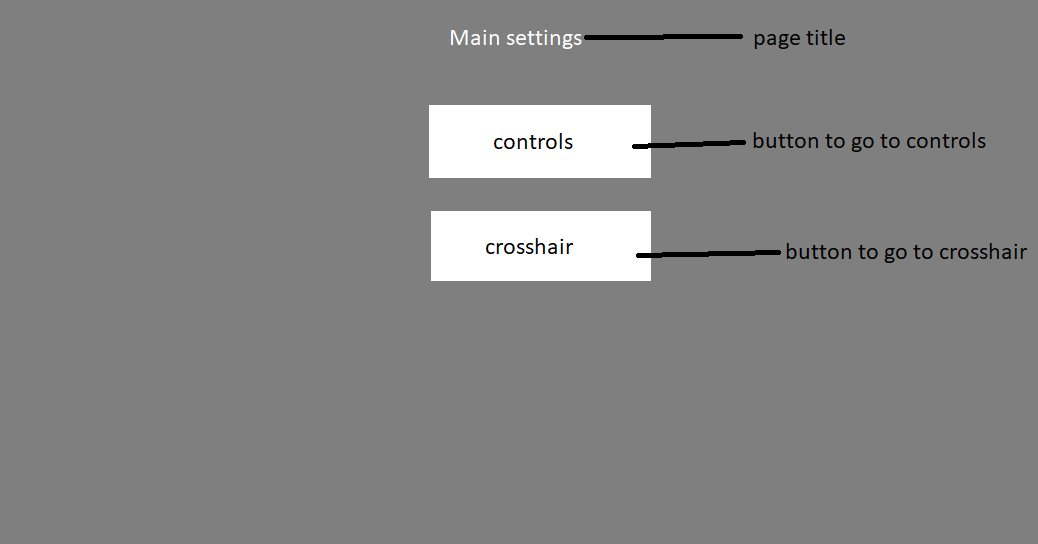
The way the user will interact with this screen will be that they will use the mouse to move the cursor around the screen and the left mouse button to click the buttons to interact with them.

### 5.2.5 Main Menu Screen



The way the user will interact with this screen will be that they will use the mouse to move the cursor around the screen and the left mouse button to click the buttons to interact with them.

### 5.2.6 Main Settings Screen

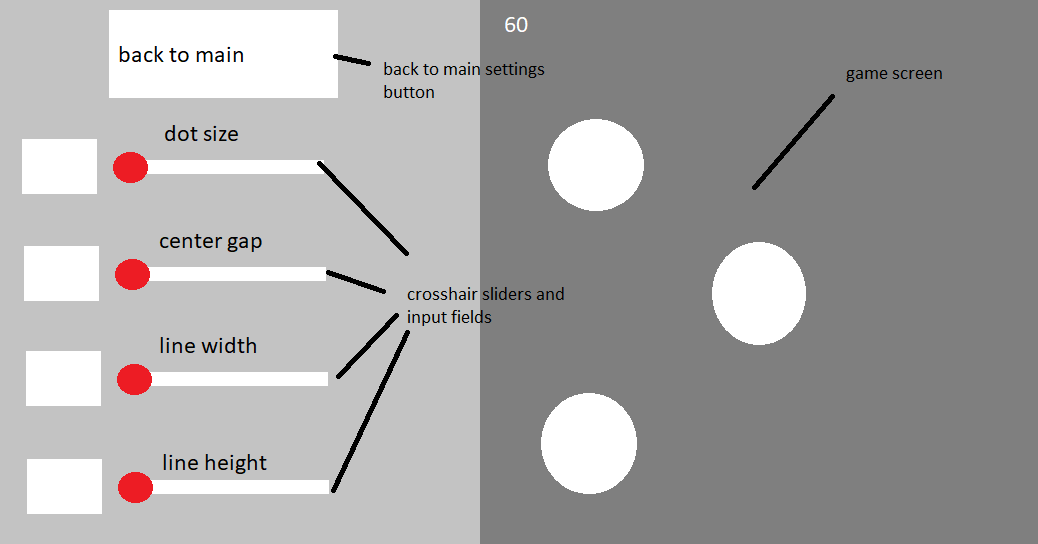


The way the user will interact with this screen will be that they will use the mouse to move the cursor around the screen and the left mouse button to click the buttons to interact with them and they can press the escape button to return back to the game screen.

### 5.2.7 Controls Menu Screen

This screen is the exact same as it is in prototype 1

### 5.2.8 Crosshair Menu Screen



The way the user will interact with this screen will be that they will use the mouse to move the cursor around the screen and the left mouse button to click the buttons, they can click the input field to select, then use the keyboard to type in the values that they want to enter and the enter button to submit what they put in the input field. To use the sliders, the user must click and drag the red circles on the slider to change the value, dragging left will increase the value of the slider until it reaches the maximum value and dragging the slider left will decrease the value of the slider until it reaches the minimum value.

## 5.3 Class Diagram

### 5.3.1 Classes

#### 5.3.1.1 GridshotController

This class will be controlling all the logic behind the Gridshot task. This includes target spawning, shooting, score and stats calculation, as well as interacting with the other classes in order to display stats and interact with the player. This class utilizes the use of 3 arrays to spawn targets in a 3x3 grid.

#### 5.3.1.2 LinePathController

This class will be controlling all the logic behind the Line Path task. This includes target spawning, shooting, score and stats calculation, as well as interacting with the other classes in order to display stats and interact with the player. This class utilizes parts of the graph generation algorithm in order to spawn a line from one target to another target.

#### 5.3.1.3 CameraController

This class controls the use of the player camera that is used to aim at the targets and look around the scene. This class also controls the change of the sensitivity, ads multiplier and FOV by using the instance of the options class that is instantiated at the beginning of the task then the setters are used to change the values of the FOV, sens and ads mult according to what the player has input through the options class.

#### 5.3.1.4 MovementController

This class is used to control the movement of the player character within the scene. This includes movement such as walking, jumping and running.

#### 5.3.1.5 Graph

This class is used to generate the graphs of selected player statistics after the task has ended. It does this by using the getter from the instance of either the linepath or gridshot class to access the array of vector 2 positions that have been calculated by that class in order to change the y positions of the graph points in order to represent the value relative to the axis. This class calculates the x positions of the graph by taking in the length of the graph background and the length of the array in order to get equal spacing between the graph points as well as making sure the graph doesn’t overrun the graph background.

#### 5.3.1.6 UIController

This class controls everything to do with the UI such as the crosshair, the timer and the text displayed throughout the use of the app. This class uses instances of the task classes in order to display the resulting statistics from the task on the results screen by using the getters in the task classes to return the statistic values that are taken as the player completes the task

#### 5.3.1.7 Target

This class controls the interaction of the player with the targets by using methods to control the state of the target. It also helps with returning important data that can be used to track player performance.

#### 5.3.1.8 Options

This class controls everything about the options menu and how the options menu interacts with the player and the rest of the classes because when something is changed in the options, that value must then be updated within the other classes using setters within those classes.

#### 5.3.1.9 LineHover

This class changes the color of the target lines in linepath task as well as count for how long the line/targets has been spawned in for and how long the player has been hovering over the line.

#### 5.3.1.10 MainMenu

This class controls switching between different tasks. I has no connection with any other class since it is only used to load different scenes in unity where each scene is a different task.

#### 5.3.1.11 DisplayGraphPointValue

This class controls what value is displayed when the user hovers over the graph point. It does this using formulas to calculate the statistic value of the graph point based on the in game space position of the graph point. For example the statistical y value would be equal to the positional y value of the graph point

### 5.3.2 Methods

#### 5.3.2.1 GridshotController Methods

##### 5.3.2.1.1 start()

This method runs at the start when the program is run. Within the method, all the spawn locations are added to the list of spawns and the graphPoints array is initialized.

##### 5.3.2.1.2 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. This method serves the same purpose within every single class that includes it.

##### 5.3.2.1.3 GenerateAccuracyGraphPositions()

This method generates the y values for the graph positions based on the calculated average accuracy during that second. It takes in the shots total, and the targets hit to calculate the accuracy when the method is run. This method runs every second. It works by using the number of seconds passed during the task and using the seconds as the index to change. For example, after 3 seconds passed, the y value of the 3rd index of graph points is changed to the value of the accuracy.

##### 5.3.2.1.4 EndTask()

This method runs when the time passed is greater that the task time of that specific task. The role of this method is to set up the task to display the results page by disabling the UI, unlocking the mouse and allowing the mouse to be seen.

##### 5.3.2.1.5 SpawnTargets()

This method is used to spawn the targets in the task. It takes in the parameters of the index of the available array to spawn in the selected target in the position of the array. This

##### 5.3.2.1.6 Shoot()

This method is run whenever the player clicks the mouse while in game. This method works by using the unity feature of raycast in order to shoot a straight line instantaneously from the center point of the player camera which returns as true if it hits an object. They raycast is given a range and also to ignore the player layer to prevent the player from shooting themselves. The method then creates an instance of the target script and then checks if it is null or not. If the instance isn't null, it means that the object that was hit is a target then it causes the target to take damage, increments shots total by 1 and updates the target array to account for the destroyed target. It also calculates the error size from the center of the target that has been hit.

##### 5.3.2.1.7 get\_paused() : bool

This is a getter which returns the paused bool

##### 5.3.2.1.8 get\_pregame() : bool

This is a getter which returns the pregame bool

##### 5.3.2.1.9 get\_taskEnded() : bool

This is a getter which returns the taskended bool

##### 5.3.2.1.10 get\_timer() float

This is a getter which returns the timer float

##### 5.3.2.1.11 get\_graphPoints() : vector2[]

This is a getter which returns the graph point array

##### 5.3.2.1.12 get\_taskTime() : float

This is a getter which returns the task time float

##### 5.3.2.1.13 set\_gunVolume(float value)

This is a setter which sets the value of the gunVolume float

#### 5.3.2.2 LinePath methods

##### 5.3.2.2.1 start()

This method runs at the start when the program is run. Within the method, the max values for the spawn positions are set using the 4 corner objects. All the arrays needed within the class are also initialized within this method.

##### 5.3.2.2.2 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. This method serves the same purpose within every single class that includes it.

##### 5.3.2.2.3 spawnTargets()

This method is used to spawn in the targets onscreen for the user. This method works by generating random spawn positions based on the positions of the corner objects which are used to mark the boundaries of the spawn area. It generates random numbers between the minimum and maximum possible values for x and y then spawning the target in the spawn position that has been randomly generated.

##### 5.3.2.2.4 generateLines()

This method generates the lines between the target that just got destroyed and the current target that just spawned in. This works by first initializing the targetLines array which will store the lines active in the current moment. To spawn in the line, first the position is calculated as a midpoint between the target that just got destroyed and the target that just spawned, then the size of the line is stretched to fit between the 2 targets and the rotation is altered so the line directly connects the 2 points.

##### 5.3.2.2.5 generateHoverTimeGraphPositions

This method generates the y positions for the graph points every time a target is hit based on the time spent hovering over the line compared to the total spawn time of the line when it is spawned in. This works by first using the get\_hoverTime() and get\_totalTime() methods from the LineHover class. These methods are called when the target is destroyed within the shoot() function in order to return the time that the line has been spawned in for and the total time the player spent hovering over the line. Then to calculate the percentage hover, we divide hover time by total time and multiply by 100. Then we add a vector2 into the graphPoints list with the y value being the percentage that we calculated and the x value being 0 by default.

##### 5.3.2.2.6 endTask()

This method runs when the time passed is greater that the task time of that specific task. The role of this method is to set up the task to display the results page by disabling the UI, unlocking the mouse and allowing the mouse to be seen.

##### 5.3.2.2.7 shoot()

This method works the exact same way as the method of the same name in the Gridshotcontroller class however, every time a target is destroyed, it calls the get\_hoverTime() and get\_totalTime() methods as well as calls the generateHoverTimeGraphPositions() method. And instead of removing targets from arrays, this method removes the target lines from the targetpoints array in order to prevent it from being able to return after the target that It was attached to was destroyed.

##### 5.3.2.2.8 get\_paused()

Getter which returns paused bool

##### 5.3.2.2.9 get\_pregame()

Getter which returns pregame bool

##### 5.3.2.2.10 get\_taskEnded()

Getter which returns taskended bool

##### 5.3.2.2.11 get\_graphPoints()

Getter which returns graphPoints list

##### 5.3.2.2.12 get\_Timer()

Getter which returns timer float

##### 5.3.2.2.13 get\_errorSize()

Getter which returns error size float

##### 5.3.2.2.14 get\_reactionTime()

Getter which returns reactionTime float

##### 5.3.2.2.15 get\_shotsTotal()

Getter which returns shotsTotal float

##### 5.3.2.2.16 get\_targetsHit()

Getter which returns targetsHit float

##### 5.3.2.2.17 get\_avgPercent()

Getter which returns the averagePercent float

##### 5.3.2.2.18 get\_taskTime()

Getter which returns taskTime float

#### 5.3.2.3 LineHover methods

##### 5.3.2.3.1 start()

This method runs at the start when the program is run. The purpose of this method within this class is to set the default colour of the object to the colour that is starts so that the colour can be reverted when the player takes their mouse off the target

##### 5.3.2.3.2 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. In this method, update is used to increment totalTime and hover time. They way it works is that is uses a bool called counting which is true when the player is hovering over the object and false when not. If counting is true, every time update is run it will add to both hoverTime and total time but if counting is false, it will only add to totalTime.

##### 5.3.2.3.3 onMouseEnter()

This method runs when the player’s mouse hovers over this object. Since the mouse is locked in the middle of the screen, if the player is aiming at the line it will run this method. When this method is run it changes the colour of the object to green and it also sets counting to true.

##### 5.3.2.3.4 onMouseExit()

This method runs when the player’s mouse goes off the object. When this method is run, it sets the object’s colour back to the default colour and also sets counting to false.

##### 5.3.2.3.5 get\_totalTime()

This is a getter which returns the totalTime float when called

##### 5.3.2.3.5 get\_hoverTime()

This is a getter which returns the hoverTime float when called

#### 5.3.2.4 CameraController methods

##### 5.3.2.4.1 start()

This method runs at the start when the program is run. When this method is run, it locks the cursor in the middle of the screen and makes it invisible.

##### 5.3.2.4.2 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. When this method is run It performs all the necessary calculations to rotate the player camera based on the input from the player. It only rotates the player camera when the game is not paused.

##### 5.3.2.4.3 set\_FOV(float value)

This setter is used to change the value of the FOV float

##### 5.3.2.4.4 set\_sens(float value)

This setter is used to change the value of the sens float

##### 5.3.2.4.5 set\_ADS(float value)

This setter is used to change the value of the ADSmultiplier float

#### 5.3.2.5 Graph methods

##### 5.3.2.5.1 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. This method only performs any actions after the game has ended when the taskEnded bool is true

##### 5.3.2.5.2 generateXpositions()

This method takes in the total distance that you want the graph to span across horizontally and uses both that and the number of graph positions to display to generate equal spacing between the graph points on the x axis. This is to make sure that no matter how many graph points or how big the graph space is, all the graph points will be displayed with equal distance between them so that the graph can be read easier

##### 5.3.2.5.3 generateGraphPoints()

This method is used to generate the actual graph points that will be displayed to show the players statistics in that task. This method works by looping through the graphpoints array which stores the positions that the graph points will be generated at. The graph points that are generated are stored in another array called “graphObjects”, first the method generates a 3d sphere shape as a game object, it then changes the position of the sphere to the position of the point in graphpoints, then it makes the sphere slightly larger and depending on the task will add a script onto the sphere to indicate which text to use when displaying the results based on what its value is.

##### 5.3.2.5.4 generateGraphLines()

This method is used to generate the lines between the graph points to connect them to make them look like a normal line graph used to display statistics. This method works by looping through the graphpoints array to access the positional values of the graph points. First it generates a cube in between the current graph point and the next graph point, then it changes the width of the cube to be the same length as the distance between the 2 points that it was generated between so the line will be long enough to connect them. Finally, it rotates the cube so that it points from the first point to the second point.

#### 5.3.2.6 Options methods

##### 5.3.2.6.1 start()

##### 5.3.2.6.2 update()

##### 5.3.2.6.3 goToMainFromCrosshair()

This method takes the user from the main settings page to the crosshair settings page

##### 5.3.2.6.4 goToMainFromControls()

This method takes the user from the main settings page to the controls settings page

##### 5.3.2.6.5 goToControlsFromMain

This method takes the user from the main settings page to the controls settings page

##### 5.3.2.6.6 goToCrosshairFromMain()

This method takes the user from the main settings page to the audio settings page

##### 5.3.2.6.7 get\_paused()

This is a getter which returns the paused bool

##### 5.3.2.6.8 get\_pregame()

This is a getter which returns the pregame bool

##### 5.3.2.6.9 FOVInput(string value)

This method is used to get the input from the input field that is used to input the FOV value that the user wants. It works by taking in a string input that is taken from the text property of the input field then using int.tryParse on it to see if the string contains only numbers or a mix of numbers and letters. The result of the tryParse is then stored in a temporary value within the method and a setter is used to change the value of FOV within the mouseClass

##### 5.3.2.6.10 SensInput(string value)

This methods works exactly the same way as the FOVInput method

##### 5.3.2.6.11 ADSInput(string value)

This methods works exactly the same way as the FOVInput method

##### 5.3.2.6.12 DotSizeInput(String value)

This methods works exactly the same way as the FOVInput method

##### 5.3.2.6.13 CenterGapInput(string value)

This methods works exactly the same way as the FOVInput method

##### 5.3.2.6.14 LineWidthInput(string value)

This methods works exactly the same way as the FOVInput method

##### 5.3.2.6.15 LineHeightInput(string value)

This methods works exactly the same way as the FOVInput method

#### 5.3.2.7 UIcontroller methods

##### 5.3.2.7.1 start()

##### 5.3.2.7.2 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. This method changes the value of the text to the value within the script, so the value displayed onscreen is always correct.

##### 5.3.2.7.3 set\_dotSize(float value)

This is a setter which sets the value of dotsize to the value passed through it

##### 5.3.2.7.4 set\_centerGap(float value)

This is a setter which sets the value of centerGap to the value passed through it

##### 5.3.2.7.5 set\_lineWidth(float value)

This is a setter which sets the value of lineWidth to the value passed through it

##### 5.3.2.7.6 set\_lineHeight(float value)

This is a setter which sets the value of lineHeight to the value passed through it

#### 5.3.2.8 Target methods

##### 5.3.2.8.1 update()

##### 5.3.2.8.2 takedamage(float value)

This method causes the target to take damage when called. This works by reducing the health attribute by the value that is passed through as the value parameter

##### 5.3.2.8.3 die()

This method causes the target to destroy itself. This method is called when the targets health attribute reaches 0.

#### 5.3.2.9 MovementController methods

##### 5.3.2.9.1 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. Within this update method, all the calculations for movement, speed and gravity are done.

#### 5.3.2.10 DisplayGraphPointValue methods

##### 5.3.2.10.1 update()

This method runs every frame. This method serves as the main game loop and all the other methods and needed calculations and procedures are ran within this method. This method only performs any actions after the game has ended when the taskEnded bool is true

##### 5.3.2.10.2 awake()

##### 5.3.2.10.3 showValue(string value)

This method is called when the user’s mouse is hovering over a graph point, it causes the canvas of the graph points to display the value that is attached to the graph point. It works by setting the text on the canvas to a value passed through and then changing the size of it’s container to the size of the text with room for padding.

##### 5.3.2.10.4 hideValue()

This method is called when the player stops hovering over the graph point and it causes the value text to stop displaying by deactivating the gameObject that the script is attached to so the script no longer runs.

#### 5.3.2.11 MainMenu methods

##### 5.3.2.11.1 update()

##### 5.3.2.11.2 goToGridshot()

This method takes the user from the main menu screen to the gridshot task

##### 5.3.2.11.3 goToLinePath()

This method takes the user from the main menu screen to the linePath task

## 5.4 Data Structure

### 5.4.1 GridshotController key attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| shotPosition | Vector3 | This vector3 is the position on the target that the player has shot relative to the target center and is used to calculate the errorsize value | no |
| errorSize | Float | This is the average distance from the center of the target that the user shoots | no |
| shotsTotal | float | This is the total shots that the user has taken | no |
| targetsHit | float | This tracks how many targets the user has destroyed | no |
| taskTime | float | This is the total time that the task lasts | no |
| paused | bool | This keeps track of if the game is paused or not, True = game is paused, False = game isn't paused | no |
| preGame | bool | This keeps track of if the game is in pregame state or not | no |
| taskEnded | bool | This keeps track of whether the task has ended or not | no |
| graphPoints | Vector2 array | This stores the x and y positions of where the graph points will spawn. | no |
| maxTargetSpawned | integer | This is the max number of targets that can be spawned on the screen at any given time | no |
| spawned | GameObject list | This stores the spawn positions of all the targets that are currently spawned in | no |
| available | GameObject list | This stores the spawn positions of all the available to spawn positions, a positions is available if there is no target spawned in that position and also if that position didn’t have a target in it in the last 3 spawns | no |

### 5.4.2 LinePath key attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| minX | float | This is the minimum value for x that the targets can spawn using | no |
| maxX | float | This is the maximum value for x that the targets can spawn using | no |
| minY | float | This is the minimum value for y that the targets can spawn using | no |
| maxY | float | This is the maximum value for y that the targets can spawn using | no |
| minZ | float | This is the minimum value for z that the targets can spawn using | no |
| maxZ | float | This is the maximum value for z that the targets can spawn using | no |
| shotPosition | vector3 | Same as gridshot class | no |
| errorSize | Float | Same as gridshot class | no |
| shotsTotal | float | Same as gridshot class | no |
| targetsHit | float | Same as gridshot class | no |
| taskTime | float | Same as gridshot class | no |
| paused | bool | Same as gridshot class | no |
| preGame | bool | Same as gridshot class | no |
| taskEnded | bool | Same as gridshot class | no |
| graphPoints | Vector2 array | Same as gridshot class | no |
| maxTargetSpawned | integer | Same as gridshot class | no |
| totalHoverTime | float | This is the average time that the player hovers over the line that is generated between targets. | no |
| targetArray | GameObject array | This array stores the targets that are currently spawned on the screen | no |
| points | Vector3 array | This array stores the positions that the line will generate between | no |

### 5.4.3 LineHover attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| TotalTime | Float | This float is used to keep track of the total time that the line is spawned in for so that it can be used to calculate percentage hovertime | no |
| HoverTime | Float | This float is used to keep track of how long the user has been hovering over the line so that it can be used to calculate percentage hovertime | no |
| Counting | Bool | This bool is used to increment the hovertime attribute. When this is true, it means that the player is hovering over the line so the hovertime attribute is incremented by the time that it is being hovered over. | no |

### 5.4.4 CameraController attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| Sensitivity | float | This is a value used to change how fast the camera moves in the calculations, the higher the value, the faster the camera moves and the lower the value the slower the camera moves. | Yes - must be a float between 10 and 1000 |
| ADSMultiplier | float | This is used to change how fast the camera moves when the player is aiming down the sights of the gun. This value can reduce the sensitivity or increase the sensitivity depending on how the user prefers | Yes – must be a float between 0.1 and 2 |
| CameraFOV | float | This controls the field of view of the camera. A higher field of view allows the users to see more of the screen at once which helps with vision of targets. | Yes – must be a float between 60 and 110 |
| paused | bool | This bool controls when the task is running or not. When true, most systems and calculations are paused to allow the player to either take a break or when they want to change their settings. | no |

### 5.4.5 Graph attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| graphObjects | GameObject array | This array stores all the spherical points of the graph that are generated in the generatGraphPoints() method | no |
| graphLines | GameObject array | This array stores all of the graph lines that are generated | no |
| graphPoints | Vector2 Array | This array stores all of the x and y position values that the graph points get generated at | no |
| xValue | Float | This float is the calculated distance between each graph point. This value is calculated in the generateXpositions() method | no |

### 5.4.6 Options attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| paused | bool | Same as the others with the same name | no |
| preGameTimer | float |  | no |
| preGameTimerBool | bool |  | no |
| preGame | bool |  | no |
| taskEnded | bool | Same as the others with the same name | no |

### 5.4.7 UIcontroller attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| paused | bool | Same as the others with the same name | no |
| taskEnded | bool | Same as the others with the same name | no |
| dotSize | float | This attribute controls the size of the center dot in the user’s crosshair. This value can be changed via the slider or the input field within the crosshair settings menu | Yes – must be a float between 0 and 100 |
| centerGap | float | This attribute controls the size of the center gap in the user’s crosshair. This value can be changed via the slider or the input field within the crosshair settings menu | Yes – must be a float between  -25 and 200 |
| lineWidth | float | This attribute controls the width of the lines in the user’s crosshair. This value can be changed via the slider or the input field within the crosshair settings menu | Yes – must be a float between  0and 100 |
| lineHeight | float | This attribute controls the height of the lines in the user’s crosshair. This value can be changed via the slider or the input field within the crosshair settings menu | Yes – must be a float between 0 and 200 |

### 5.4.8 Target attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| health | float | This is how much health the target has and controls when the target is destroyed. When the health reaches 0, the die() method is run and the target is destroyed. This value is decremented using the takedamage() method. | no |
| paused | bool | Same as the others with the same name | no |
| taskEnded | bool | Same as the others with the same name | no |

### 5.4.9 MovementController attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| paused | bool | Same as the others with the same name | no |
| speed | float | The speed attribute controls how fast the player moves when a move input is pressed, the higher the value the faster the player moves. This is because the speed value is used as a multiplier in the calculations for the movement. | no |
| gravity | float | This attribute controls how strong or weak the gravity is. This value is used in the equation to calculate the player movement due to gravity and the equation used to calculate the velocity needed to jump a certain height | no |
| isGrounded | bool | This bool is used to make sure that the user cant jump when they are in the air. When it is true, it means the player is on the ground and is allowed to jump. When the player is not on the ground, this bool is false and doesn’t allow the player to jump until the bool is true. | no |

### 5.4.10 DisplayGraphPointValue attributes

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| taskTime | float | Same as the others with the same name |  |
| hittingPoint | bool | This bool is true when the user mouse is hovering over a graph point. When it is true, it calls the showvalue() method to display the values attached to that graph point and when it is false, it deactivates the gameObject that the script is attached to so it is no longer being displayed. |  |

## 5.5 Algorithm Design

### 5.5.1 gridshot target spawning algorithm

PROCEDURE spawnTargets(index : integer):

Spawned.add(sphere)

Spawned[totalTargetsSpawned].position = Available[index]

Spawned[totalTargetsSpawned].size \*= 4.5

Spawned[totalTargetsSpawned].addComponent(Target Script)

Available.remove(Available[index])

TotalTargetsSpawned += 1

END PROCEDURE

Justification:

This algorithm is the one used to spawn the targets in the gridshot task, it spawns targets in a 3x3 grid by taking a random index from an array called available which stores all the available spawn positions when a target spawns in a position, that position is taken out of the available array. It works by taking in an index which is used to take the index of available that it corresponds to. Then it creates a target in that position then removes the index that was used to spawn the target from the available array so that it can't be used to spawn another target in that same position.

### 5.5.2 linePath target spawning algorithm

PROCEDURE spawnTargets():

Float randx = random.range(minx, maxx)

Float randy = random.range(miny,maxy)

Spawnposition.x = randx

Spawnposition.y = randy

Spawnposition.z = corner1.position.z

targetArray[TotalTargetsSpawned] = Sphere

targetArray[TotalTargetsSpawned].position = Spawnposition

targetArray[TotalTargetsSpawned].size \*= 1.5

targetArray[TotalTargetsSpawned].addComponent(target script)

targetPoints[1] = Spawnposition

TotalTargetsSpawned += 1

ENDPROCEDURE

Justification:

This algorithm spawns targets in random positions that are within a boundary set by 4 corner object. It works by first generating random x and y positions to spawn the targets in based on the minimum and maximum possible values for x and y which are taken from the corner objects. Then it uses the randomly generated spawn positions and places them into a vector3 called spawnposition. It then starts to generate the targets into an array called targetArray which stores the current targets on the screen. First it creates the 3d sphere, then It sets the spawn position to the SpawnPosition vector3, then it increases the size and adds the target script onto the object to allow the other scripts to interact with it. Finally it sets index 1 of the targetPoints array to the spawnposition so that it can be used to connect the line to it and increments the totalTargetSpawned by 1.

### 5.5.3 linePath line spawning algorithm

PROCEDURE generateLines():

FOR i = 0 to targetArray.length - 1:

targetLines[i] = cube

targetLines[i].addComponent(Line Hover script)

targetLines[i].position = (targetPoints[i] + targetPoints[i + 1]) / 2

targetLines[i].size = distance(targetPoints[i], targetPoints[i + 1])

targetLines[i].rotation = fromToRotation(targetPoints[i + 1] - targetPoints[i])

NEXT i

ENDPROCEDURE

Justification:

This algorithm is used to generate the lines that will be connected to the targets in the linepath task, the algorithm generates the lines that will connect to the targets by using the positions that are stored in the targetPoints array which stores the positions of targets that have previously been destroyed so that a line can be generated from the last target that had been destroyed to the current target that has just spawned in.

Steps:

* Generates a 3d cube
* Adds the line hover script to it to allow it to interact with other systems
* Sets the position of the cube to be between the current target spawned and the last target that got destroyed
* Sets the size to be equal to the distance between the 2 points it was generated between
* Rotates the cube to face from one point to the other so they are connected

### 5.5.4 generating graph y positions

PROCEDURE GenerateAccuracyGraphPositions():

accuracy = (targetsHit / shotsTotal) \* 100

IF secondsPassed < graphPoints.Length:

graphPoints[secondsPassed].y = accuracy

ENDIF

ENDPROCEDURE

Justification:

This algorithm is used to generate the y positions of the graph points so that they can be used to display statistics when they are generated, this algorithm takes the value that the graph point will be displaying then sets the y position of the graph point to be equal to the value that it is displaying.

### 5.5.5 generating graph x positions

PROCEDURE generateXpositions(float maxDistance):

pointDistance = maxDistance / graphPoints.Length

FOR i = 0 to graphPoints.Length:

graphPoints[i].x = xValue

xValue += pointDistance

NEXT i

ENDPROCEDURE

Justification:

This algorithm

### 5.5.6 gridshot target shooting algorithm

PROCEDURE shoot():

shotsTotal += 1

create raycast shotHit

IF raycast hits object:

create instance of targetClass

IF targetClass is present on object:

target.takeDamage()

targetsHit += 1

totalTargetsSpawned -= 1

shotPosition = shotHit.point

justDestroyed.add(shotHit.position)

spawned.remove(shotHit.object)

errosize += distance(shothit.position, shotPosition)

ENDIF

ENDIF

ENDPROCEDURE

Justification:

This algorithm is one that is used to destroy targets when they are shot at. It works by using a raycast to shoot at the objects, checks if there is a target script or not, if there is it causes the target to take damage, then increments targetshit, decrements totalTargetsSpawned , sets the shotPositions, removes the spawn position from the spawned array and calculates the errorsize.

### 5.5.7 linepath target shooting algorithm

PROCEDURE shoot():

shotsTotal += 1

create raycast shotHit

IF raycast hits object:

create instance of targetClass

IF targetClass is present on object:

target.takeDamage()

shotPosition = shothit.point

errosize += distance(shothit.position, shotPosition)

IF targetsHit > 0:

hovertime = targetLines[0].getComponent(LineHover script).get\_hoverTime

totalTime = targetLines[0].getComponent(LineHover script).get\_totalTime

ENDIF

destroy(targetLines[0])

targetPoints[0] = shotPosition

targetsHit += 1

totalTargetsSpawned -= 1

ENDIF

ENDIF

ENDPROCEDURE

Justification:

This algorithm works the same way as the gridshot one but the only difference being that it uses a different system to track the targets that have been destroyed

### 5.5.8 generating graph points

PROCEDURE generateGraphPoints():

FOR i = 0 to graphPoints.Length:

graphObjects[i] = sphere

graphObjects[i].position = graphPoints[i]

graphObjects[i].size \*= 3

graphObjects[i].addComponent(displayGraphPointValue script)

NEXT i

ENDPROCEDURE

Justification:

This algorithm generates the physical graph points that will display the statistics of the player in the task.

Steps:

* Creates the 3d sphere object
* Sets the position of the sphere to the current index of the graph points
* Triples the size of the sphere
* Adds the graph point script component

### 5.5.9 generating graph lines

PROCEDURE generateGraphLines():

FOR i = 0 to graphPoints.Length - 1:

graphLines[i] = cube

graphLines[i].position = (graphLines[i] + graphLines[i + 1]) / 2

graphLines[i].size = distance(graphLines[i], graphLines[i + 1])

graphLines[i].rotation = fromToRotation(graphLines[i + 1],graphLines[i])

NEXT i

ENDPROCEDURE

Justification:

This algorithm generates the lines between the graph points.

Steps:

* Creates a 3d cube object
* Sets the position to be in between the current index and the next index
* Sets the size to be equal to the distance between the 2 points it was generated between
* Sets the rotations to rotate from one point to the other so the points connect

## 5.6 Implementation

### 5.6.1 GridshotController Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*using* UnityEngine.UI;

*using* UnityEngine.SceneManagement;

*using* TMPro;

*public class* P2\_GSCONTROLLER : *MonoBehaviour*

{

[SerializeField] *private* P2\_OPT optionsClass;

[SerializeField] *private* GameObject graphBackground;

[SerializeField] *private* Canvas resultsPage;

*private float* damage = 10f;

*private float* range = 100f;

[SerializeField] *private* GameObject gun;

[SerializeField] *private* GameObject playerCharacter;

[SerializeField] *private* Camera playerCamera;

[SerializeField] *private* LayerMask Player;

*//STATS*

*private* Vector3 shotPosition;

*private float* errorSize;

*private float* shotsTotal;

*private float* targetsHit;

*private float* accuracy;

*private float* totalScore;

*private float* targetsHitInSecond;

*public static float* totalReactionTime;

*//STATS*

*//TIME*

*private float* time;

*private float* timer;

*private int* secondsPassed;

*private int* taskTime = 60;

*private* bool taskEnded = *false*;

*private* bool paused;

*private* bool preGame = *true*;

*private* bool pregametimerbool;

*private float* pregametimer = 0;

*//TIME*

*//GRAPH*

[SerializeField] *private* Camera graphCamera;

[SerializeField] *private* Canvas graphCanvas;

*private* Vector2[] graphPoints;

*//GRAPH*

*//Gridshot vars*

*private int* TotalTargetsSpawned = 0; *//Used to be TotalTargets*

*private int* MaxTargetsSpawned = 3; *//Used to be TargetCount*

*private* Vector3 startspawn;

[SerializeField] *private* GameObject spawn1;

[SerializeField] *private* GameObject spawn2;

[SerializeField] *private* GameObject spawn3;

[SerializeField] *private* GameObject spawn4;

[SerializeField] *private* GameObject spawn5;

[SerializeField] *private* GameObject spawn6;

[SerializeField] *private* GameObject spawn7;

[SerializeField] *private* GameObject spawn8;

[SerializeField] *private* GameObject spawn9;

*private List*<GameObject> Spawned = *new List*<GameObject>();

*private List*<Vector3> Available = *new List*<Vector3>();

*private List*<Vector3> justDestroyed = *new List*<Vector3>();

*//End Gridshot vars*

*private void* Start()

{

*//adding all of the possible spawn locations into the available list*

Available.Add(spawn1.transform.position);

Available.Add(spawn2.transform.position);

Available.Add(spawn3.transform.position);

Available.Add(spawn4.transform.position);

Available.Add(spawn5.transform.position);

Available.Add(spawn6.transform.position);

Available.Add(spawn7.transform.position);

Available.Add(spawn8.transform.position);

Available.Add(spawn9.transform.position);

*//setting the max framerate to 60*

Application.targetFrameRate = 60;

*//initialising the graphPoints array to be the same lengh as the number of seconds*

graphPoints = *new* Vector2[taskTime];

}

*void* Update()

{

*//gets the paused bool from the optionsclass every frame*

paused = optionsClass.get\_paused();

preGame = optionsClass.get\_preGame();

*//counts down the timers only when the game isnt paused and not in pregame state*

*if* (!paused && !preGame)

{

time += Time.deltaTime;

timer += Time.deltaTime;

}

*//if the player's y position reaches a certain value*

*if* (playerCharacter.transform.position.y <= -20)

{

*//resets the player's position*

playerCharacter.transform.position = *new* Vector3(-111f, 2f, -19f);

}

*//when the fire button is clicked it calls the shoot() function*

*if* (Input.GetButtonDown("Fire1"))

{

*//only calls shoot when the game isnt paused or in pregame state*

*if*(!preGame && !paused && !taskEnded)

{

shoot();

}

}

*//while there are less targets than the max*

*while* (TotalTargetsSpawned < MaxTargetsSpawned)

{

*//generates a random index between 0 and the length of the list that hold the available spawn positions*

spawnTargets(Random.Range(0,Available.Count));

}

*//if at least one second has passed, it generatest he graph positions every second*

*if* (time > 1)

{

generateAccuracyGraphPositions();

secondsPassed += 1;

time = 0;

}

*//calls the endtask method when secondspassed becomes greater than the task time*

*if*(secondsPassed >= taskTime && !taskEnded)

{

endTask();

}

}

*private void* generateAccuracyGraphPositions()

{

*//calculates accuracy using targetshit and shotstotal as a percentage*

accuracy = (targetsHit / shotsTotal) \* 100;

*//if the timepassed since the start is less than the length of the graph(aka the total time of the task)*

*if*(secondsPassed < graphPoints.Length)

{

*//sets the index of the graphpoints as the second that it is calculated on*

graphPoints[secondsPassed].y = accuracy;

}

}

*private void* endTask()

{

taskEnded = *true*;

resultsPage.gameObject.SetActive(*true*);

playerCamera.gameObject.SetActive(*false*);

graphCamera.gameObject.SetActive(*true*);

Cursor.lockState = CursorLockMode.None;

Cursor.visible = *true*;

}

*private void* spawnTargets(*int* index)

{

*//if at least one target has been hit*

*if* (targetsHit > 1)

{

*//adds the 0 index of justdestroyed to available since that is the spawn position that has been spawned the furthest spawn cycles away from the current spawn cycle*

Available.Add(justDestroyed[0]);

*//removes that spawn position from just destroyed*

justDestroyed.Remove(justDestroyed[0]);

}

*//creates an empty sphere*

Spawned.Add(GameObject.CreatePrimitive(PrimitiveType.Sphere));

*//Adds the chosen index of available to the list of spawned targets*

Spawned[TotalTargetsSpawned].transform.position = Available[index];

*//sets the size of the sphere to be 4.5x larger than default*

Spawned[TotalTargetsSpawned].transform.localScale = *new* Vector3(4.5f, 4.5f, 4.5f);

*//adds the target script onto the target*

Spawned[TotalTargetsSpawned].AddComponent<P2\_TARGET>();

*//changes the target colour to black*

Spawned[TotalTargetsSpawned].GetComponent<Renderer>().material.color = Color.black;

*//Removes the chosen spawn position from the available list since there is now a target in that position and it is no longer available*

Available.Remove(Available[index]);

*//Increments TotalTargetsSpawned by 1 for the counter controlled loop*

TotalTargetsSpawned += 1;

}

*private void* shoot()

{

*//increments shots total by 1 for stats*

shotsTotal += 1;

RaycastHit shotHit;

*//Returns true if the raycast hits an object*

*if* (Physics.Raycast(playerCamera.transform.position, playerCamera.transform.forward, *out* shotHit, range, ~Player))

{

*//Checks the GameObject that has been hit to see if it has a "Target" Script on it and gets that instance of the class*

P2\_TARGET target = shotHit.transform.GetComponent<P2\_TARGET>();

*//If the GameObject does have a target script on it, cause the target to take damage*

*if* (target != *null*)

{

*//tells the target to take damage using the public method in the class*

target.takedamage(damage);

*//increments targetshit by 1 for stats*

targetsHit += 1;

*//decrements totaltargetsspawned by 1 to allow for another target to be spawned*

TotalTargetsSpawned -= 1;

shotPosition = shotHit.point;

*//adds the destroyed target to justdestroyed so it cant be spawned again sraight after*

justDestroyed.Add(shotHit.transform.position);

*//removes the target from spawned since it is no longer spawned in*

Spawned.Remove(shotHit.transform.gameObject);

*//adds the distance between the shot and the center of the target to the errosize value for stats*

errorSize += Vector2.Distance(shotHit.transform.position, shotPosition);

}

}

}

*//SETTERS*

*public void* set\_paused(*bool* state)

{

paused = state;

}

*public void* set\_preGame(*bool* state)

{

preGame = state;

}

*//SETTERS*

*//GETTERS*

*public bool* get\_paused()

{

*return* paused;

}

*public bool* get\_preGame()

{

*return* preGame;

}

*public bool* get\_taskEnded()

{

*return* taskEnded;

}

*public* Vector2[] get\_graphPoints()

{

*return* graphPoints;

}

*public float* get\_Timer()

{

*return* timer;

}

*public int* get\_TaskTime()

{

*return* taskTime;

}

*public float* get\_accuracy()

{

*return* accuracy;

}

*public float* get\_errorSize()

{

*return* errorSize;

}

*public float* get\_reactionTime()

{

*return* totalReactionTime;

}

*public float* get\_shotsTotal()

{

*return* shotsTotal;

}

*public float* get\_targetsHit()

{

*return* targetsHit;

}

*//GETTERS*

}

### 5.6.2 LinePathController Code

*using* System.Collections.Generic;

*using* UnityEngine;

*public class* P2\_LP : *MonoBehaviour*

{

*private float* minx;

*private float* maxx;

*private float* miny;

*private float* maxy;

*private float* minz;

*private float* maxz;

*private* Vector3 SpawnPosition;

[SerializeField] *private* GameObject corner1;

[SerializeField] *private* GameObject corner2;

[SerializeField] *private* GameObject corner3;

[SerializeField] *private* GameObject corner4;

*private float* hoverTime;

*private float* totalTime;

*private float* percentage;

*private* GameObject[] targetArray;

*private* GameObject[] targetLines;

*private* Vector3[] targetPoints;

[SerializeField] *private* P2\_OPT optionsClass;

[SerializeField] *private* GameObject graphBackground;

[SerializeField] *private* Canvas resultsPage;

*private float* damage = 10f;

*private float* range = 100f;

[SerializeField] *private* GameObject gun;

[SerializeField] *private* GameObject playerCharacter;

[SerializeField] *private* Camera playerCamera;

[SerializeField] *private* LayerMask Player;

*//STATS*

*private* Vector3 shotPosition;

*private float* errorSize;

*private float* shotsTotal;

*private float* targetsHit;

*private float* accuracy;

*private float* totalScore;

*private float* targetsHitInSecond;

*public static float* totalReactionTime;

*//STATS*

*//TIME*

*private float* time;

*private float* timer;

*private int* secondsPassed;

*private int* taskTime = 60;

*private* bool taskEnded = *false*;

*private* bool paused;

*private* bool preGame = *true*;

*private* bool pregametimerbool;

*private float* pregametimer = 0;

*//TIME*

*//GRAPH*

[SerializeField] *private* Camera graphCamera;

[SerializeField] *private* Canvas graphCanvas;

*private List*<Vector2> graphPoints = *new List*<Vector2>();

*//GRAPH*

*//Gridshot vars*

*private int* TotalTargetsSpawned = 0; *//Used to be TotalTargets*

*private int* MaxTargetsSpawned = 1; *//Used to be TargetCount*

*private* Vector3 startspawn;

*//End Gridshot vars*

*private void* Start()

{

maxy = corner1.transform.position.y;

miny = corner3.transform.position.y;

maxx = corner2.transform.position.x;

minx = corner1.transform.position.x;

targetArray = *new* GameObject[MaxTargetsSpawned + 1];

targetLines = *new* GameObject[targetArray.Length - 1];

targetPoints = *new* Vector3[MaxTargetsSpawned + 1];

Application.targetFrameRate = 60;

}

*void* Update()

{

*while* (TotalTargetsSpawned < MaxTargetsSpawned)

{

spawnTargets();

}

paused = optionsClass.get\_paused();

preGame = optionsClass.get\_preGame();

*//counts down the timers only when the game isnt paused and not in pregame state*

*if* (!paused && !preGame)

{

time += Time.deltaTime;

timer += Time.deltaTime;

}

*//if the player's y position reaches a certain value*

*if* (playerCharacter.transform.position.y <= -20)

{

*//resets the player's position*

playerCharacter.transform.position = *new* Vector3(-111f, 2f, -19f);

}

*//when the fire button is clicked it calls the shoot() function*

*if* (Input.GetButtonDown("Fire1"))

{

shoot();

}

*if* (time > 1)

{

*//generateHoverTimeGraphPositions();*

secondsPassed += 1;

time = 0;

}

*if* (secondsPassed >= taskTime && !taskEnded)

{

endTask();

}

}

*private void* spawnTargets()

{

*//Generates random values for x and y spawn coordinates*

*float* randy = Random.Range(miny, maxy);

*float* randx = Random.Range(minx, maxx);

*//applies the randomly generated numbers to the spawnpositon vector to use to spawn the targets*

SpawnPosition.y = randy;

SpawnPosition.x = randx;

SpawnPosition.z = corner1.transform.position.z;

*//Creates a sphere and adds to targetArray*

targetArray[TotalTargetsSpawned] = GameObject.CreatePrimitive(PrimitiveType.Sphere);

*//Changes the position of the spawned target to the spawnposition generated above*

targetArray[TotalTargetsSpawned].transform.position = SpawnPosition;

*//Changes the scale to size it by 1.5x*

targetArray[TotalTargetsSpawned].transform.localScale = *new* Vector3(1.5f, 1.5f, 1.5f);

targetArray[TotalTargetsSpawned].AddComponent<P2\_TARGET>();

targetPoints[1] = SpawnPosition;

TotalTargetsSpawned += 1;

*if* (targetsHit > 0)

{

generateLines();

}

}

*private void* generateLines()

{

*for* (*int* i = 0; i < targetArray.Length - 1; i++)

{

*//Creates a primitive cube object*

targetLines[i] = GameObject.CreatePrimitive(PrimitiveType.Cube);

*//Adds the "Colourchanger" script onto the created object*

targetLines[i].AddComponent<Colourchanger>();

*//Sets the position of the cube to be in between the position of the current and next index in the array*

targetLines[i].transform.position = (targetPoints[i] + targetPoints[i + 1]) / 2;

*//Changes the length to be the equal to the distance between the 2 points*

targetLines[i].transform.localScale = *new* Vector3(0.3f, Vector2.Distance(targetPoints[i], targetPoints[i + 1]), 0.3f);

*//Changes the rotation from the default rotation to rotate towards the position of the next point in the array*

targetLines[i].transform.rotation = Quaternion.FromToRotation(Vector3.up, targetPoints[i + 1] - targetPoints[i]);

}

}

*private void* generateHoverTimeGraphPositions()

{

*if* (targetsHit >= 1)

{

*float* tempPercent = (hoverTime / totalTime) \* 100;

percentage += (hoverTime / totalTime) \* 100;

graphPoints.Add(*new* Vector2(0, tempPercent));

}

}

*private void* endTask()

{

taskEnded = *true*;

resultsPage.gameObject.SetActive(*true*);

playerCamera.gameObject.SetActive(*false*);

graphCamera.gameObject.SetActive(*true*);

Cursor.lockState = CursorLockMode.None;

Cursor.visible = *true*;

}

*private void* shoot()

{

*//increments shots total by 1 for stats*

shotsTotal += 1;

RaycastHit shotHit;

*//Returns true if the raycast hits an object*

*if* (Physics.Raycast(playerCamera.transform.position, playerCamera.transform.forward, *out* shotHit, range, ~Player))

{

*//Checks the GameObject that has been hit to see if it has a "Target" Script on it and gets that instance of the class*

P2\_TARGET target = shotHit.transform.GetComponent<P2\_TARGET>();

*//If the GameObject does have a target script on it, cause the target to take damage*

*if* (target != *null*)

{

*//tells the target to take damage using the public method in the class*

target.takedamage(damage);

shotPosition = shotHit.point;

errorSize += Vector2.Distance(shotHit.transform.position, shotPosition);

*if* (targetsHit > 0)

{

hoverTime = targetLines[0].GetComponent<Colourchanger>().get\_hoverTime();

totalTime = targetLines[0].GetComponent<Colourchanger>().get\_totalTime();

}

generateHoverTimeGraphPositions();

GameObject.Destroy(targetLines[0]);

targetPoints[0] = shotPosition;

*//increments targetshit by 1 for stats*

targetsHit += 1;

*//decrements totaltargetsspawned by 1 to allow for another target to be spawned*

TotalTargetsSpawned -= 1;

}

}

}

*//SETTERS*

*public void* set\_paused(*bool* state)

{

paused = state;

}

*//SETTERS*

*//GETTERS*

*public bool* get\_paused()

{

*return* paused;

}

*public bool* get\_preGame()

{

*return* preGame;

}

*public bool* get\_taskEnded()

{

*return* taskEnded;

}

*public List*<Vector2> get\_graphPoints()

{

*return* graphPoints;

}

*public float* get\_Timer()

{

*return* timer;

}

*public int* get\_TaskTime()

{

*return* taskTime;

}

*public float* get\_hoverTime()

{

*return* hoverTime;

}

*public float* get\_errorSize()

{

*return* errorSize;

}

*public float* get\_reactionTime()

{

*return* totalReactionTime;

}

*public float* get\_shotsTotal()

{

*return* shotsTotal;

}

*public float* get\_targetsHit()

{

*return* targetsHit;

}

*public float* get\_avgPercent()

{

*return* percentage / targetsHit;

}

*//GETTERS*

}

### 5.6.3 LineHover Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*public class* LINEHOVER : *MonoBehaviour*

{

*private* Color defaultColor;

*private float* totalTime;

*private float* hoverTime;

*private* bool counting;

*private void* Start()

{

*//Sets the default colour to the colour that the object spawns in with*

defaultColor = gameObject.GetComponent<Renderer>().material.color;

}

*private void* Update()

{

*//adds time to hovertime every frame*

totalTime += Time.deltaTime;

*//if the player is hovering over the line*

*if* (counting)

{

*//adds to hovertime every frame*

hoverTime += Time.deltaTime;

}

}

*private void* OnMouseEnter()

{

*//when hovering over the object, it changes the colour to green*

gameObject.GetComponent<Renderer>().material.color = Color.green;

counting = *true*;

}

*private void* OnMouseExit()

{

*//When no longer hovering over the object, it changes the colour back to the default colour*

gameObject.GetComponent<Renderer>().material.color = defaultColor;

counting = *false*;

}

*public float* get\_hoverTime()

{

*return* hoverTime;

}

*public float* get\_totalTime()

{

*return* totalTime;

}

}

### 5.6.4 CameraController Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*using* UnityEngine.SceneManagement;

*public class* P2\_CAMERACONTROLLER : *MonoBehaviour*

{

[SerializeField] *private* P2\_GSCONTROLLER gridshotClass;

[SerializeField] *private* P2\_LP linePathClass;

*private float* sens = 125f;

*private float* tempsense;

*private float* ADSmultiplier = 0.75f;

*private* bool paused;

*private* Transform playerbody;

[SerializeField] *private* GameObject player;

*private float* xrotation = 0f;

*private float* CamFOV = 60f;

*private float* tempFOV;

*void* Start()

{

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible = *true*;

tempsense = sens;

tempFOV = CamFOV;

playerbody = player.transform;

}

*// Update is called once per frame*

*void* Update()

{

*//changes which script to use the getter from depending on which task the user is doung*

*if*(SceneManager.GetActiveScene().buildIndex == 1)

{

paused = gridshotClass.get\_paused();

}

*else* if(SceneManager.GetActiveScene().buildIndex == 2)

{

paused = linePathClass.get\_paused();

}

*if* (!paused)*//If not paused*

{

*//Gets the X input and multiplies it by the sens for the speed and Time.deltaTime to make the camera movement frame independant*

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

*//Gets the X input and multiplies it by the sens for the speed and Time.deltaTime to make the camera movement frame independant*

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

*//Takes x rotation from mouseY because if it was equal, the rotation of the camera would be flipped compared to the Y input since it is rotating around the X axis*

xrotation -= mouseY;

*//Clamps the rotation of the camera around the X axis to make sure that the player cant look too far behind or too far down*

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

*//Rotates the camera using quaternion and euler angles around the X axis to allow the camera to look up and down*

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

*//Applies a rotation transformation on the up vector(y vector) of the playerbody to rotate it around the y axis to look left and right*

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

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible = *false*;

}

*if* (Input.GetMouseButtonDown(1))

{

CamFOV \*= 0.75f;

sens = sens \* ADSmultiplier;

}

*if* (Input.GetMouseButtonUp(1)) *// rohannn:)*

{

CamFOV /= 0.75f;

sens /= ADSmultiplier;

}

*if* (paused)

{

Cursor.lockState = CursorLockMode.None;

Cursor.visible = *true*;

}

*//changes the main camera's FOV every frame to update it when it changes*

Camera.main.fieldOfView = CamFOV;

}

*//SETTERS*

*public void* set\_FOV(*float* value)

{

*if* (value <= 110 && value >= 60)*//validating to make sure that the input given is within the allowed bounds*

{

CamFOV = Mathf.FloorToInt(value);

}

}

*public void* set\_Sens(*float* value)

{

sens = value;

}

*public void* set\_ADS(*float* value)

{

ADSmultiplier = value;

}

*//SETTERS*

*//GETTERS*

*public float* get\_sens()

{

*return* sens;

}

*public float* get\_fov()

{

*return* CamFOV;

}

*public float* get\_ADS()

{

*return* ADSmultiplier;

}

*//GETTERS*

}

### 5.6.5 Graph Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*using* UnityEngine.SceneManagement;

*public class* P2\_GRAPH : *MonoBehaviour*

{

[SerializeField] *private* P2\_GSCONTROLLER gridshotClass;

[SerializeField] *private* P2\_LP linePathClass;

*private* GameObject[] graphObjects;

*private* GameObject[] graphLines;

*private* GameObject graphParent;

*private* Vector2[] graphPoints;

*private float* pointDistance;

*private float* xValue;

*private* bool taskEnded;

*private* bool graphGenerated = *false*;

*private void* Update()

{

*if*(SceneManager.GetActiveScene().buildIndex == 1)

{

taskEnded = gridshotClass.get\_taskEnded();

}

*else* if(SceneManager.GetActiveScene().buildIndex == 2)

{

taskEnded = linePathClass.get\_taskEnded();

}

*if* (taskEnded && !graphGenerated)

{

*if* (SceneManager.GetActiveScene().buildIndex == 1)

{

graphPoints = gridshotClass.get\_graphPoints();

}

*else* if (SceneManager.GetActiveScene().buildIndex == 2)

{

graphPoints = linePathClass.get\_graphPoints().ToArray();

}

*//Initiates the arrays for storing the physical points and lines of the graph*

graphLines = *new* GameObject[graphPoints.Length];

graphObjects = *new* GameObject[graphPoints.Length];

*//runs the methods to generate the graph*

generateXPositions(200);

generateGraphPoints();

generateGraphLines();

*//sets graph generated to true to prevent the graph from generating again if not needed*

graphGenerated = *true*;

}

}

*private void* generateXPositions(*float* maxDistance)

{

*//calculates the distance between each point using the number of points and total graph distance*

pointDistance = maxDistance / graphPoints.Length;

*for* (*int* i = 0; i < graphPoints.Length; i++)

{

*//loops through the array and sets the x position to be equal to the calculated distance*

graphPoints[i].x = xValue;

xValue += pointDistance;

}

}

*private void* generateGraphPoints()

{

*for* (*int* i = 0; i < graphPoints.Length; i++)

{

*//loops through the list and sets the i index of each one to a sphere*

graphObjects[i] = GameObject.CreatePrimitive(PrimitiveType.Sphere);

*//sets the position to be equal to the same index in graph points*

graphObjects[i].transform.position = graphPoints[i];

*//makes the sphere 3x bigger*

graphObjects[i].transform.localScale = *new* Vector3(3f, 3f, 3f);

*//depending on which task it is, adds different scripts to display different values when hovering over the point*

*if* (SceneManager.GetActiveScene().buildIndex == 1)

{

graphObjects[i].AddComponent<graphPointHoverAccuracy>();

}

*else* if (SceneManager.GetActiveScene().buildIndex == 2)

{

graphObjects[i].AddComponent<graphPointHoverLinePath>();

}

}

}

*private void* generateGraphLines()

{

*for* (*int* i = 0; i < graphPoints.Length - 1; i++)

{

*//sets the current index to be a cube*

graphLines[i] = GameObject.CreatePrimitive(PrimitiveType.Cube);

*//sets the position to be in between current and next index*

graphLines[i].transform.position = (graphPoints[i] + graphPoints[i + 1]) / 2;

*//sets the size to be equal to the distance between the current and next point*

graphLines[i].transform.localScale = *new* Vector3(1, Vector2.Distance(graphPoints[i], graphPoints[i + 1]), 1);

*//rotates the cube to go from one point to the next*

graphLines[i].transform.rotation = Quaternion.FromToRotation(Vector3.up, graphPoints[i + 1] - graphPoints[i]);

}

}

}

### 5.6.6 Options Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*using* UnityEngine.UI;

*using* UnityEngine.SceneManagement;

*using* TMPro;

*public class* P2\_OPTIONS : *MonoBehaviour*

{

[SerializeField] *private* P2\_GSCONTROLLER gridshotClass;

[SerializeField] *private* P2\_LP linePathClass;

[SerializeField] *private* P2\_MOUSE mouseClass;

[SerializeField] *private* P2\_UI UIclass;

[SerializeField] *private* GameObject optionsMenu;

[SerializeField] *private* GameObject mainSettings;

[SerializeField] *private* GameObject controlsMenu;

[SerializeField] *private* GameObject audioMenu;

[SerializeField] *private* GameObject crosshairMenu;

[SerializeField] *private* GameObject preGameCanvas;

[SerializeField] *private* Canvas gameUI;

[SerializeField] *private* TextMeshProUGUI timerText;

[SerializeField] *private* Button controlsButton;

[SerializeField] *private* Button audioButton;

[SerializeField] *private* Button crosshairButton;

[SerializeField] *private* Button controlsToMainButton;

[SerializeField] *private* Button audioToMainButton;

[SerializeField] *private* Button crosshairToMainButton;

[SerializeField] TextMeshProUGUI sensValueText;

[SerializeField] TextMeshProUGUI FOVValueText;

[SerializeField] TextMeshProUGUI ADSValueText;

[SerializeField] *private* Slider fovSlider;

[SerializeField] *private* Slider sensSlider;

[SerializeField] *private* Slider ADSslider;

[SerializeField] *private* Slider dotSizeSlider;

[SerializeField] *private* Slider centerGapSlider;

[SerializeField] *private* Slider lineWidthSlider;

[SerializeField] *private* Slider lineHeightSlider;

[SerializeField] *private* TMP\_InputField fovInputField;

[SerializeField] *private* TMP\_InputField sensInputField;

[SerializeField] *private* TMP\_InputField ADSInputFeild;

[SerializeField] *private* TMP\_InputField dotSizeInputField;

[SerializeField] *private* TMP\_InputField centerGapInputField;

[SerializeField] *private* TMP\_InputField lineWidthInputField;

[SerializeField] *private* TMP\_InputField lineHeightInputField;

[SerializeField] *private* TextMeshProUGUI preGameTimerText;

*private* bool paused;

*private* bool preGame = *true*;

*private* bool preGameTimerBool;

*private float* preGameTimer = 3;

*private* bool taskEnded;

*private int* testint;

*private void* Start()

{

optionsMenu.SetActive(*false*);

gameUI.gameObject.SetActive(*false*);

preGameTimerText.text = preGameTimer.ToString();

}

*private void* Update()

{

*if*(SceneManager.GetActiveScene().buildIndex == 1)

{

gridshotClass.set\_preGame(preGame);

}

*else* if(SceneManager.GetActiveScene().buildIndex == 2)

{

}

*//updates the text onscreen to the value its based on every frame*

sensValueText.text = mouseClass.get\_sens().ToString();

FOVValueText.text = mouseClass.get\_fov().ToString();

ADSValueText.text = mouseClass.get\_ADS().ToString();

*//BUTTONCLICK LISTENERS*

*//every time the buttons are clicked //these methods are run*

controlsButton.onClick.AddListener(*delegate* { goToControlsFromMain(); });

audioButton.onClick.AddListener(*delegate* { goToAudioFromMain(); });

crosshairButton.onClick.AddListener(*delegate* { goToCrosshairFromMain(); });

controlsToMainButton.onClick.AddListener(*delegate* { goToMainFromControls(); });

audioToMainButton.onClick.AddListener(*delegate* { goToMainFromAudio(); });

crosshairToMainButton.onClick.AddListener(*delegate* { goToMainFromCrosshair(); });

*//BUTTONCLICK LISTENERS*

*//SLIDER LISTENERS*

*//when the values of the sliders change, it runs the setters to set the value, with the paramater passed through being the value of the slider*

fovSlider.onValueChanged.AddListener(*delegate* { mouseClass.set\_FOV(value:fovSlider.value); });

sensSlider.onValueChanged.AddListener(*delegate* { mouseClass.set\_Sens(value: sensSlider.value); });

ADSslider.onValueChanged.AddListener(*delegate* { mouseClass.set\_ADS(value: ADSslider.value); });

dotSizeSlider.onValueChanged.AddListener(*delegate* { UIclass.set\_dotSize(value: dotSizeSlider.value); });

centerGapSlider.onValueChanged.AddListener(*delegate* { UIclass.set\_centerGap(value: centerGapSlider.value); });

lineWidthSlider.onValueChanged.AddListener(*delegate* { UIclass.set\_lineWidth(value: lineWidthSlider.value); });

lineHeightSlider.onValueChanged.AddListener(*delegate* { UIclass.set\_lineHeight(value: lineHeightSlider.value); });

*//SLIDER LISTENERS*

*//INPUTFIELD LISTENERS*

*//when the user submits the inputfield it runs the input methods with the paramater passed through being the text in the inputfield*

fovInputField.onSubmit.AddListener(*delegate* { FOVInput(value: fovInputField.text); });

sensInputField.onSubmit.AddListener(*delegate* { SensInput(value: sensInputField.text); });

ADSInputFeild.onSubmit.AddListener(*delegate* { ADSInput(value: ADSInputFeild.text); });

dotSizeInputField.onSubmit.AddListener(*delegate* { dotSizeInput(value: dotSizeInputField.text); });

centerGapInputField.onSubmit.AddListener(*delegate* { centerGapInput(value: centerGapInputField.text); });

lineWidthInputField.onSubmit.AddListener(*delegate* { lineWidthInput(value: lineWidthInputField.text); });

lineHeightInputField.onSubmit.AddListener(*delegate* { lineHeightInput(value: lineHeightInputField.text); });

*//INPUTFIELD LISTENERS*

*if* (SceneManager.GetActiveScene().buildIndex == 1)

{

taskEnded = gridshotClass.get\_taskEnded();

}

*else* if(SceneManager.GetActiveScene().buildIndex == 2)

{

taskEnded = linePathClass.get\_taskEnded();

}

*if* (taskEnded)

{

*//if the task has ended, it makes the gameUI invisible*

gameUI.gameObject.SetActive(*false*);

paused = *false*;

}

*if* (Input.GetKeyDown(KeyCode.Escape))

{

*//shows the settings page when escape is clicked*

mainSettings.SetActive(*true*);

}

*if* (paused)

{

*//activates the optionsMenu if the game is paused*

optionsMenu.SetActive(*true*);

}

*else* if(!paused && !taskEnded)

{

optionsMenu.SetActive(*false*);

controlsMenu.SetActive(*false*);

audioMenu.SetActive(*false*);

crosshairMenu.SetActive(*false*);

gameUI.gameObject.SetActive(*true*);

timerText.gameObject.SetActive(*true*);

}

*//When escape is pressed, it performs the NOT operation on paused*

*if* (Input.GetKeyDown(KeyCode.Escape))

{

*if* (!taskEnded)

{

paused = !paused;

}

}

*//performs the actions when the game is in pregame state and the player isnt in the options menu*

*if* (preGame && !paused)

{

*if* (Input.GetMouseButtonDown(0))

{

preGameTimerBool = *true*;

}

*//starts the pregametimer countdown when true*

*if* (preGameTimerBool)

{

preGameTimer -= Time.deltaTime;

preGameTimerText.text = preGameTimer.ToString();

}

*//when the pregametimer reaches 0 or below*

*if* (preGameTimer <= 0)

{

*//sets pregame to false to start the game*

preGame = *false*;

gameUI.gameObject.SetActive(*true*);

preGameCanvas.SetActive(*false*);

}

}

}

*//these methods run when the input field is submitted*

*public void* FOVInput(*string* value)*//takes in a string parameter*

{

*//if there is only integers in the string passed through, pass it into the temp value*

*if* (*int*.TryParse(value, *out int* tempval))

{

*//if the tryparse returns true, use the setFOV method on the temp value*

mouseClass.set\_FOV(tempval);

}

}

*public void* SensInput(*string* value)

{

*if* (*int*.TryParse(value, *out int* tempval))

{

mouseClass.set\_Sens(tempval);

}

}

*public void* ADSInput(*string* value)

{

*if* (*int*.TryParse(value, *out int* tempval))

{

mouseClass.set\_ADS(tempval);

}

}

*public void* centerGapInput(*string* value)

{

*if* (*int*.TryParse(value, *out int* tempval))

{

UIclass.set\_centerGap(tempval);

}

}

*public void* dotSizeInput(*string* value)

{

*if* (*int*.TryParse(value, *out int* tempval))

{

UIclass.set\_dotSize(tempval);

}

}

*public void* lineWidthInput(*string* value)

{

*if* (*int*.TryParse(value, *out int* tempval))

{

UIclass.set\_lineWidth(tempval);

}

}

*public void* lineHeightInput(*string* value)

{

*if* (*int*.TryParse(value, *out int* tempval))

{

UIclass.set\_lineHeight(tempval);

}

}

*//GO TO MAIN*

*//These methods run when the buttons are clicked and they control which options screens to show*

*private void* goToMainFromControls()

{

controlsMenu.SetActive(*false*);

mainSettings.SetActive(*true*);

}

*private void* goToMainFromAudio()

{

audioMenu.SetActive(*false*);

mainSettings.SetActive(*true*);

}

*private void* goToMainFromCrosshair()

{

crosshairMenu.SetActive(*false*);

mainSettings.SetActive(*true*);

gameUI.gameObject.SetActive(*false*);

timerText.gameObject.SetActive(*true*);

}

*//GO TO MAIN*

*//GO TO SUBSETTINGS*

*private void* goToControlsFromMain()

{

mainSettings.SetActive(*false*);

controlsMenu.SetActive(*true*);

}

*private void* goToAudioFromMain()

{

mainSettings.SetActive(*false*);

audioMenu.SetActive(*true*);

}

*private void* goToCrosshairFromMain()

{

mainSettings.SetActive(*false*);

crosshairMenu.SetActive(*true*);

gameUI.gameObject.SetActive(*true*);

timerText.gameObject.SetActive(*false*);

}

*//GO TO SUBSETTINGS*

*//GETTERS*

*public bool* get\_paused()

{

*return* paused;

}

*public bool* get\_preGame()

{

*return* preGame;

}

*//GETTERS*

}

### 5.6.7 UIcontroller Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*using* UnityEngine.UI;

*using* TMPro;

*using* UnityEngine.SceneManagement;

*public class* P2\_UI : *MonoBehaviour*

{

Scene currentScene;

[SerializeField] *private* P2\_GSCONTROLLER gridshotClass;

[SerializeField] *private* P2\_LP linePathClass;

[SerializeField] *private* Camera playerCamera;

[SerializeField] *private* Camera graphCamera;

[SerializeField] *private* Canvas resultsPage;

[SerializeField] *private* Canvas graphCanvas;

[SerializeField] TextMeshProUGUI accuracyText;

[SerializeField] TextMeshProUGUI shotsTotalText;

[SerializeField] TextMeshProUGUI targetsHitText;

[SerializeField] TextMeshProUGUI errorsizeText;

[SerializeField] TextMeshProUGUI reactionTimeText;

[SerializeField] TextMeshProUGUI centerGapValueText;

[SerializeField] TextMeshProUGUI dotSizeValueText;

[SerializeField] TextMeshProUGUI lineWidthValueText;

[SerializeField] TextMeshProUGUI lineHeightValueTtext;

[SerializeField] *private* Button backToMenuButton;

[SerializeField] *private* Button goToGraphButton;

[SerializeField] *private* Button restartTaskButton;

[SerializeField] *private* Button goToResultsButton;

[SerializeField] *private* TextMeshProUGUI timerText;

*private* bool paused;

*private* bool taskEnded;

[SerializeField] *private* RectTransform crosshair;

[SerializeField] *private* RectTransform centerDot;

[SerializeField] *private* RectTransform topLine;

[SerializeField] *private* RectTransform bottomLine;

[SerializeField] *private* RectTransform leftLine;

[SerializeField] *private* RectTransform rightLine;

*private float* dotSize;

*private float* centerGap;

*private float* lineWidth;

*private float* lineHeight;

*private void* Start()

{

currentScene = SceneManager.GetActiveScene();

*//sets the default values for the crosshairs*

centerGap = 50;

lineHeight = 30;

lineWidth = 10;

dotSize = 23.5f;

}

*private void* Update()

{

*//changes which scripts to get the values from depending on which task the user is doing*

*if*(currentScene.buildIndex == 1)

{

timerText.text = Mathf.FloorToInt(gridshotClass.get\_TaskTime() - gridshotClass.get\_Timer()).ToString();

paused = gridshotClass.get\_paused();

taskEnded = gridshotClass.get\_taskEnded();

}

*else* if(currentScene.buildIndex == 2)

{

timerText.text = Mathf.FloorToInt(linePathClass.get\_TaskTime() - linePathClass.get\_Timer()).ToString();

paused = linePathClass.get\_paused();

taskEnded = linePathClass.get\_taskEnded();

}

*//updates the text every frame*

centerGapValueText.text = centerGap.ToString();

dotSizeValueText.text = dotSize.ToString();

lineWidthValueText.text = lineWidth.ToString();

lineHeightValueTtext.text = lineHeight.ToString();

*//updates the crosshair size every frame based on the values*

crosshair.sizeDelta = *new* Vector2(centerGap, centerGap);

centerDot.sizeDelta = *new* Vector2(dotSize, dotSize);

topLine.sizeDelta = *new* Vector2(lineWidth, lineHeight);

bottomLine.sizeDelta = *new* Vector2(lineWidth, lineHeight);

leftLine.sizeDelta = *new* Vector2(lineHeight, lineWidth);

rightLine.sizeDelta = *new* Vector2(lineHeight, lineWidth);

*//buttons run methods when clicked*

goToGraphButton.onClick.AddListener(*delegate* { goToGraphFromResults(); });

goToResultsButton.onClick.AddListener(*delegate* { goToResultsFromGraph(); });

restartTaskButton.onClick.AddListener(*delegate* { restartTask(); });

backToMenuButton.onClick.AddListener(*delegate* { goToMainMenuFromResults(); });

*if* (taskEnded)

{

*//gets the statistics from the main scripts when the task has ended*

*if* (currentScene.buildIndex == 1)

{

accuracyText.text = gridshotClass.get\_accuracy().ToString("F2");

shotsTotalText.text = gridshotClass.get\_shotsTotal().ToString();

targetsHitText.text = gridshotClass.get\_targetsHit().ToString();

errorsizeText.text = gridshotClass.get\_errorSize().ToString("F2");

reactionTimeText.text = gridshotClass.get\_reactionTime().ToString();

}

*else* if (currentScene.buildIndex == 2)

{

accuracyText.text = linePathClass.get\_avgPercent().ToString("F2");

shotsTotalText.text = linePathClass.get\_shotsTotal().ToString();

targetsHitText.text = linePathClass.get\_targetsHit().ToString();

errorsizeText.text = linePathClass.get\_errorSize().ToString("F2");

reactionTimeText.text = linePathClass.get\_reactionTime().ToString();

}

}

}

*public void* goToGraphFromResults()

{

resultsPage.gameObject.SetActive(*false*);

graphCanvas.gameObject.SetActive(*true*);

}

*public void* goToMainMenuFromResults()

{

SceneManager.LoadScene(sceneBuildIndex: 0);

}

*public void* restartTask()

{

SceneManager.LoadScene(sceneBuildIndex: currentScene.buildIndex);

}

*public void* goToResultsFromGraph()

{

resultsPage.gameObject.SetActive(*true*);

graphCanvas.gameObject.SetActive(*false*);

}

*//SETTERS*

*public void* set\_dotSize(*float* value)

{

dotSize = value;

}

*public void* set\_centerGap(*float* value)

{

centerGap = value;

}

*public void* set\_lineWidth(*float* value)

{

lineWidth = value;

}

*public void* set\_lineHeight(*float* value)

{

lineHeight = value;

}

*//SETTERS*

}

### 5.6.8 Target Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*public class* P2\_TARGET : *MonoBehaviour*

{

*//target health*

*private float* health = 10f;

*//called when target is shot*

*public void* takedamage(*float* damage)

{

*//reduces health by value passed through*

health -= damage;

*//if health reaches 0 call die method*

*if* (health <= 0)

{

Die();

}

*//destroys the target when called*

*void* Die()

{

Destroy(gameObject);

}

}

}

### 5.6.9 MovementController Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*public class* P2\_MOVE : *MonoBehaviour*

{

*private* bool paused;

*//Taken from brackeys tutorial on fps character controller https://www.youtube.com/watch?v=\_QajrabyTJc*

*//allows the referencing of the character controller in the code*

[SerializeField] *private* CharacterController controller;

*//How fast the player wants to move*

*private float* speed = 12f;

*//Standard value for gravity to be used in suvat and jumping equations*

[SerializeField] *private float* gravity = -19.62f;

*//The value for how high the player wants to jump*

*private float* jumpheight = 3.0f;

*//Makes a vectory3 called velocity to allow the change of the players velocity in both the jumping calculations and the gravity calculations*

*private* Vector3 velocity;

*//This transform(position) is placed at the bottom of the player mesh and is used to check wether the player is touching the ground or not*

[SerializeField] *private* Transform groundcheck;

*//Distance the player is from the ground when touching it*

*private float* groundDistance = 0.4f;

*//this layermask allows the player and the ground to be on the same layer to make sure that the code works if the player is touching a ground object*

[SerializeField] *private* LayerMask groundMask;

*//Either true or false on wether the player is touching the ground based on certain parameters*

*private* bool isgrounded;

*void* Update()

{

*if* (Input.GetKeyDown(KeyCode.Escape))

{

paused = !paused;

}

*//creates a tiny sphere at the bottom of the player and checks if anything is colliding with the ground mask and changes the bool "isgrounded" accordingly*

isgrounded = Physics.CheckSphere(groundcheck.position, groundDistance, groundMask);

*//Assigns a value to x and z based on the vertical and horizontal inputs from the player to be used in the movement code*

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

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

*//if the player is grounded AND the player has no Y velocity, the y velocity of the player is removed. this is to make sure that the gravity pull is not being increased*

*//while the player is grounded so that when the player is grounded for a long time, the y velocity resets to make sure that gravity acts normal on the next fall*

*if* (isgrounded && velocity.y < 0)

{

velocity.y = -2f;

}

*//takes the directions that the player is facing and multiplies it by it by the value taken from the player input and places it into a vector3(x,y,z)*

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

*//When the left shift button is pressed, it multiplies the movespeed by 2 to allow the player to sprint*

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

{

speed = (speed \* 2);

}

*//When the shift button is released, the movespeed is divided by 2 to stop the player from sprinting when the shift button is not pressed*

*if* (Input.GetKeyUp(KeyCode.LeftShift))

{

speed = (speed / 2);

}

*//Uses the Move function with the move vector3 and mutliplies it by the player speed and time to allow the player to move*

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

*//uses the physics equation v = sqrt(h \* -2 \* g) to calculate the amount of velocity needed to jump a certain height*

*if* (Input.GetKeyDown(KeyCode.Space) && isgrounded)

{

velocity.y = Mathf.Sqrt(jumpheight \* -2f \* gravity);

}

*//Uses the suvat equation DeltaY = (1/2 \* g) \* sqr(t) to calculate the change in the y velocity of the player due to the gravitational pull*

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

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

}

}

### 5.6.10 DisplayGraphPointValue Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*using* TMPro;

*public class* DisplayGraphPointValue : *MonoBehaviour*

{

*//This script was adapted from a code monkey tutorial on making tooltips*

*//https://www.youtube.com/watch?v=YUIohCXt\_pc*

[SerializeField] *private* RectTransform canvasRect;

*private* TextMeshProUGUI valueText;

*private* RectTransform backgroundRect;

*private* RectTransform rect;

*private string* timeValue;

*//We can get this value based on the build index of the scene that we are currently in*

*//E.g if the build index == index(gridshot 3x3) then we can set this value to be 30*

*private float* taskTime = 5;

[SerializeField] *private* Camera graphCamera;

*private* bool hittingPoint;

*private void* Update()

{

*//Sets the position of the value to be on the mouse position (divided by scale of canvas to make sure it stays constant even through resoluion change)*

Vector2 anchoredPosition = Input.mousePosition / canvasRect.localScale.x;

*//Makes sure the tooltip doesnt go off the screen*

*if* (anchoredPosition.x + backgroundRect.rect.width > canvasRect.rect.width)

{

anchoredPosition.x = canvasRect.rect.width - backgroundRect.rect.width;

}

*if* (anchoredPosition.y + backgroundRect.rect.height > canvasRect.rect.height)

{

anchoredPosition.y = canvasRect.rect.height - backgroundRect.rect.height;

}

rect.anchoredPosition = anchoredPosition;

RaycastHit hit;

*//Shoots out a raycast from the mouse position from the graph camera*

Ray ray = graphCamera.ScreenPointToRay(Input.mousePosition);

*//returns true if the raycast is hitting an object and the object has the "graphPointHover" script*

*//The only objects that contain that script are the graph points generated in the script "Gridhsot.cs"*

*if* (Physics.Raycast(ray, *out* hit) && hit.transform.GetComponent<graphPointHoverAccuracy>())

{

*//Sets the A(accuracy) to the Y value since the Y value of the point is the accuracy*

*//Sets the T(Time) to the X position since the X position of the point is the Time*

ShowValue("Accuracy = " + (Mathf.Round(hit.transform.position.y \* 100) / 100) + " " + "Time = " + (((hit.transform.position.x / 200) \* taskTime) + 1));

*//Since in the Gridshot script, we divide 180 (which is the target length of the graph) by the number of seconds the task is (TaskTime)*

*//That means that we dont get an accurate reading of the time since the x position of the graph points is no longer the seconds. So to fix that issue we just reverse the*

*//Operation that is performed on the x position by /180 then \* by task time to get the actual index of the point then we add 1 to compensate for the array starting at index 0*

*//(since the player cant hit something on the 0 seconds so it makes sense to start from the first seconds instead)*

}*if* (Physics.Raycast(ray, *out* hit) && hit.transform.GetComponent<graphPointHoverLinePath>())

{

*//Same explanation for Accuracy but replace with reaction time instead of accuracy*

ShowValue("HT = " + Mathf.Round(hit.transform.position.y \* 10) + "% " + "Time = " + (((hit.transform.position.x / 200) \* taskTime) + 1));

}

}

*private void* Awake()

{

backgroundRect = transform.Find("background").GetComponent<RectTransform>();

valueText = transform.Find("value").GetComponent<TextMeshProUGUI>();

rect = transform.GetComponent<RectTransform>();

HideValue();

}

*private void* ShowValue(*string* value)

{

*//Sets the text to be the value passed through as a parameter*

valueText.SetText(value);

*//Function to force regeneration of the mesh before its normal process time*

valueText.ForceMeshUpdate();

*//Sets the gap between the text and edge of rectangle to be 8 by 8*

Vector2 textGap = *new* Vector2(8, 8);

*//Sets the size of the text*

Vector2 valueTextSize = valueText.GetRenderedValues(*true*);

*//Sets the size of the rectangle to be the size of the text + the textgap to make sure the text isnt hugging the edges*

backgroundRect.sizeDelta = valueTextSize + textGap;

}

*//Hides the text*

*private void* HideValue()

{

gameObject.SetActive(*false*);

}

}

### 5.6.11 MainMenu Code

*using* System.Collections;

*using* System.Collections.Generic;

*using* UnityEngine;

*using* UnityEngine.SceneManagement;

*using* UnityEngine.UI;

*public class* P2\_MAIN : *MonoBehaviour*

{

[SerializeField] *private* Button gridshotButton;

[SerializeField] *private* Button linePathButton;

*private void* Update()

{

gridshotButton.onClick.AddListener(*delegate* { goToGridshot(); });

linePathButton.onClick.AddListener(*delegate* { goToLinePath(); });

}

*public void* goToGridshot()

{

SceneManager.LoadScene(sceneBuildIndex: 1);

}

*public void* goToLinePath()

{

SceneManager.LoadScene(sceneBuildIndex: 2);

}

}

## 5.7 Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Expected result | Actual result | Proof | Works as intended? | Resolved? |
| Timers work | The timers both pregame and midgame countdown by 1 every second and when they reach 0 either the game will start or the game will end | Both timers count down by 1 evey second and when the timers reach 0 the correct process happens |  | Yes – as you can see in the proof imags, the timer on the top image is 7 seconds and on the bottom is 0 seconds meaning the timer has counted down | No bug to resolve |
| Game pauses when options open | The timers will stop, the user should not be able to move the player character and they will not be able to shoot | Both the pregame and midgame timers do stop when the game is paused, the player cannot move or shoot | Hard to prove using screenshots | Mostly – while the timers and player character are stopped like intended, there was an oversight where the accuracy graph calculations still kept happening even while paused which will affect the statistics of the task | Yes – I changed the code so that the accuracy calculations no longer happen when the game is paused and resume when the game is unpaused |
| Gridshot accuracy graph correctly generated and displays statistics | The y position of the graph points should display the value of accuracy at that second | The graph works correctly as intended and each graph point displays the correct values |  | Yes - as you can see in the proof images, the one showing the results page displays that I had 100% accuracy throghout the entire task and this is reflected in the graph where all of the values are at the same y position and show 100% accuracy | No bugs to resolve |
| Line path hover time graph correctly generated and displays statistics | The y position of the graph point displays the average time spent hovering over the target line for that target | The graph works correctly as intended and each graph point displays the correct values |  | yes – as you can see on the graph, the first point shows that the value is 76% which is reflected in the console below where I printed all of the values being passed into the graph where the first value also shows 76%, furthermore you can see that the next value is 23% which is shown in the graph where it is much lower than the first graph point since it’s a lower value. | No bugs to resolve |
| Crosshair changes in options menu | All aspects of the crosshair change using the options menu and stays that way after resuming the game | The crosshair changes correctly based on player input using both the slider and the input field |  | Yes – as you can see in the proof images, the one above shows the crosshair in the default sate where the botton one shows me changing all of the values to the max value in the slider and the entire crosshair changes meaning the system works as intended | No bugs to resolve |
| Statistics correctly displayed in the results page | All of the statistics calculated are displayed on the results page after the task ends | All of the statistics set to display are correctly shown on the results page |  | Yes – as you can see in the proof image, the results that are displayed on the results page match exactly the raw statistics that were printed into the console straight after the calculation using the raw player data | No bugs to resolve |
| Gridshot target generation | Targets spawn correctly in a 3x3 grid in random positions and no targets spawning inside eachother | The targets spawn in a 3x3 grid and no targets spawn inside eachother |  | Partially – even though the system works as initially intended, there was a huge oversight which is that I didn’t consider the possibility of a target spawning in the same position as the one that just got destroyed which is not how the system is intended to work since the target must spawn in a different position each time to make it random for the player. | Yes – to fix this issue, I added another array called “justDestroyed” which stores the positions of the targets that just got destroyed. This means that the only way for a spawn position to be available is if there isnt currently a target in that position and if there wasn’t a target in there within the last 3 spawns meaning no targets spawn inside eachother and no targets spawn in the same place it just got destroyed. |
| LinePath target and line generation | Targets spawning in random positions within a predeterminded boundary and lines spawning from the last target to the next target | The targets do spawn in random positons within the boundary and the target lines spawn as normal |  | Yes – the target and line generation is working perfectly fine with no issues | No bugs to resolve |

## 5.9 Remedial Action

Oversight 1 – In the methods that handle the inputs from the input fields, I used the C# method int.TryParse which means that the system does still work, however, I forgot to take into account that the values the player are inputting are meant to be floats since that’s what the sensitivity is meaning the player cannot input a float since I used int instead of float.tryParse.

Bug fix 1 – the pre-game timer shows when in the crosshair menu before the game starts so in order to fix this bug I just have to make the pre-game timer invisible when in the options menu.

-which test was failed

-the before and after code

-how and why the fix worked

-if there is no test failed, add the test into the table

Bug fix 2 – when the task is in pregame state, the pre-game timer is shown behind the crosshair, making it less visible to the user. Also the pre-game timer is not rounded making it too long and visually unpleasing. To fix these bugs, I will make the crosshair invisible when the task is in pregame state and only visible in the options menu when in pregame and will also round the pre-game timer to make it more visually pleasing for the user.

Oversight 2 – I realised upon testing the game that the framerate was too low for me personally to play on and it made me realise that different people would like to play on different frames per second than others since it Is all down to preference, also some people may not have machines powerful enough to run the game at the base framerate that I had set of 60fps. So to change this in the next prototype, I will add an input field in the controls menu for the player to set the target framerate that they would like. The default value will still be 60fps but can be changed at any time within the task.

# 6.0 Prototype 3

## 6.1 introduction

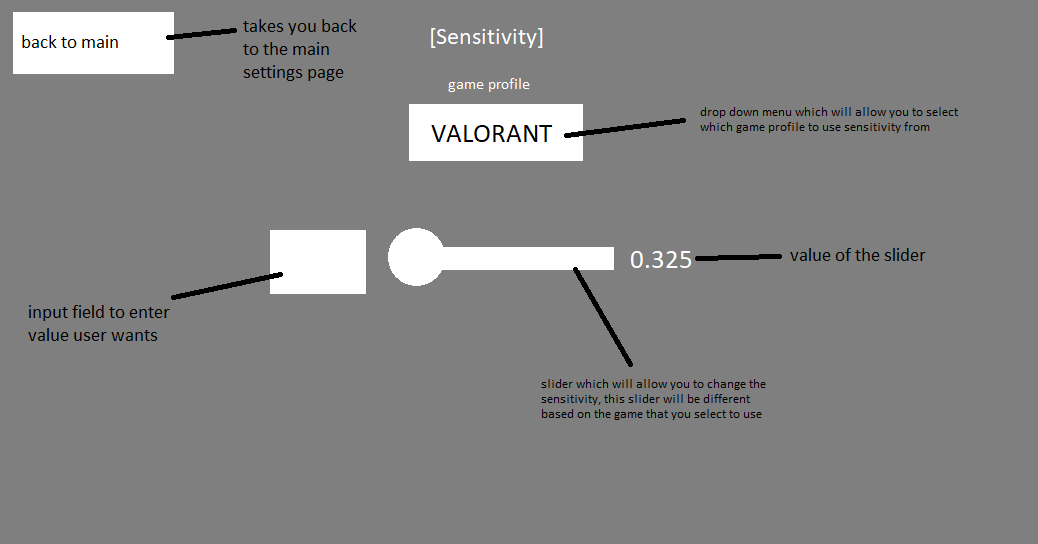
Features I will be adding or completing:

* Adding storing player data
* Completing game sensitivity
* Completing data and feedback
* Calculating an overall score value and implementing it into a graph

The aim of this prototype is to finish the features that I started in the other prototypes and to add the remaining features and to generally polish up the game both looks wise and system wise. The main features I will be focusing on in this prototype will be both the feedback part of data and feedback and the game sensitivity feature where I will work to implement conversion from one game sensitivity to another sensitivity. I will also be focusing slightly on storing player data such as their score and their settings to allow for an improved data and feedback feature and to improve the overall user experience by allowing the user to store their settings so when they load up the game then they have their settings saved from the last time they played.

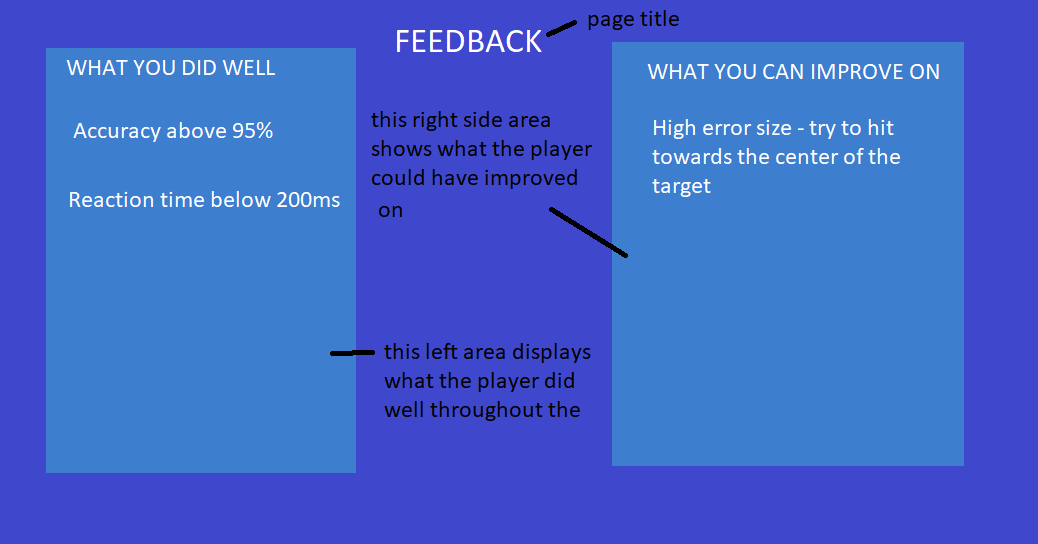
## 6.2 UI design

### 6.2.1 Game Sensitivity Screen



The way the user will interact with this screen is that they will use the mouse to move the cursor around the screen and the left mouse button to interact with the objects on screen. To use the dropdown menu, the user must click the dropdown menu to show the options then select the option they want from the available options. To use the slider they must click and drag the circle to the left to decrease the value and right to increase the value. To use the input field they must click to select, use the keyboard to enter the value then enter to submit the value.

### 6.2.2 Feedback screen



The way the user will interact with this screen is that they can use the mouse to move the cursor around the screen and the left mouse button to interact with the objects on the screen.

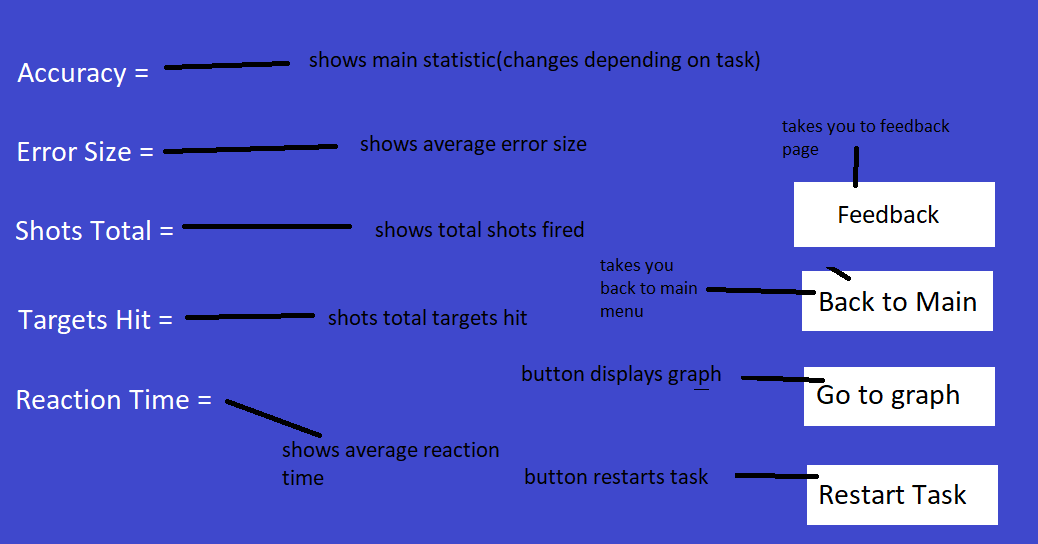
(add the back to results button)

### 6.2.3 Updated Main Settings Screen



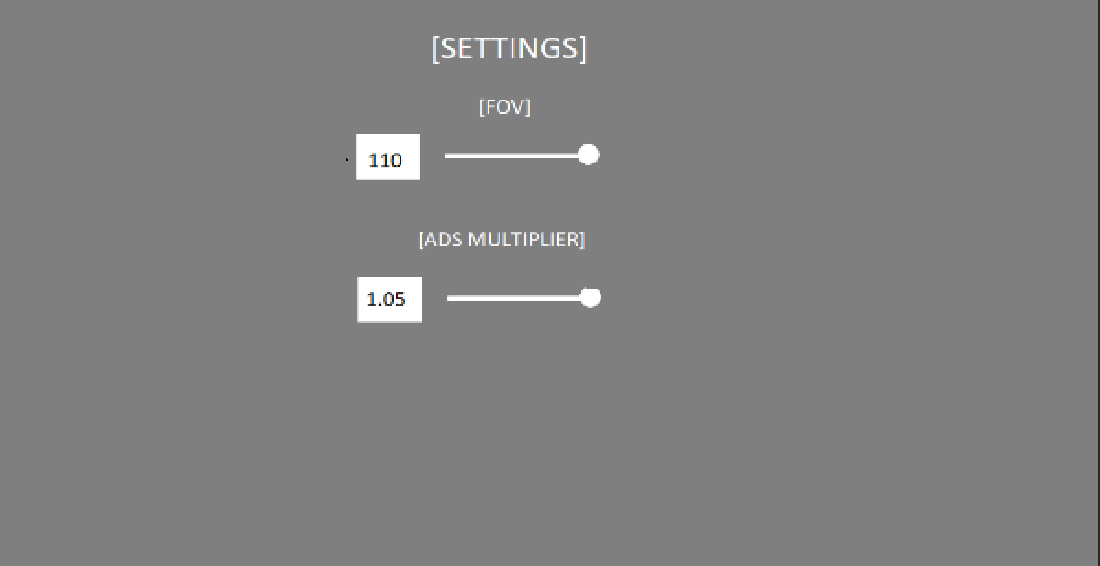
The way the user interacts with this screen is the exact same as how the user interacts in the 5.2.5 Main settings screen but with the addition of one more button to take the user to the game sensitivity page when clicked

### 6.2.3 Updated Results Screen



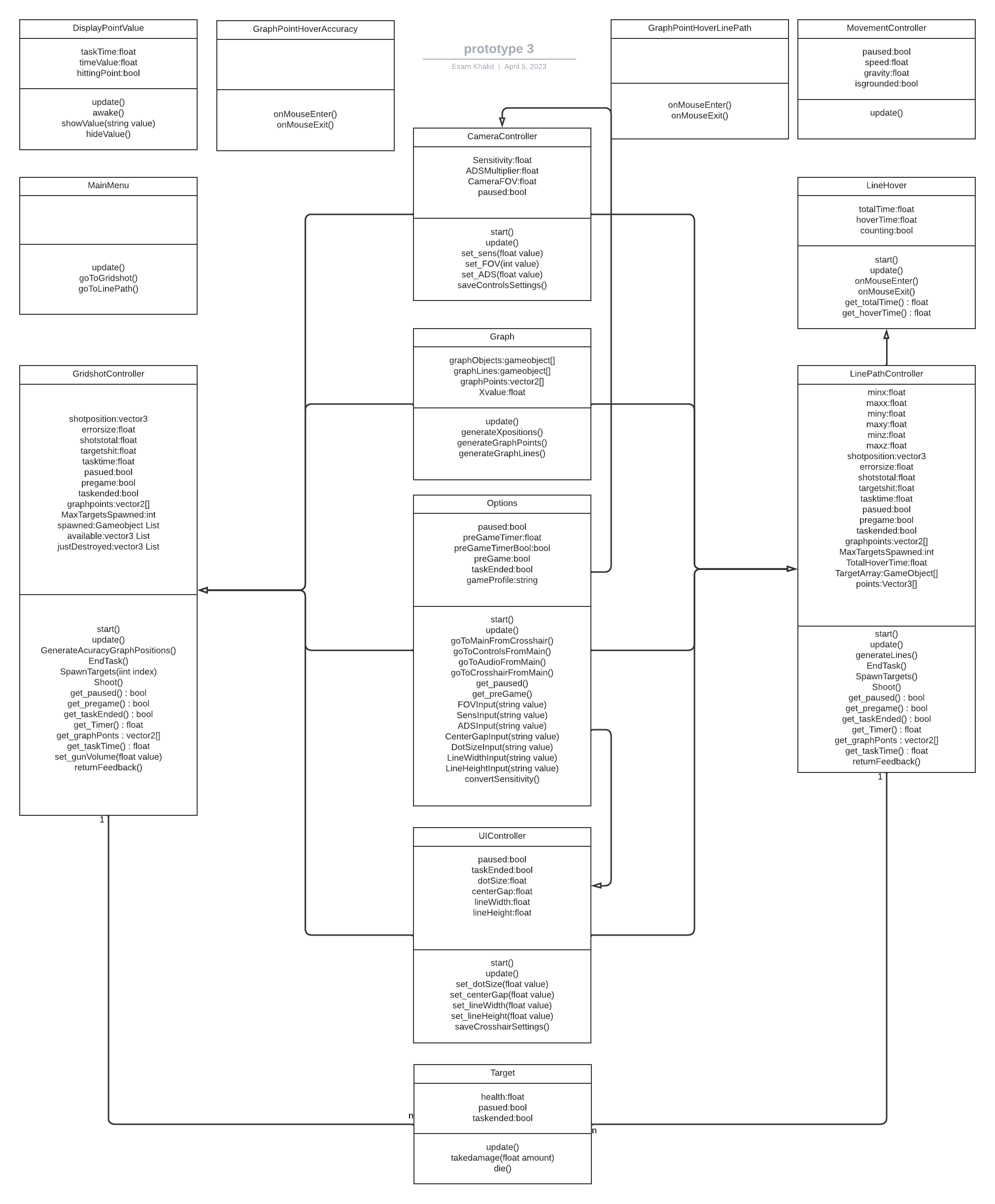
The way the user interacts with this screen is the exact same as how the user interacts in the 5.2.4 Results screen but with the addition of one more button that when clicked will take the user to the new feedback page.

### 6.2.4 Updated Controls Settings Screen



The way the user interacts with this screen is the exact same as 4.2.1 and 5.2.7 screens however the sensitivity slider and input field has now migrated to 6.2.1 Game Sensitivity Screen

## 6.3 Class diagram



### 6.3.1 Classes

All classes serve the exact same purpose as before with just a few added methods and attributes

### 6.3.2 Methods

All methods are the same as prototype 2 with a couple added methods

#### 6.3.2.1 convertSensitivity()

This method is used to convert between the game settings. The player will use a drop down menu to choose which game profile they want to have the sensitivity from then this method will be used to calculate the new raw sensitivity based on the game that is selected which will have a unique multiplier that will allow for the conversion between the sensitivity of this game to make it feel like the sensitivity of the other game.

#### 6.3.2.2 returnFeedback()

This method is used for returning feedback when the task has ended. It will compare the player data to the average, below average and above average data and will give feedback to the player based on how well they performed in different fields based on their statistics from the task.

#### 6.3.2.3 saveControlsSettings()

This method is used to save the controls settings of the player when the save button is clicked. This is done by using the playerPrefs system by unity to save the data into the registry of the computer the player is on.

#### 6.3.2.4 saveCrosshairSettings()

This method works exactly the same way as the saveControlsSettings() method except it saves the crosshair settings when the save button is pressed in the crosshair menu.

#### 6.3.2.5 goToGameSensFromMain()

#### 6.3.2.6 goToMainFromGameSens()

## 6.4 Data structure

All the attributes that were in prototype 2 have the same function as they did and no attributes have been removed and a couple attributes have been added

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Data Type | Justification | Validation? |
| gameProfile | integer | This integer will be an input using a drop down menu in the game sensitivity menu. This integer determines which game profile the user wants to use to convert the sensitivity from the base game to the game of their choice that available to choose from in the menu. The value of the integer will depend on which game profile is selected. | Yes – must be an integer that is selected using the dropdown menu |

## 6.5 Algorithm design

The only new key algorithm would be the sensitivity conversion algorithm, this is because it is the only new algorithm that is somewhat complex whereas the other algorithms are either a bunch of IF statements or something simple like storing and returning data.

### 6.5.1 Sensitivity conversion algorithm

For this algorithm to work correctly, I first need to find out the cm/360 for my game and one other game as a baseline to find out the multiplier to convert between the sensitivities. The game I have chosen is Valorant. Then to get the multiplier to the other games I can use the sensitivity converter from <https://gamingsmart.com/mouse-sensitivity-converter/> in order to convert from Valorant to the other games. This allows me to convert from my game to all the other games using the multipliers I calculated.

In order to get the first multiplier value between Valorant and my aim trainer, I put my aim trainer on 50 sensitivity and used a ruler to see how far the mouse moved in real life to get the in game cursor to move 360 degrees. At 50 sensitivity the mouse moved 26cm in real life.

Then to get a multiplier to convert to Valorant, I used the sensitivity converter above to see what sensitivity in Valorant has a 26cm/360 which was 0.628.

Finally to get the multiplier I divided 50 by 0.628 to get the multiplier which is 79.617

This process was repeated for the other games except I used Valorant as a baseline in the converter since my game is not on there, this makes no impact on the results of the multipliers since if 2 games at 2 different sensitivities have the same cm/360, they feel the same in both games so my game at 50 sensitivity feels the exact same as Valorant at 0.628 sensitivity.

All of these measurements and calculations are done with the assumption that the mouse used is at 800dpi.

|  |  |  |  |
| --- | --- | --- | --- |
| Game | Cm/360 | In-game sensitivity | Multiplier to get to my aim trainer |
| My aim trainer | 26 | 50 | 0 |
| Valorant | 26 | 0.628 | 79.617 |
| Overwatch 2 | 26 | 6.661 | 7.506 |
| Apex Legends | 26 | 1.998 | 25.025 |
| CS:GO | 26 | 1.998 | 25.025 |
| Rainbow 6 | 26 | 7.672 | 6.517 |

## 6.6 Implementation

## 6.7 Testing