A-Level Computer Science
OCR H446-03 Coursework

==========

Candidate Name: Ryan Chan
Project Name: Aim Trainer Game

==========

Contents

Analysis

Introduction of Aim Trainer

In 1987, the first networked FPS (First Person Shooter) was “made and published for the Atari ST by Hybrid Arts”. It was a revolutionary and new type of game, where it is known as a Pac-Man like maze game that also included bullets that were represented as bullets to kill other users in the game.

The aim of these early-developed games was to provide entertainment for a group of users from winning or killing other users, before they can do the same to you. From the start, an Aim Trainer would help practise shooting at other users or AI to be faster than other people for FPS-type games, but this genre has slowly developed into having a separate community and user-base because of its competitive and addictive gameplay to try and get a higher score or ranking.

In this project, I aim to add the simplistic design and features of games, inspired from a game called “Aim Lab”, where I would plan to innervate using my stakeholders opinions on which feature would be beneficial.

In the future of this project, I will plan to read articles consisting of opinions of what an Aim Trainer should have, depending on which type of game people would like to practise their aim into. This will help interest my target audience of the provided game and establish a long-term community.

Stakeholders

The general target audience for this game would be users of computers, who also have a keen interest in getting better at the range of motion control with the mouse and reaction

speed for their games that they play. This game will aim to also have a huge range from beginners who have never played a FPS/aiming-focused game before, to veterans who have over 1000 hours in shooting games.

Therefore, I would choose stakeholders that fit into 2 different demographics to get different views and opinions on what improvements to make and what features are valuable/useful to people with different aims to my game.

My stakeholders that I have close communications with are:

- Cavan (who will be my primary stakeholder)
- Adam (who will be my secondary stakeholder)

I have selected my main stakeholder as my friend called Cavan, because he is one of the most experienced in FPS/Aiming games that I know closely.

With over 3000+ hours, in different styles of FPS, he will be able to help with all the sections of the project. For example, for Analysis, he will be able to deliver some key requirements and useful information, such as keeping the style of the Aim Trainer basic to prevent new users from being overwhelmed with excess parts of the game that they would not use.

Additionally, during the testing phase, he can find more issues and problems than a regular casual stakeholder would be able to, due to his experience with similar games and help me discern features that might not be suitable or reasonable to put into this game. My stakeholder would also be able to provide more accurate and precise information as he has 200+ hours in “Aim Lab”, which this project is inspired by, so it will help give me confirmation on if it works for my particular game.

Cavan will make use of the finished product by allowing his other friends that struggle with basic aiming skills to hone it with this program, and help him personally get into the zone of aiming at fast speeds quickly. Not only is it suitable at improving my stakeholder’s friends aiming capabilities drastically and therefore helping them win together easier, it would also help strain the computer’s memory capacity on storing the variables in the scripts or database because it would account for any overflux of data that might not be accounted for due to the small time frame of this project.

I have selected my secondary stakeholder as my friend called Adam, because he is a casual gamer that likes to play a huge range of games, with data that helps add features from other game genres that will work alongside the main gameplay.

Furthermore, Adam brings a lot of mechanics and features from different games that one might not think to add to an Aiming Game. For example; having different ways of winning or keeping a high score instead of a number score would add some uniqueness to the game, adding a feature that speeds up the time the dots appear and disappear or adding more accessibility features such as using a draw pad instead of a mouse.

If Adam wanted to improve, he would make use of the proposed solution by having a quick 10 minute session everyday and steadily getting higher scores, but this game is especially

rewarding for people like Adam who play other games that require a lot of consistency. This is because my particular style of the game rewards both reaction time and accuracy, but it has a higher emphasis on if the user can endure the full 60 seconds while clicking on each cube with the most efficient travel path consistently. For example, in a fighting game, you would need to be able to achieve combos to win against the opponent. Since one combo attack would not end the opponent, you would need to hit multiple combos and be able to calculate the fastest way to do so.

Primary Research

Before designing my game, I need to recognise which parts I would need to specifically develop further and what features can make my game more sophisticated.

Initial-Project Interview

For my initial interview, I will ask a list of questions to my primary and secondary stakeholders, on how helpful why Aim Trainers are even used and if it can help make the game more enjoyable.

Interview questions:

1. **What makes a FPS game fun to play?**
2. **How long do you think you will use an Aim Trainer for?**
3. **Why did you use an Aim Trainer in the first place?**
4. **What aspects would you like or do you like in particular about an Aim Trainer?**
5. **Have you seen any drastic improvements in your aim for other games?**
6. **Do you have anything else you would like to add for now?**

Question 1 and 2 asks general questions that establish their connection with online gaming and identifies a reason why they would play FPS games in the first place.

Question 3 sets the overall premise of why each of my stakeholders individually chose to use an Aim Trainer, which would be vital in keeping the game focussed on the main reason it was made, instead of the sub-features.

Question 4 dives deeper into each person's individual opinion on what they actually want to achieve from playing the game. Since the perspectives are completely different, I would expect an obvious disparity between the two, so I will come to a conclusion by myself in the end.

Question 5 shows how useful playing my Aim Trainer would be in their other games, which would in return succeed in helping me achieve my own personal goal of the game.

Question 6 ends the interview by asking if there is any more important information they would like to respond with.

Cavan

1. “The development and competitiveness that is experienced throughout the game is why I think they are fun. Development happens during personal gameplay, where it achieves a goal by showing your skill difference through competing with others. This is what gives me a sense of thrill to FPS games, as getting better gives me a sense of accomplishment.”

2. “An Aim Trainer will be used alongside an FPS game as they have similar goals to get better in particular categories, which can always be improved upon by investing time and effort.”

3. “To give an edge over others, by improving at one aspect of the gameplay, specifically to others who have not used Aim Trainers before. This has been mentioned by a professional gamer before, who is strong in all aspects for these types of games that I currently do not have the capability to achieve at this moment.”

4. “With an Aim Trainer, it is important to have a different measurement of skills for each individual user. Having different levels of difficulty will help new users improve at their own personal rate. Also, rather than thinking about aiming the fastest, you could have a way to track accuracy of the shots will help bring the user to be more resourceful and in return, accurately secure a kill.”

5. “Yes, I have seen a clear improvement in both an FPS game and an Aim Trainer for me. Personally, my training in “Aim Labs” directly corresponded with the games I played, so I saw a massive improvement right from the start. As aim is one of the most important aspects, it has without a doubt improved consistently throughout my journey through FPS gaming.”

6. “No, I think I have said what I wanted to.”

Adam

1. “The ability to compete against others, as well as being able to increase your knowledge about the game and succeed with friends. The most fun FPS games that I play encourages the users to play smart and aim well with rewards in-game, whether it be ranks or special skins for just being better than others.”

2. ”As long as I was trying to play a game at a good standard for, if an Aim Trainer also helped me think about when and if I can hit a certain amount of targets in a small amount of time, it would tremendously help my game sense as well.”

3. “To improve at games that I enjoy playing.”

4. “The ability to see a quantitative score of some kind. This would act as a goal to try and improve through further training, which would make me want to come back for more. This is

helped by ranks in other games such as “KovaaK’s”, which help to give a more qualitative idea as to a user's skill level in other games.”

5. “Yes, while it is not a full aim crossover, I generally think improving aim in one game will usually help to improve aim in other games as well, however the mechanics and game sense may not transfer.”

6. “Definitely add a score or a way of measuring and comparing your score against yourself and others. A nice feature would be ways of training different types of aiming, such as flicking, tracking, tracing, etc.”

Analysis

Both of my stakeholders have experience with Aim Trainers, they have different ways of thinking when it comes to an Aim Trainer. Cavan is more focussed about the accuracy of each shot, while Adam is more interested about the rewards you receive from getting better at their aim. This is expected, but still very interesting about how the game is used by different people, where Cavan shows a more deep understanding on improving their skill as a talent.

On the other hand, Adam has produced some interesting points, where adding different styles of Aim Training would be a good idea and having a score counter allows them to easily see how much they have improved during the game. Cavan showed an interest in wanting a new type of Aim Training, where it consisted of high-level techniques of holding off for a major advantage, rather than just taking the small advantage (E.g. Killing one user would be worse than having the chance to kill multiple people at once, just by waiting for them to come into your sight.)

Inspired/Related Solutions

Aim Labs (Case study)

Overview

Aim Labs is one of the most popular and reliable games to increase in aim consistently. It uses many features, ranging from data shown visually on a graph to the reaction time/accuracy, where it factors into what multiplier would be added to the score that the user would get per time.

Features that are helpful

- Mentioned above, it uses a complex algorithm that consists of multiple databases, where I will use referential integrity to keep all the data functionally.
- A leaderboard/ranking section where it gets users to compete with each other, while increasing their skill level.
- Multiple game modes which will need to have separate databases that can also access each other to be able to calculate an overall score.
- Has the ability to notice weaknesses and strengths that the user has, and changes particular patterns to priorities weaknesses in their next playthrough.
- It is run on a platform called “Steam”, which has sections for community designed maps that allows new and different game modes to be made.
- Uses libraries on Unity that users automatically download when downloading the game.

Features to avoid

- The 3D aspect of the game is not feasible using coding languages/engines such as Python/Godot. Godot specialises in 2D art and functionality, so I will be taking into consideration how much I can do in the physical part of the game.
- The maps would need to be able to be played at all levels of experience, not just experts. This could mean when users are designing maps, there are some restrictions that might need to happen to prevent patterns from being too hard.
- Making each section of the game accessible no matter the ranking (tournaments etc.)

Kovaak's (Case study)

Overview

Kovaak's is known to be one of the first FPS aim trainer that got popularised with the exploding popularity of aim-intensive games such as Counter-Strike, Destiny 2 and more. It presented itself with a simple, but effective graphics that differentiated itself from other aim trainers because it drastically helped some users that met their limit before playing the game.

Features that are helpful

- Access to freely move around a selected map, with more freedom to control the character and gun you are aiming.
- Kovaak gives the opportunity to create your own separate maps that can be shared with other users, as people have different needs depending on which different game they come from.
- There are some movements that are more relevant to a particular game than others, such as a leap or a sprint, so people would be more likely to be able to play on community maps as people can customise their maps for themselves.
- There are over 1500 tasks that can be completed during the aim session, so there is always a small goal to pass each time you play the aim trainer, that will help you to keep on the correct path.
- Statistics are more focused about the difference from the close few runs, where it takes the data and compares it with the current run, rather than saying the difference in averages.

Features to avoid

- Having a more user-friendly experience, such as focusing on the usability and intuitiveness of the selecting game mode screens. Having this explains what specific

aspects of the aim you are currently training, rather than a rough overall aim train for a specific game.
- Don't overload the number of tasks that the user could do. Have some levels in place where users would need to do part A for example, to then do part B (which might be the same technique but harder), so users can be able to see which level of skill they are currently at and if they can skip a section out if they need to.
- Pay attention to how new users would act. If it becomes confusing, they wouldn't want to waste more time to find where to actually start. A simple reorganisation would help these users, or even have a tutorial that gives helpful tips/game modes for their particular game.

Essential Features

Since doing my research, I have now come to the conclusion of the essential features of what my game would need to require. I would need to have a main "select" screen, where I will have multiple options/game modes that the user can choose, depending on which part of their aim they would like to improve. This means that the game modes will currently be made by me, as this allows beginners to not get overwhelmed with where everything is and the game modes chosen will develop their basic skills, before moving onto more advanced skills that can be added as a feature, if I have the relevant time.

A database integrity would need to be established as well with each game mode, as this correlates with the leaderboard and storage of data, as it allows the user to know where their strengths and weaknesses are. Furthermore, I will need to produce an algorithm, where the speed and accuracy of hit objects will have a huge impact on the main score. This will help the users to have an objective to do when training, as it allows them to be competitive in the leaderboard. With having a leaderboard, I will need to have a side database to store and process personal information for people's profiles, so they are able to login and submit a score.

More essential features are identified and explained within the requirements.

Requirements

From all my past research, including my stakeholders opinions and by also having a more comfortable idea with what I want my game to do, there would be a set amount of requirements that need to be met for the game to be successful.

These include:

1. **Creating 3 Different and Unique game modes**

As this is an aim trainer, there will need to be different modes of aiming that the user will require training in different scenarios. As we have a limited amount of time, I think 3 majorly different game modes will keep the important competitive aspects of the game said by my

first interview with Cavan as it allow the users to train more and more efficiently because they would be able to compare themselves with others on what part of aiming they are best at. I have decided on “Gridshot”,”Motionshot” and “Snipershot”. I have chosen these from doing further research into why each of them are effective, and these are the best and unique options that I have seen that helps train the basic skills that a user would need to use, so these are quite important to succeed in.

2. The Functionality behind Gridshot

In this first game mode, it would test the user’s general skills based on a score based algorithm that increments every time the user had clicked on a target within a grid of space for a standard time frame of 60 seconds. The aim of this game mode would be to increase the user’s reaction time and accuracy because they will need to train to a high level to be able to play the other game modes comfortably. This would be a game mode primarily tailored towards my second stakeholder because he aims to get transferable skills that would help him in other games to develop his other skills in those games, as mentioned in the interview.

3. The Functionality behind Motionshot

In this second game mode, it would assess the user’s capability to track a target for 60 seconds - moving the crosshair to be inline with the cube that moves in random directions. This will be an algorithm that increasingly adds more score on, depending on how long the user was able to maintain the crosshair on the cube. This game mode would help my first stakeholder especially, as this is to help the advanced skill of flicking and tracking that only really improves expert user’s ability since they are able to master the general skills.

4. The Functionality behind Snipershot

In this last game mode, it has a high emphasis on improving the accuracy of the user. In the algorithm, it will gain more points than in the gridshot, but the cubes would be exponentially smaller and ranging in size. This would mean that the smaller cubes would give more score than the medium-big sized cubes, which results in this game mode being usable for either of my stakeholders, and any new users. From this, I can achieve both my first stakeholder’s aim for hard game modes and my second stakeholder’s aim to get more accurate in aiming.

5. Gaining Score depending on reaction time/skill

Rather than gaining a set amount of score from clicking each object, I will attempt to vary the score depending on a set criteria that will happen in a coded algorithm. The set criteria will depend on which game mode the user chooses, but the essentials ones will consist of reaction time/speed, accuracy and if they are hit in rapid succession (preferably no pauses between hits). This will be seen in the leaderboard section that will be saved eternally, unless someone beats their score. This was mentioned in both of the interviews, where it is an important requirement that would need to be in the game.

6. Database Integrity or Synergy

The database system will need to maintain the most up to date information for the user to know their own strengths and weaknesses. This is ranging from having a database for each different game mode to a database to store profiles and logins. However, these will all need to have a relationship with one another as they might need to retrieve and send information between them to either verify or see if there is a new high score on the leaderboard. I have found this out during solidifying my essential features choices.

7. Profiles

To make each user who plays this game unique, they will need to have their own specific profile to store data and accomplishments that they might want registered. This means that there will need to be a login/password database that stores all the secret information before even entering the game, as then each game will be stored, no matter if they performed well or badly. In return, the game will be able to calculate the strengths and weaknesses and try and perform patterns in those weaknesses to make the user better. I have come up with this idea from my case study of Aim Labs, where it has been shown how making a separate profile will help the user learn more about what they need to improve.

8. Settings

Each user will have different settings that they are comfortable with to aim with. If they are unable to gain such a setting, then the training won't be as easily transferable for the games they are looking to get better at, which will defeat the purpose of the game. This includes, mouse sensitivity, crosshair, colour just to name a few, but I will need to take into account everyone's preferences with either presets or small customization changes. This was mentioned during Adam's interview, where an aim crossover is vital in an aim trainer.

9. Low Spec System Requirements

As this game aims to help any user out with their aim skills. It will need to be a software that is able to be easily run, no matter the circumstances. However, I don't think this will be a problem as the game is likely to be made using godot, which doesn't strain the computer too much, even when running. I have noticed this from the Kovaak's case study, where I can see how simple but effective the graphics are, which I will also aim to achieve. Furthermore, the specific minimum hardware requirements would consist of having components such as; an average CPU that has higher clock speed than 3.0 GHz with 4 cores, a set of 8 gb DDR4 RAM, hard drive/ solid state drive, and importantly a GPU with at least a max memory size of over 1024 MB if the CPU does not come with integrated graphics.

10. Relevant Theme / User Friendly

This is a vital requirement that is constant throughout each of the case studies, which is using a set range of colours and making them work with each other to be attractive. Although, the colours would also need to be professional as it would make the user take the training seriously and make the game overall function better. Additionally, to make it user friendly, keeping the layout similar will allow new users to not be confused where everything is, which would keep the game organised, and adding new updates would not be a problem.

Success Criteria

These are the lists of points I will try and succeed in, as they all include essential aspects of the game that allows the game to run as intended and smoothly.

These include:

1. **Creating a Login Screen**

Having a login screen will allow the user to login to their own specific profile, which would be the first requirement to gain access to the game. As such, it links with requirement 7, mentioned above. This would be vital as it then sends them to the actual interactive part of the game of the home page, which will consist of buttons sending to other pages such as settings, leaderboard and training area.

2. **Score and Multiplier**

Creating the algorithm to track the multiplier will need to be completed successfully, as this is the only indicator in the game to where their skill level is at. This requirement links with 5, where the user can see an increase during their round of training.

3. **Keeping database accurate and up to date**

The final score after each of the individual rounds the user plays will be noted down and summarised to the user using charts and graphs to show weaknesses and strengths. This links to requirement 6 and 7, where it is vitally important for the user to see their improvement in a specific category. Therefore, charts and graphs show round data and average data to compare with so the user knows if they performed over-average or under-average.

4. **Successfully make the 3 game modes**

The 3 different game modes will consist of differently unique training that consists of “Snipershot”,”Motionshot” and “Gridshot”, which relates to requirement 1. In each of these game modes, I aim to create a modular design within my code that will make it easy to replicate in the other game modes. This is to speed up the process of writing each game mode as I can simply just copy over the basic template and iteratively test it with new code that aligns with the purpose of that specific game mode.

5. **The Algorithm behind Gridshot**

The algorithm behind gridshot should be a smooth experience for the user by adding a good collision between the crosshair and the cube. This is especially important in this game mode, as the user will be flicking rapidly, so the hit detection needs to register correctly so the algorithm can pick it up. This relates back to requirement 2.

6. **The Algorithm behind Motionshot**

Motionshot is a tricky concept because it requires the user to hover over a cube and track it to wherever the cube goes. This means that I will need to change the collision detection on this specific cube so any part of the cube, including the edges, are detected as collision. Additionally, to add more difficulty to this game mode, the cube can accelerate in the direction it intended to go towards which in return would grant the user with more score. This relates back to requirement 3.

7. **The Algorithm behind Snipershot**

The last game mode is a unique iteration of the gridshot cubes. More elements can be re-used in this application, with the difference of the range of random sizes for the cubes and the area of play being a lot larger than the previous one. Similar to Motionshot, the collision detection needs to be very accurate, as the collision of the smaller cubes will be really hard to hit fast, or I can make the collision of the smaller cubes slightly bigger to prevent any of the users struggling heavily on one of them. This relates back to requirement 4.

8. **Keeping the game easy to run**

This would be especially useful as anyone should be able to run the game, despite their system components, as it aims to improve someone's aim rather than the game sense which would only be attainable in that particular game which will definitely have higher system requirements. This means that my game will have more functionality, as it will be able to be played on a worser machine, school as library or school computers that will help them improve when they can't run the game they want to play. Seen in requirement 9.

9. **Usability and customisation**

This needs to be easy to navigate for the user as mentioned in requirement 10, everyone needs to have an easy experience when going through the program. This would also help the user to know where everything is for future use, and prevent them from clicking on something that they didn't want to do. Their settings also need to be easy to customise and useful for the user as mentioned in requirement 8, everyone has different crosshairs that they like to play with and having more than one style would give each user a preference and fairness.

Limitations

The first limitation I would have would be not being able to make the game into 3D. This is because doing 3D games would force me to use a different game engine from godot, as godot does not specialise in coding 3D games, which would make it a harder time to code

the game as intended and it might not turn out as well as if it was in 2D. Additionally, if I used a game engine such as Unity, I would come across the problem of needing to create or customise 3D models which would not be worth the effort for this project, as the functionality is the key point I am aiming to succeed in. As such, the player would not be able to move around or look around them as you could in 3D, which means that there would be a limit to how much they can train their aim for a 3D FPS game. Although this was a key feature that was mentioned in the interviews with my stakeholders, I do not think it would be much of an issue because it will still provide the same level of help as a 3D game, as aiming can be transferable between games which will overall make the user a better player.

My second limitation would be that the program would be program run, instead of a website run. This would result in the computer needing to have the specific program to be able to be run successfully, which would not make the game easily distributable. Furthermore, any databases and other code would need to be updated fast and organised neatly, so then it would not only make it less confusing for me to work with, but also help the user get the correct information (seen in my success criteria 3).

Computational Solutions

There will be many computational solutions that I will plan to include in my game. This will consist of software or hardware applications, such as abstraction, divide and conquer, and parallel processing will be a key aspect of how I will make my game successful.

With abstraction, this would be used during the creation of the 2D models (mentioned in limitations), where it simplifies the game down and allows me to add more essential features that I might not have been able to include if I did it in 3D. Additionally by using abstraction, it will also remove the unnecessary need of having the game look glamorous, which would be beneficial for computers with low specification components.

With divide and conquer, I will be able to sort out each game mode database data effectively using a binary search, as this would help the user get their results quick and accurate at the end of each round, as the computer would need to know if the round that they just played was worse than their average or better than their average. Thus, this can be solved by organising the data correctly (in order from smallest to largest) and seeing if the new data is showing improvements, or if they got their new personal high score. This would help give the user a short goal to achieve, so they will be able to compare with others on the leaderboard that my stakeholders had mentioned.

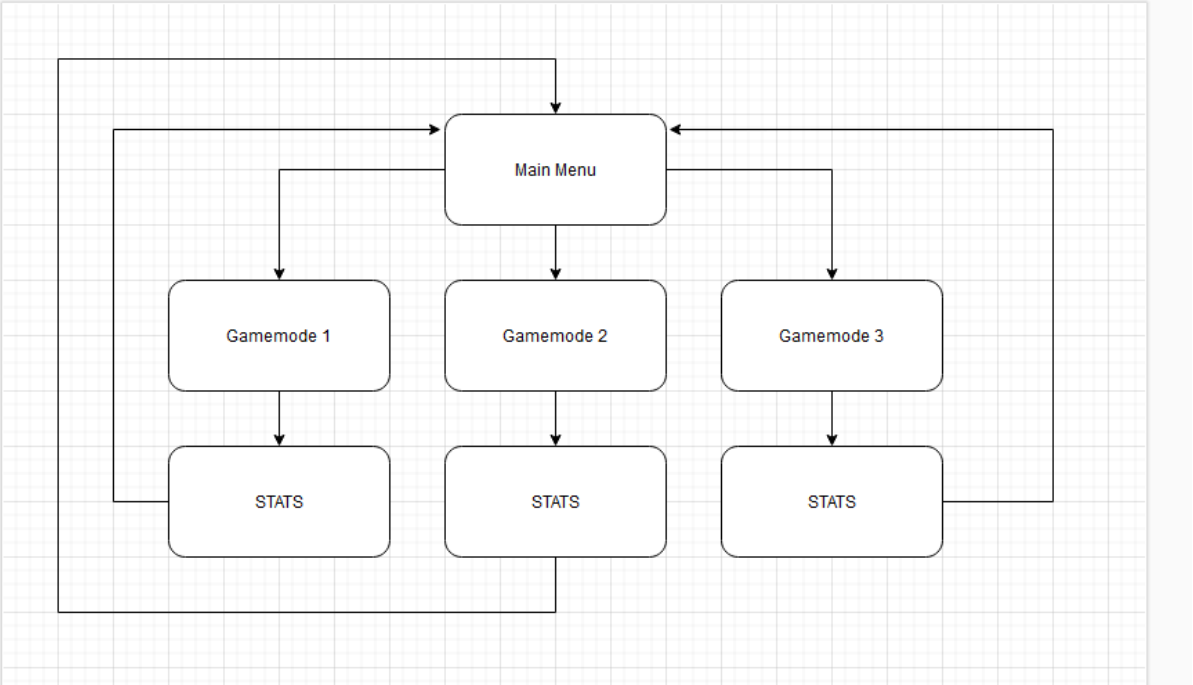
With parallel processing, it is a vital requirement when doing the accuracy and reaction time algorithm during the game mode. This is because one processor would need to focus on making sure the data is up to date on accuracy, whilst also having another process on reaction speed of the player. This would not be able to be concurrent, as each data point needs to be updated at the same exact time, so then there would not be any errors in mixing different data types, which would make the game unable to run. Furthermore, this will allow other independent cores to work on organising and other applications, so then stutters or freezes in the game will be uncommon.

Design

Project Structure

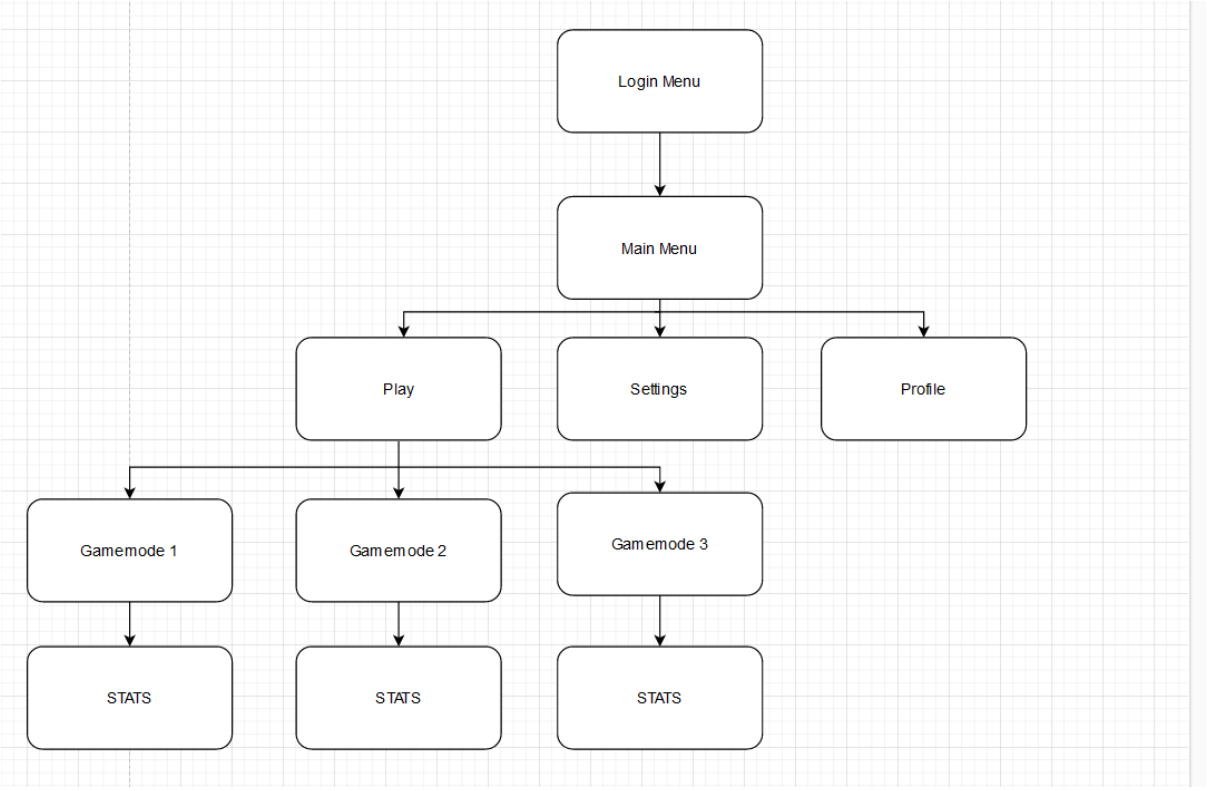
In each of my phases, I will plan to decompose the project into key parts through iterations using structured flow diagrams. Thus, this design will be split up into a project structure for the overview, and the rest allows me to show my main algorithms behind the project. This will allow me time to refine them, until I get onto development. Additionally, this will allow me to see each individual feature and meet with my stakeholders to see in which order and the importance of knowing which features I can complete in parallel or concurrently.

Iteration 1



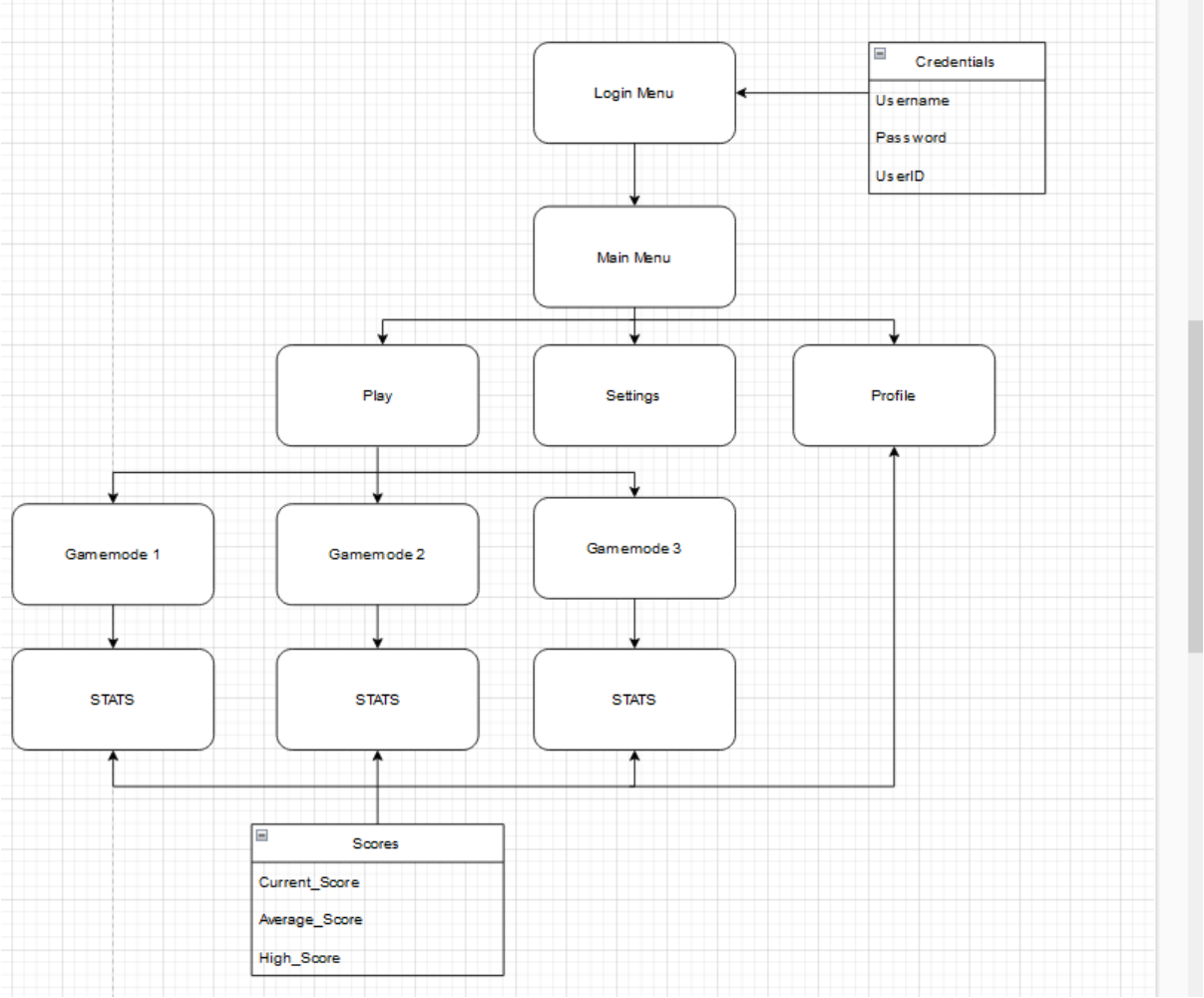
The first flowchart shows the initial structure of the project I intend to create. I have split them into the most important parts of the project, where I will further develop with the aid of the stakeholders when I have a better idea of where everything is. This was created to give me a basic understanding of which parts I will focus on the most, and as seen below, each game mode follows the same path. This results in reusable modular code after developing one game mode with a slight alteration, so I might not need to develop all of the game modes, as it won't showcase any unique or new features that I haven't already shown. Furthermore, this means that my first requirement might not necessarily need to happen, as it doesn't change the complexity of the project that much.

Iteration 2



After discussing my first iteration with my stakeholders, they wanted me to add a login screen in order to keep data on which user is currently logged in and so each score saves to their own unique database. For this, I included it as the first menu the user will see before they get access to the rest of the program for the system to know which personal database to store data within. Furthermore, I added a few more different options for the main menu, such as the “settings” and “profile”, where they will add more features for the user to utilise when playing the main game.

Iteration 3



In this iteration, I have included the databases that the information will be stored and calculated from. As a result, I currently only need 2 databases, 1 for the player credentials, and another one for storing the “score” data that the user will achieve, and then utilising that to show the users their “current_score”, “average score”, and “high_score”. Furthermore, I will make use of referential integrity within the databases, where each UserID will have a separate table within the score data database, so the scores get put into their corresponding table to make the calculations accurate. This will make it easier to see what I can do at the same time as the others.

Login Screen

Screen Design

Iteration 1

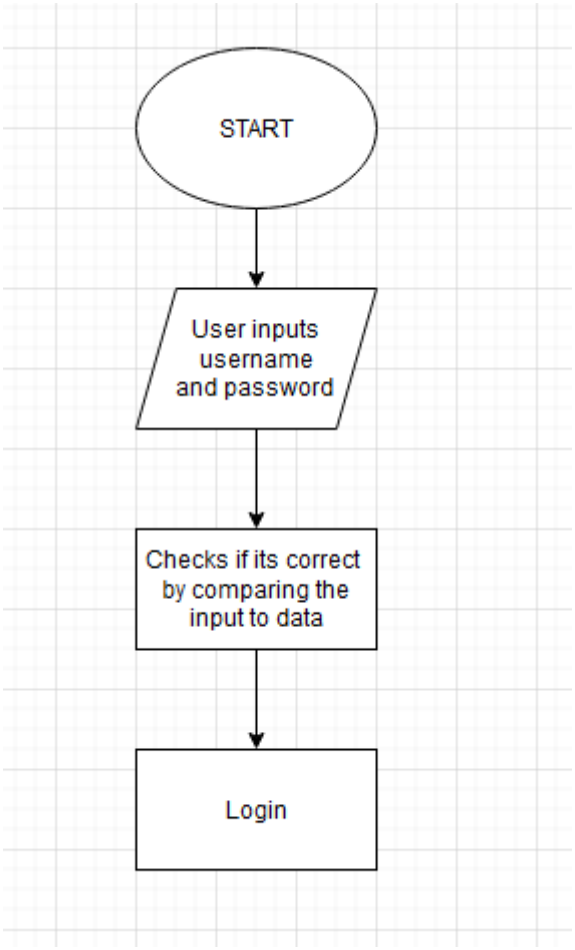
In this login screen, I have used a simple black and white colour scheme to try and first create the layout of the screen. The big input boxes allow the user to see what and which one to put the specific information in, and the logo on the right just to fill up the empty space. However, I feel like there obviously needs to be a vibrant colour scheme, and my stakeholders said that adding a login button would give the user control of when they want to login, or if they want to quit.

Iteration 2

After researching similar games online and through discussion with my stakeholders, I added the vibrant colour of red and a more prominent white area to stand out from the red background. This makes my 10th requirement done, as I intended to reuse these colours and keep it user-friendly. Now, there is also a login button so the user can have the option to choose when they want to login and a quit button is intuitively placed in the bottom right corner of the screen, as this location is where the least amount of clashing with other buttons/images that are in the scene. Additionally, this sets a common layout through the project, so the user will always know where to quit or back depending on what the button is doing and where they are in the project.

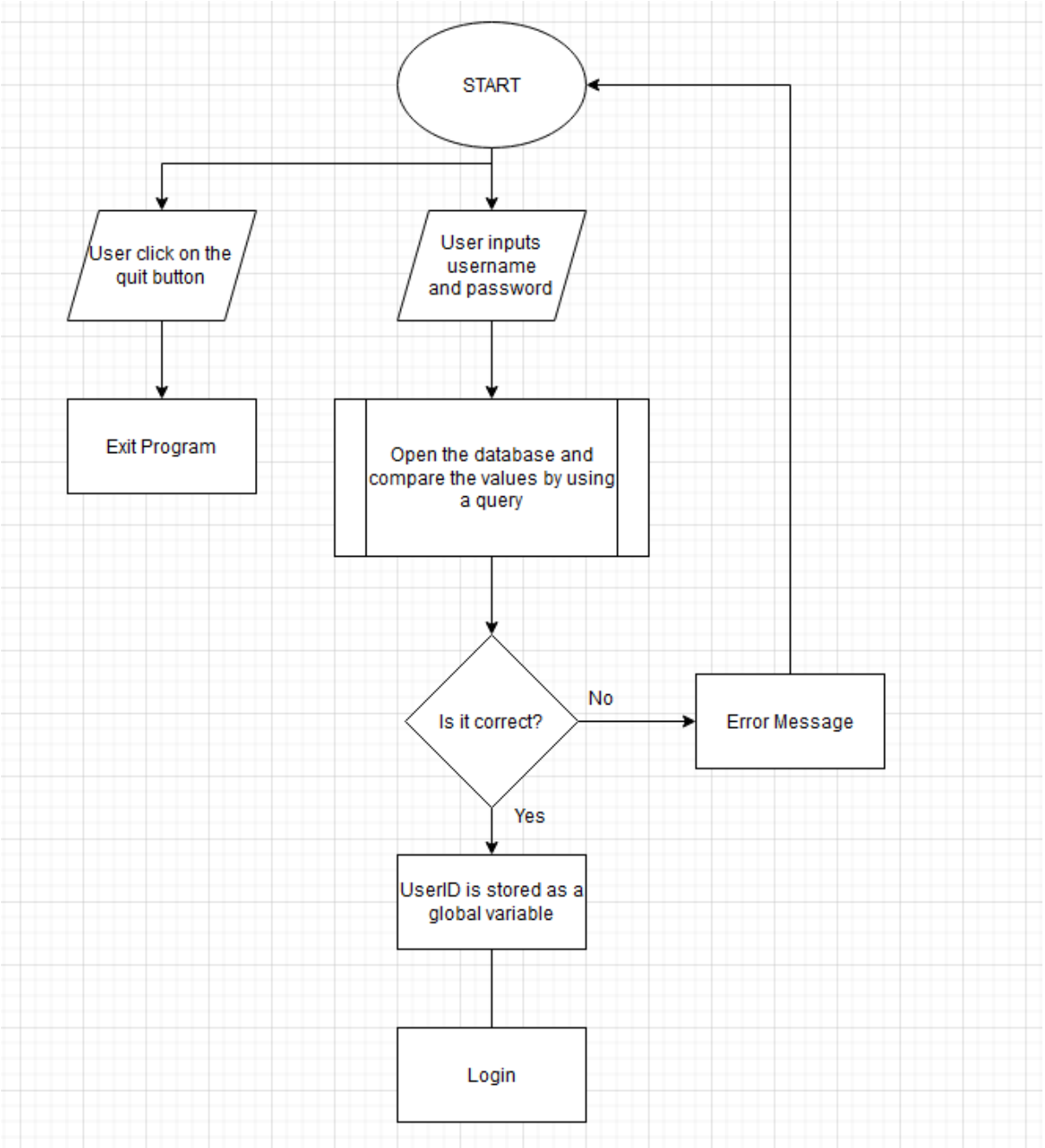
Flowcharts

Iteration 1



This is a basic representation of what the login screen will look like and function. By starting the game, the user will then be able to input their data that they already have, and a function built into a button will compare the data. However, I can show this in more detail with decision and process operators added to give my stakeholders what I want to achieve.

Iteration 2



This is the advanced structure of my login screen. The user inputs their data, and then the process operator sends it off to a function that opens the specific database and finds if the data is available (with set conditions). Then, depending on if the data is correct or not, it will proceed further into the project or go back to the start for the user to retry once again. In addition to this, I had forgotten about the quit button, so I added a 2nd feature for the user to input into the flowchart as seen above.

Pseudocode

Naming Conventions

I have decided from personal experience and use that the variables will be named differently depending on if there is more than one word.

If it's only one word, the variable has one capital at the start and the rest is lower case. E.g. var Game, var Timer

If it's more than one word, the variable is all lower case with the separation between the words replaced using an underscore. E.g var table_name

Iteration 1

```
# Player loads up the program by running
start the game
load login_screen

# Function

function ComparisonLogin():

    # set the user input data to variables
    variable Username = (user input data).text
    variable Password = (user input data).text
    variable table_name = "(table name)"


    open database
    database.query(select all from the table_name, where the username = Username or
where the password = Password)
    # Sees if the user input is valid or invalid
    IF query_result of username and password are found:
            login_screen.change_scene(“main_menu”) # changes the scene

    ELIF query_result of username found but password not found:
            return reveal.("password is incorrect") # error message

    ELIF query_result of password found but username not found:
            return reveal.("not found username") # error message

   ELIF query_result of password not found and username not found:
            return reveal.("not found user") # error message

   ELSE:
            return reveal.(“error”) # User should never see this
```

```
    ENDIF
    return close database

ENDFUNCTION

# Main Game

IF user clicks login_button:
    start ComparisonLogin()

ENDIF
IF user clicks quit_button:
   close game

ENDIF

END login_screen
```

This simple piece of pseudocode is the frame for the code that will be written for the login screen. First, it starts the game through user input, and then the game loads up the login_screen. If the user clicks on the login_button, the ComparisonLogin() function is called. The data inputted into the fields will be set to their respective variable, and then the database is queried with the variables set as the conditions, which changes the query result and if statement result that is sent back to the system. If the user clicks on the quit_button, the game simply closes.

Variable/Function	Data Type	Use	Justification	Validation
Username	String	This is a string that is taken from the user input. This will only be set when the login function is pressed, so there would not be a loop function checking if the user had inputted anything.	I am using this as within godot, there are nodes in which I can reference using their own feature called “singletons”. This makes it easy to store any values that the user has inputted into the nodes/text boxes that I will make.	This is something that the user can test when they add their login details in. Validation is received through an error message.
Password	String	This is a string that is taken from the user input. This will	I am using this as within godot, there are nodes in which I can	This is something that the user can test when they

		only be set when the login function is pressed, so there would not be a loop function checking if the user had inputted anything.	reference using their own feature called “singletons”. This makes it easy to store any values that the user has inputted into the nodes/text boxes that I will make.	add their login details in. Validation is received through an error message.
table_name	String	This string is already assigned the table_name of the UserID, and this is a private attribute as the player cannot change this value.	I have used this so I am able to use it within the query statement that I make during the function.	Not applicable for validation because it is not accessible for the user.
ComparisonLogin()		This function will be used to verify the data and see which error message to reveal, or if the information is correct to load the main menu.	This not only allows certain users which have a login to get into the game, but also saves each score to their respective user.	Not applicable for validation because it is not accessible for the user.

Test plan

Some of the tests below are designed to break my program, so I know what programs may occur if they ever happen.

Test Description	Test Data	Expected Outcome
Test to see if the login will accept a valid username but invalid password, and then clicking login	Correct username and random password that is not correct	reveal.("password is incorrect")
Test to see if the login will accept a valid password but invalid username, and then	Correct password and random username that is not correct	reveal.("not found username")

clicking login		
Test by Instantly clicking login_button	Clicking login_button	return reveal.("not found user")
Test with correct credentials	Correct username and password	load main_menu
Test by clicking quit_button	Clicking quit_button	close game

Main Menu Screen (+ Profile, Leaderboard, and Settings)

Screen Design

Iteration 1

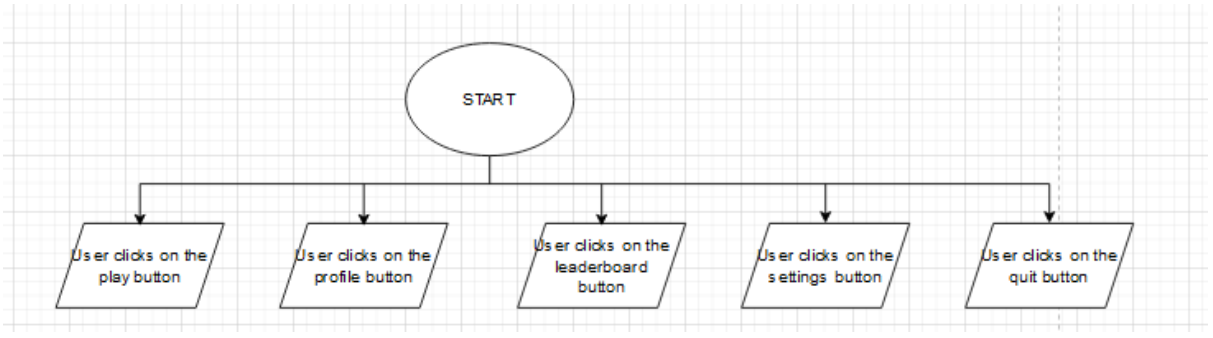
This is the plain black background I have created as an initial draft.

Iteration 2

I have changed this to keep consistent with my 10th requirement. The 4 buttons are big for the user to click and the logo is in the middle because it is the first thing a user sees when the scene has changed.

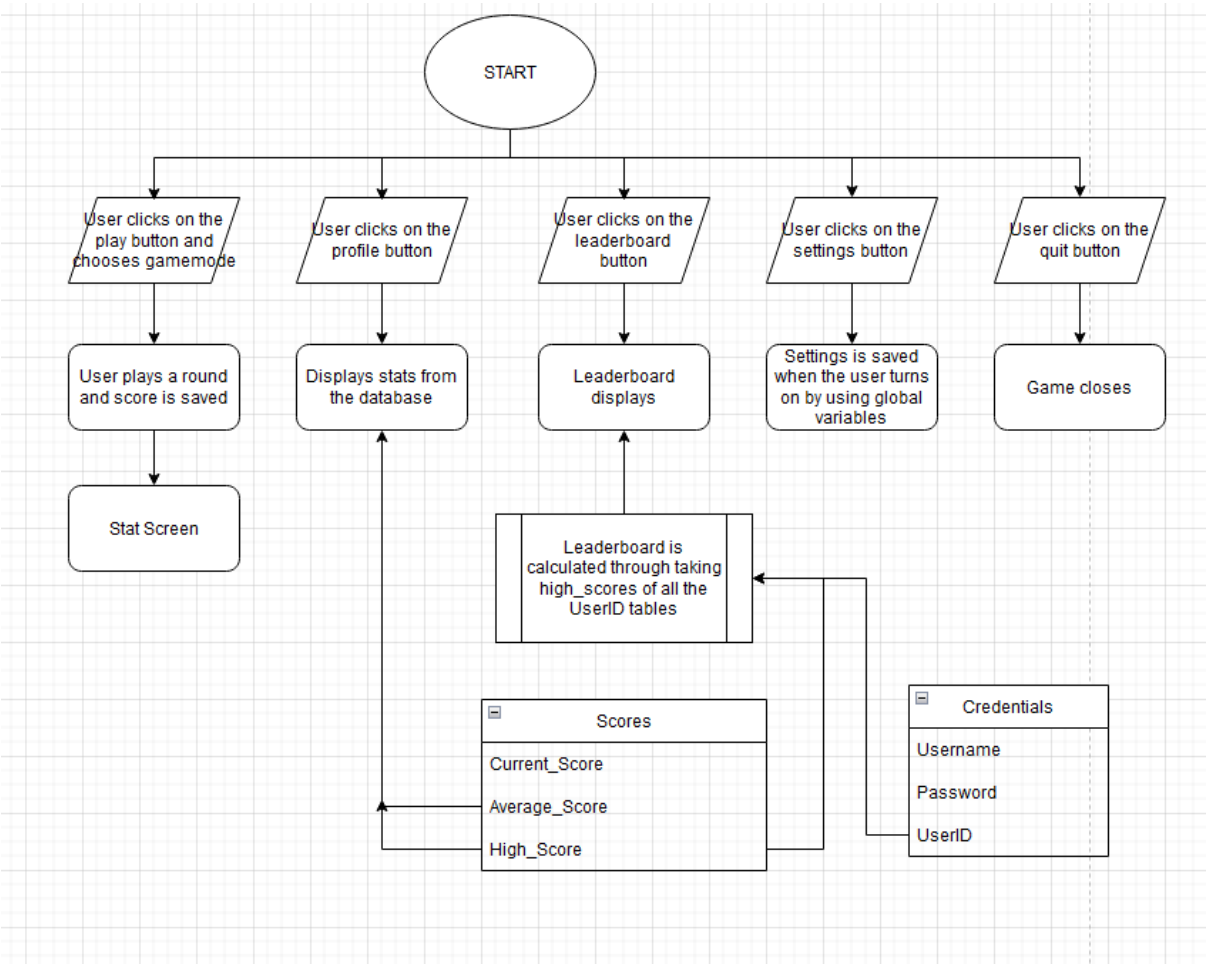
Flowcharts

Iteration 1



In this basic flowchart, I have simplified the scene through only what the user will do.

Iteration 2



By adding database synchronisation and system changes to the program, this more advanced and decomposed version of the flowchart explains what will happen in the sub-category scenes as I think it would be more time efficient to explain them with the main menu. This plays a key part in acquiring my 6th requirement of database synergy, as this will help provide those scenes with the correct information.

Pseudocode

Iteration 1

```
# Scene is loaded

load main_menu
preload global_variables # This preloads any global variables that are in other sections of code
```

```
Sqlite = preload("(SQLITE ADDON PATH)") # preload the sqlite addon so it is running before the code is ran
variable db # assigning database to a variable
variable db_path = "(path)" # getting the path to the database
variable table_name = "(table name)" # name of the table I want to utilise


# Functions

function Play_button_pressed():
# This section is expanded more on the (Game mode 1) section
    main_menu.change_scene("Game mode_1")

ENDFUNCTION

function Profile_button_pressed():
    main_menu.change_scene("Profile") # Change the scene to the profile scene
    # The average_score and high_score should be stored within a global variable before loading the scene as it is retrieved from a database
    SET node Average_score.text = global variable Average_score
    SET node High_score.text = global variable High_score
    IF back_button pressed: # there is always a back button for all these scenes
            Profile.change_scene("main_menu")
    ENDIF

ENDFUNCTION

function Leaderboard_button_pressed():
    main_menu.change_scene("Leaderboard")
    db.Sqlite.new() # new instance of the sqlite addon
    db.path = db_path # set the path of the database
    open db # open the database
    db.query(SORT SCORE IN DESCENDING ORDER FOR ALL USERS FROM table_name)
    variable i = 0
    for i in range(i,4):
    FROM UserID of db.query_result[i],
    SET node N(i+1).text = USERNAME AND SET node N(i+1)S.text = SCORE
    ENDFOR
    # For the non-loop version, it is shown below;
    # From UserID of the db.query_result[0], SET their node N1.text = Username AND SET node N1S.text = Score
    # From UserID of the db.query_result[1], SET their node N2.text = Username AND SET node N2S.text = Score
    # From UserID of the db.query_result[2], SET their node N3.text = Username AND SET node N3S.text = Score
    # From UserID of the db.query_result[3], SET their node N4.text = Username AND SET node N4S.text = Score
```

```
    # From UserID of the db.query_result[4], SET their variable N5.text = Username AND SET variable N5S.text = Score
    IF back_button pressed:
        Leaderboard.change_scene("main_menu")
    ENDIF

ENDFUNCTION

function Settings_button_pressed():
    main_menu.change_scene("Settings")
    IF any of Crosshair 1 - 6 are turned on: # this should be saved as a global variable so every time this scene is ran, it double checks the value
        global variable crosshair_1 - 6 respectively = true
    ELSE:
        global variable crosshair_1 - 6 respectively = false
    ENDIF
    IF back_button pressed:
        Settings.change_scene("main_menu")
    ENDIF

ENDFUNCTION

# Main Game

IF Play_button is pressed:
    Play_button_pressed()

ENDIF

IF Profile_button is pressed:
    Profile_button_pressed()

ENDIF

IF Leaderboard_button is pressed:
    Leaderboard_button_pressed()

ENDIF

IF Settings_button is pressed:
    Settings_button_pressed()

ENDIF
```

Within my leaderboard function, a new instance of a database query is created, so by sorting the scores in descending order throughout all the users, I repeatedly use a for loop to replace the text in the nodes (what the user can see), with the database output. In the

profile function, I simply set the node values to the global variables that are calculated within the database, and in the settings function, I need to make sure the global variables Crosshair_1 - 6 are saved and loaded every time the user clicks on it. (Requirement 7 and 8 are completed through this)

Variable/Function/Node	Data Type	Use	Justification
Global_variables (E.g. Current_Score, Average_Score, High_Score, Crosshairs_1 - 6)	Integer, Boolean	This will be a library/section of globally accessible code, which can be changed through new assignments.	This is so nodes and other variables in one scene can be assigned using these global variables, which can be used in another.
db, db_path, table_name	String	A standard practice within godot to do something with a database.	So the database will be connected and I can understand which and how the database is inputting and outputting results.
Average_score.text, High_Score.text	Integer	This is changing a visible object that the user can see, in this case, the text/string will change depending on what I assign it.	This is the easiest and most efficient way of changing a visible object's text to the correct information.
i	Integer	This is used as a counter/condition in the for loop, so the loop knows when to stop.	Easy way of looping something 5 times to give the output result that I want.
N(i+1).text, N(i+1)S.text	String, Integer	These nodes are assigned what the SCORE and USERNAME holds for that user's row.	Justifies the database synergy between the username and scores.

Test plan

Some of the tests below are designed to break my program, so I know what programs may occur if they ever happen.

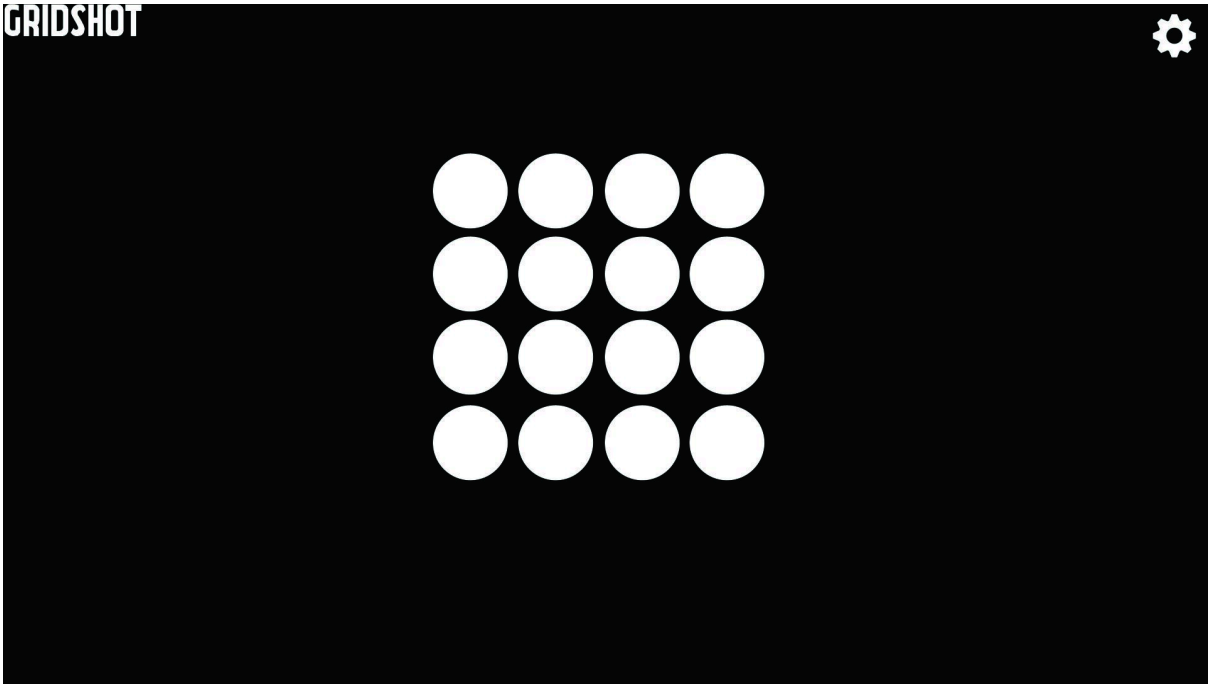
Test Description	Test Data	Expected Outcome

Test to see if clicking on leaderboard shows info	User clicks on leaderboard button	Database queries and sets the nodes to the queried information.
Test to see if clicking on the settings button takes the user to the settings screen	User clicks on settings button	Proceed to settings with preloaded information on the nodes.
Test by clicking any other key on the keyboard	User clicks any key on keyboard	Nothing
Test to see if play button works	User clicks on play button	Should be sent to the first game mode
Test by invalid userID	New userID outside of database	error/crash

Game mode 1

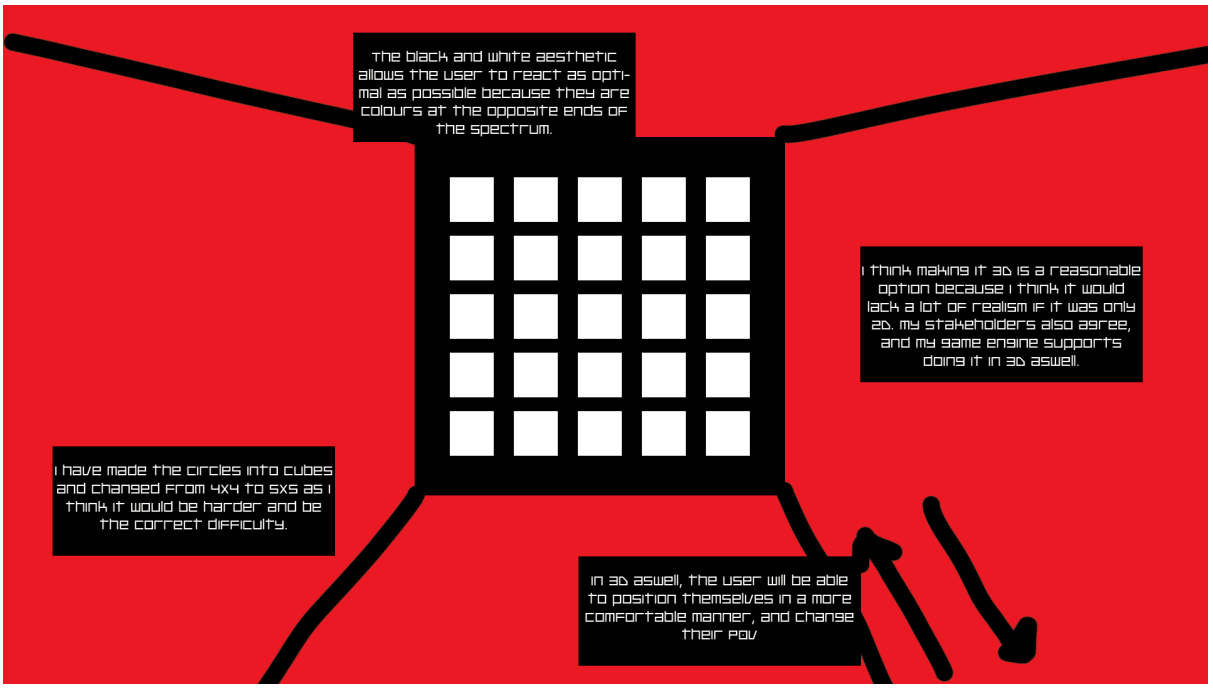
Screen Design

Iteration 1



My simple rendition with a black and white layout for simplicity.

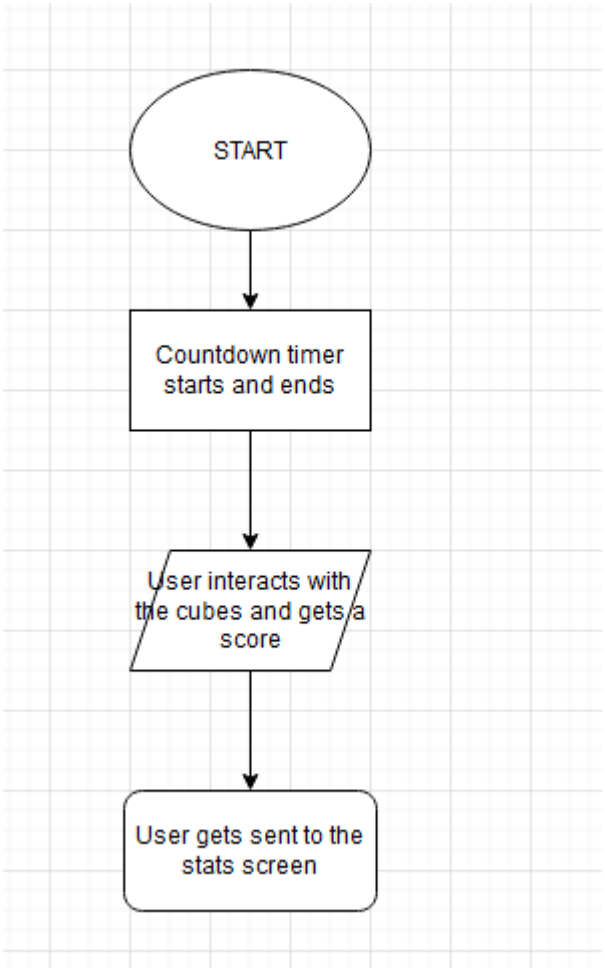
Iteration 2



I have decided that I will create one really complex game mode instead of 3 simple ones. This means that this game mode will probably be the only one, but it will have a modular design within the code, so I can use it if I wanted to add more functionality and game modes. This counteracts my first requirement, but having a quality game mode that takes account of user input accurately and reaction time will produce a more wonderful game for the user in the end. My stakeholders also agree with this choice of action because other features such as movement and having a better angle to click on the cubes make it more realistic.

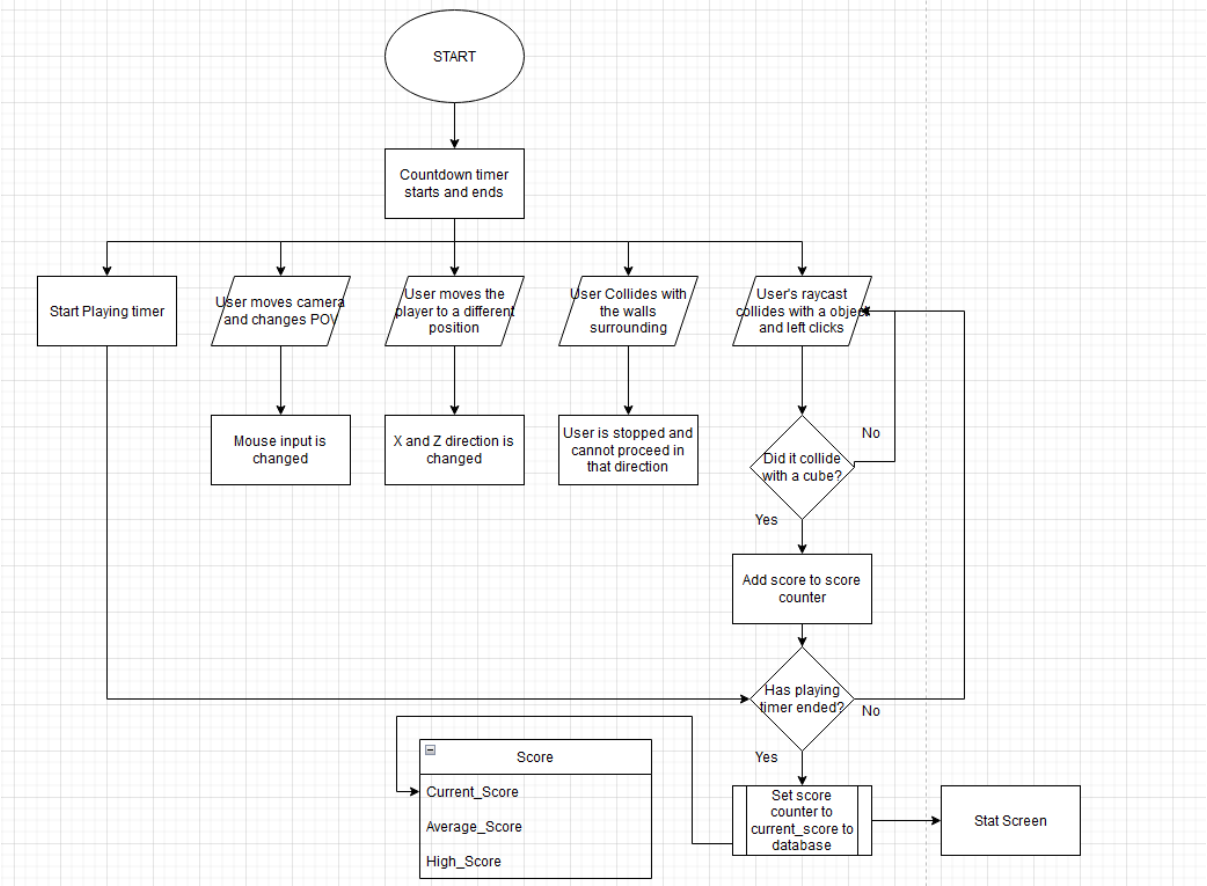
Flowcharts

Iteration 1



This is the initial flowchart of how the round will go, with these key parts, I will be able to expand with decision operators and databases.

Iteration 2



This flow chart shows the 5 possibilities the program will have the capability to go towards. Because I'm making it in 3D, the user is not confined to a spot, and they will be able to move freely until colliding with a wall. The score counter is also added, where it will loop when the user has left click on specifically a cube or not until the playing timer stops.

Pseudocode

Iteration 1

#Scene is loaded

load Game mode_1
preload global_variables # This preloads any global variables that are in other sections of code
Sqlite = preload("(SQLITE ADDON PATH)") # preload the sqlite addon so it is running before the code is ran
variable db # assigning database to a variable
variable db_path = "(path)" # getting the path to the database that I am going to work with
variable table_name = "(table name)" # name of the table I want to work with
variable Playingtimer = false # preset to false to not trigger one of my if statements
variable raycast = (node) # assigning a reference to a node

```
variable UID = retrieved ID from login that logged in previously

# Functions
function Countdowntimer(): # visual representation of the 3,2,1 countdown
    variable counter = 3
    If counter = 3:
            set visible image "3"
            counter = counter - 1
            wait(1)
            Countdowntimer() # calls itself
    ELIF counter = 2:
            set visible image "2"
            counter = counter - 1
            wait(1)
            Countdowntimer()
    Elif Counter = 1:
            set visible image "1"
            counter = counter - 1
            wait(1)
            Countdowntimer()
    ELSE:
            return

    ENDIF

ENDFUNCTION
function Playingtimer(): # Timer counting down from 60 within the 3D space
    variable timer = 60
    for (i, timer):
          timer = timer - 1
          timer.set_text("timer.text")
    ENDFOR
    return Playingtimer = false

ENDFUNCTION

# Main Game
IF the event = load Game mode_1: # When the game is loaded, this statement is happens
first
    Countdowntimer()
    Playingtimer = true # set this to true so the if statement below runs
    Playingtimer()
END IF

IF the event = Input(left_click) and raycast = colliding() and Playingtimer = true: # when the
countdown is done, the player needs to left-click a cube to get score
    variable hit_target = raycast.colliding()
    IF hit_target is in group "Cubes":
```

```
            variable Score = Score + 10
           hit_target(hide)
            IF Playingtimer = true:
                    pass
            ELSE:
                    db.Sqlite.new() # new instance of the sqlite addon
                    db.path = db_path # set the path of the database
                    open db # open the database
                    db.insert_row (into table_name, where field SCORE = Score AND field
UserID = UID) # Inserts score achieved into the database
                    global variable current_score = Score
                    Game mode_1.change_scene("Stats")
            ENDIF
    ELSE:
            pass

    ENDIF

ENDIF
```

In this pseudocode, I put an emphasis on the timers, getting the conditions for the if statement to happen and writing the score to the database. I did not include the reaction time section as this is not that major in terms of the functionality of the program, so there was no need for a design for it. However, this pseudocode approves of many requirements, such as the 9th requirement where low spec computers can run it, and the data application of writing data to the database.

Variable/Function/Node	Data Type	Use	Justification	Validation
Global_variables (E.g. Current_Score, Average_Score, High_Score)	Integer,	This will be a library/section of globally accessible code, which can be changed through new assignments.	This is so nodes and other variables in one scene can be assigned using these global variables, which can be used in another.	Not applicable for validation because it is not accessible for the user.
db, db_path, table_name	String	A standard practice within godot to do something with a database.	So the database will be connected and I can understand which and how the database is inputting and outputting results.	Not applicable for validation because it is not accessible for the user.

Average_score.text, High_Score.text	Integer	This is changing a visible object that the user can see, in this case, the text/string will change depending on what I assign it.	This is the easiest and most efficient way of changing a visible object's text to the correct information.	Not applicable for validation because it is not accessible for the user.
Playingtimer	Boolean	A changing variable that will be assigned depending on what needs to happen in the program.	As a decision, decide when to run an if statement or not.	Not applicable for validation because it is not accessible for the user.
Countdowntimer()		To set visibility to the images of the 3,2,1, where the user cannot click a left click.	As a "get-ready" phase for the user before the round actually starts.	Not applicable for validation because it is not accessible for the user.

Test plan

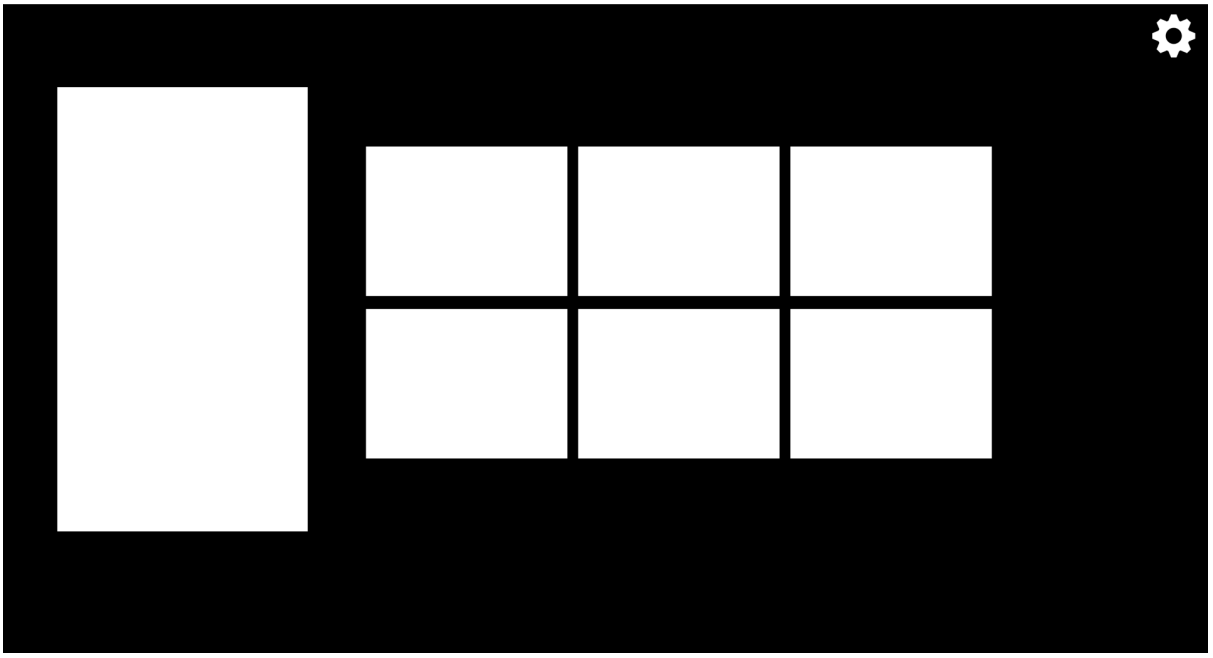
Some of the tests below are designed to break my program, so I know what programs may occur if they ever happen.

Test Description	Test Data	Expected Outcome
Player colliding with the walls	WASD keys and collision	The player stops
The raycast collides with a cube	Left clicking a cube	The cube hides and the score is incremented depending on the reaction time between cube hits.
The raycast collides with the wall	Left clicking the wall	Nothing should happen
The countdown timer is active	The PlayingTimer() function is still running	Left clicking on a cube should have no effect, but movement should be normal.

Stat Screen

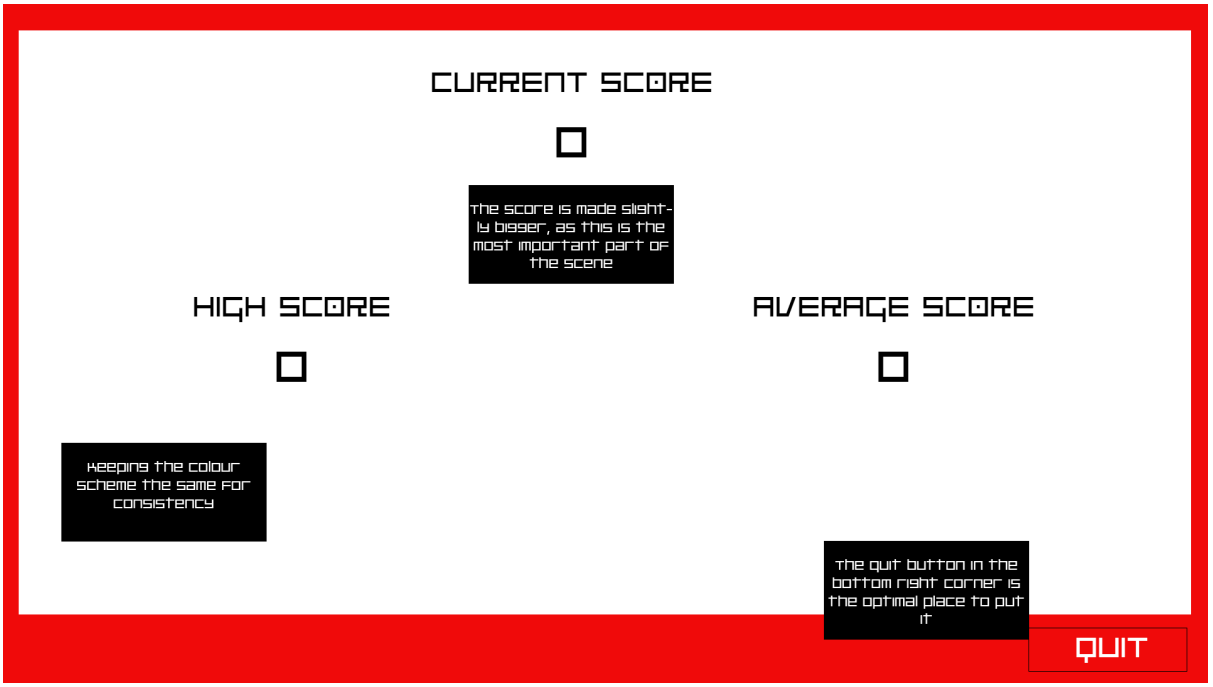
Screen Design

Iteration 1



This is the standard statistics menu that I have seen in most similar games that generally works well for the user.

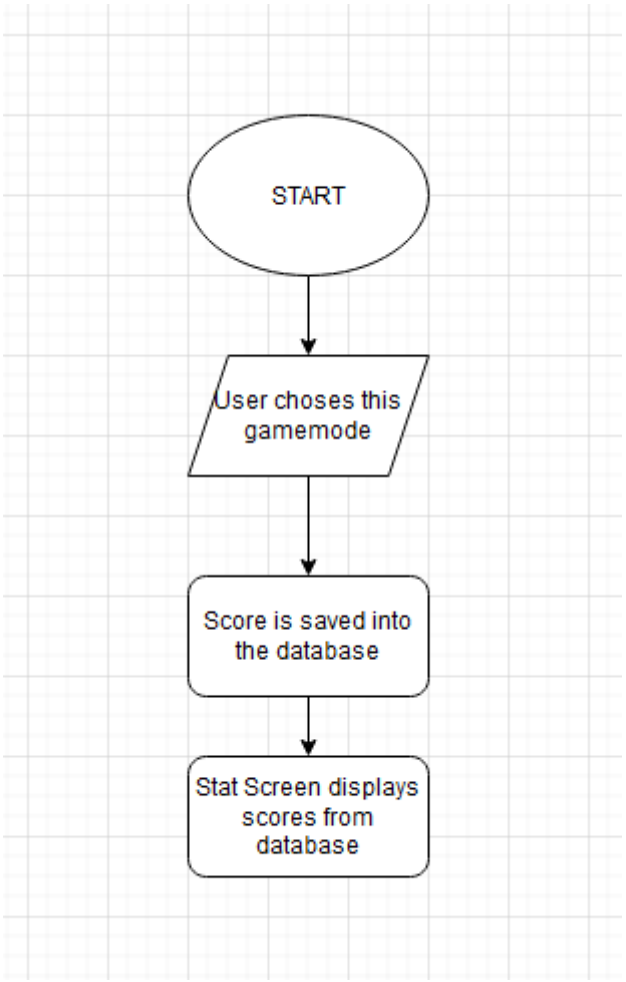
Iteration 2



By adding the relevant colour schemes, I plan to keep the layout simple but I don't want it to be the same as the standard. This is why I just added the 3 score counters in different locations in the scene so the user can see all of them at the same time when it first changes scene.

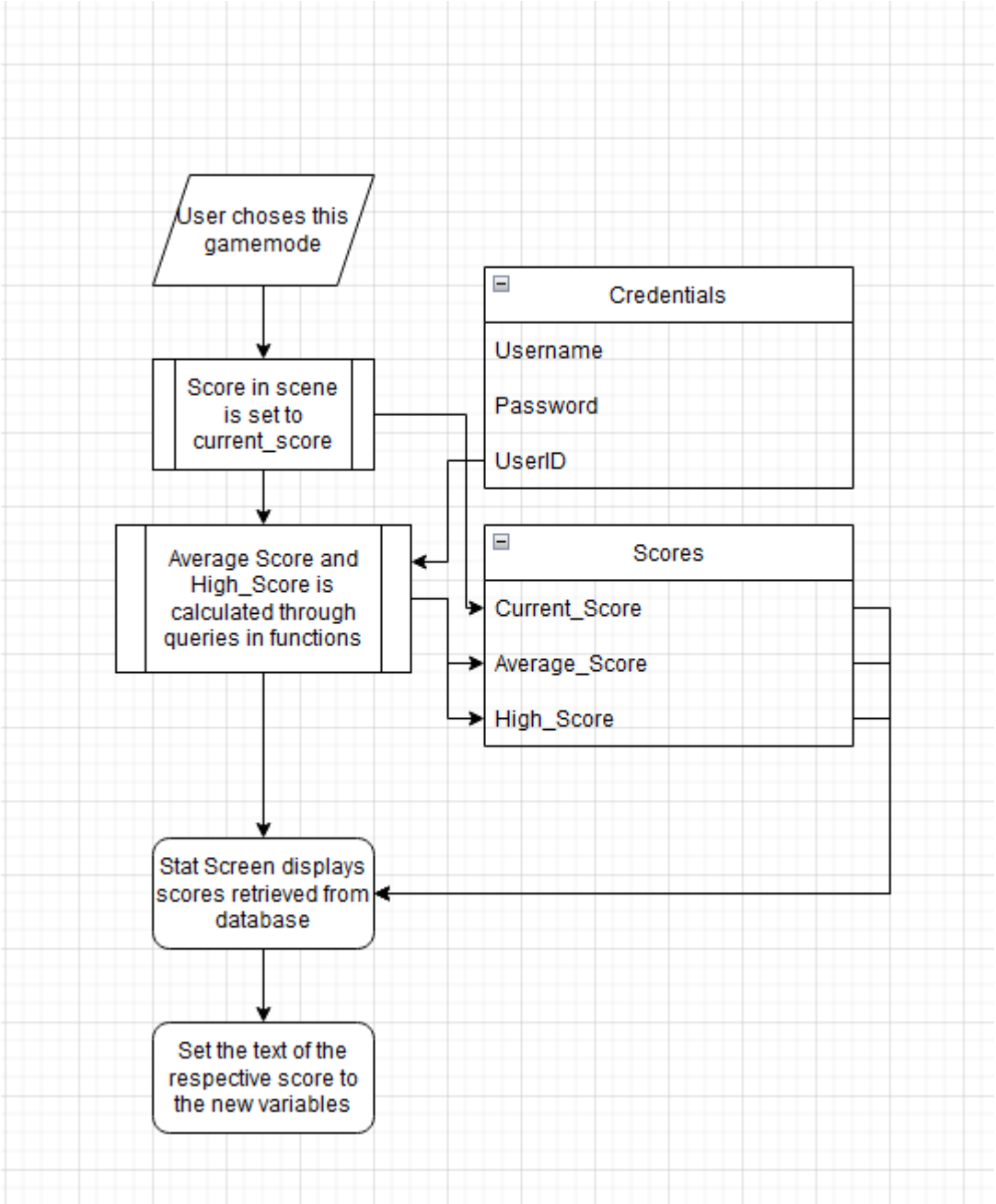
Flowcharts

Iteration 1



Similar to the game mode design, this is my initial draft of the user obtaining a score and the stat screen retrieving the information from the database.

Iteration 2



This gives more evidence to complete my 6th requirement and the average and high score is calculated within this scene using a function, as shown in the pseudo code.

Pseudocode

Iteration 1

#Scene is loaded

load Stats

```
preload global_variables # This preloads any global variables that are in other sections of code
Sqlite = preload("(SQLITE ADDON PATH)") # preload the sqlite addon so it is running before the code is ran
variable db # assigning database to a variable
variable db_path = "(path)" # getting the path to the database that I am going to work with
variable table_name = "(table name)" # name of the table I want to work with
variable UID = retrieve ID from login that logged in previously

# Function
def calc_scores():
    db.Sqlite.new() # new instance of the sqlite addon
    db.path = db_path # set the path of the database
    open db
    db.query(SELECT ALL FROM table_name WHERE USERID = UID AND SCORE = MAX(SCORE))
    global variable High_score = db.query_result[0]["MAX(SCORE)"] # Finds the max value and sets it to the global variable
    db.query(SELECT ALL FROM table_name WHERE UID = UserID AND SCORE = AVG(SCORE))
    variable Average_score = db.query_result[0]["AVG(SCORE)"] # Finds the max value and sets it to the global variable
    return

# Main Game

If Event = Game mode_1.change_scene("Stats"):
    calc_scores()
    # The text shown to the user are the nodes.
    node High_Score_Number.text = global variable High_score
    node Average_Score_Number.text = global variable Average_score
    node Current_Score_Number.text = global variable Current_score

IF back_button pressed: # there is always a back button for all these scenes
            Stats.change_scene("main_menu")
```

When the program is changed to this scene, it should calculate the average and high scores and set them to the nodes for the user to visibly see. Additionally, now the global variables high_score and average_score are now got their up-to-date values to be used elsewhere in the program.

Variable/Function	Data Type	Use	Justification	Validation
Global_variables (E.g.	Integer, Boolean	This will be a library/section of globally accessible	This is so nodes and other variables in one	Not applicable for validation because it is not

Current_Score, Average_Score, High_Score)		code, which can be changed through new assignments.	scene can be assigned using these global variables, which can be used in another.	accessible for the user.
db, db_path, table_name	String	A standard practice within godot to do something with a database.	So the database will be connected and I can understand which and how the database is inputting and outputting results.	Not applicable for validation because it is not accessible for the user.
Average_score_Number.text, High_Score_Number.text, Current_Score_Number.text	Integer	This is changing a visible object that the user can see, in this case, the text/string will change depending on what I assign it.	This is the easiest and most efficient way of changing a visible object's text to the correct information.	Not applicable for validation because it is not accessible for the user.
calc_scores()	Integer	This function calculates both the average and high score that the scene needs.	This is done through queries in the database, where the data is retrieved on set conditions.	Not applicable for validation because it is not accessible for the user.

Test plan

Test Description	Test Data	Expected Outcome
Test the back button	Left click the back button	Return to the main menu
Test to check the profile statistics	Left click the back button, into the profile button	The data should be the same
Test to see if there is a current score	Enter the database using the DB browser, and query the database with the same statement	Error message
Test to see if the average and high score is accurate	Enter the database using the DB browser, and query the database with the same	Same result

	statement.	

Post Development Testing

Once I have completed development, I shall test everything to see if it is running correctly for my stakeholders to use.

For my login screen, I need to test all the available options for the user to input into both the username and password field. It should go onto the main menu if it's correct, or display error messages if it's wrong.

Test Description	Test Data	Expected Outcome
Test to see if the correct username and password is accepted into database	A user username and password	Changes scene to the main menu
Test to see if the incorrect username and password is accepted into database	Incorrect username and password	An error message is displayed for the user, which changes depending on which error is made.

Next, I would test the main menu and some of the subsections such as the settings and profile.

Test Description	Test Data	Expected Outcome
Test to see if the quit button in the main menu works	Left click the quit button	Closing down the program, it stops all of the scripts that were storing variables.
Test to see if values in the database is stored within the profile	Left click the profile button	Database should be opened and queried to find max and average scores.
Test to see if the setting that I have turned on has stayed the same	Left clicking the settings button and then the back button (repeat)	The setting switch that has been turned on, remained on.

Finally, and most importantly, the game mode 1 section needs to be tested heavily.

Test Description	Test Data	Expected Outcome
Test to see if the player can move outside the red boundary walls.	WASD, or Up/Down/Left/Right keys to move the player.	The player should be brought to a halt, where it should not allow the user to move in that direction.
Test to see if the player can click the cubes in rapid succession.	Mouse input, raycast and object.	The score counter should be incremented depending on the reaction speed.
Test to see if clicking a cube will hide it, and make another one visible.	Input left click	Cube that got hit is set to hide and a random other cube is set to visible.

Usability

In this section, I will describe and justify any usability features that I have made within each of the design phases that will be made into my final iteration of my game.

From the Login Screen, I have tailored the colour scheme to make the UI the least confusing as possible by making use of the complementary colours of white and black. This works well since the background is also the main colour of the game, and keeping it a simple and intuitive palate that minimises any sight-impaired issues that may arise. Furthermore, this screen is the first screen that the user will see, so the square, or rectangular aspect of each box or even the logo is done intentionally because everything in the game is cube-related that makes it have a consistent theme. By designing both the login button and quit button to be the same colour scheme results in the user subconsciously recognising that these are buttons that are actions to change screen or quit the game. Validation is crucially implemented into this scene, in particular from the error messages. I have annotated this in the screen design, where each error message provides a message that directly links with the problem so the user doesn't need to check everything they had inputted.

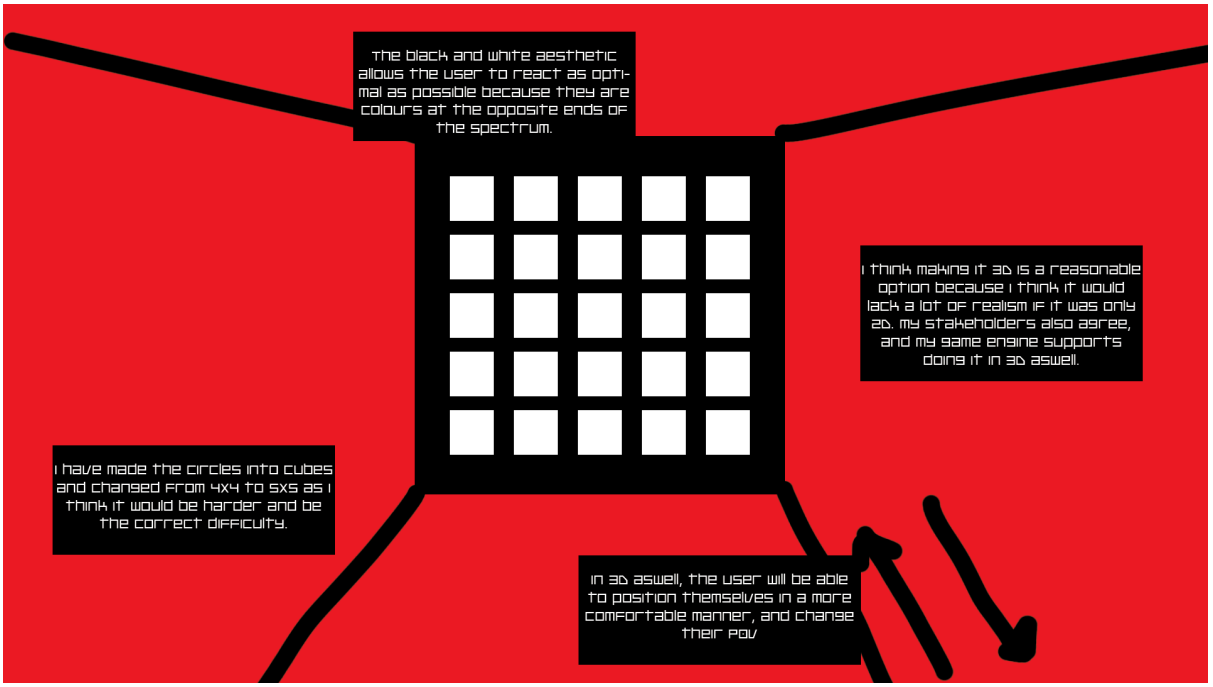
From the Main Menu, I have laid out the buttons and the logo to be centred around the middle of the screen. This is because most users will see the centre first, and this screen is used for navigation through other scenes, so this scene will probably be the most used one. Also, the validation is done throughout this game from the navigation buttons on the main menu, or the back button in the internal scenes. For each of my screens, I aim to keep the colour scheme consistent so only red, white and black will be used, and the buttons will be large enough for anyone who has problems accurately left clicking.

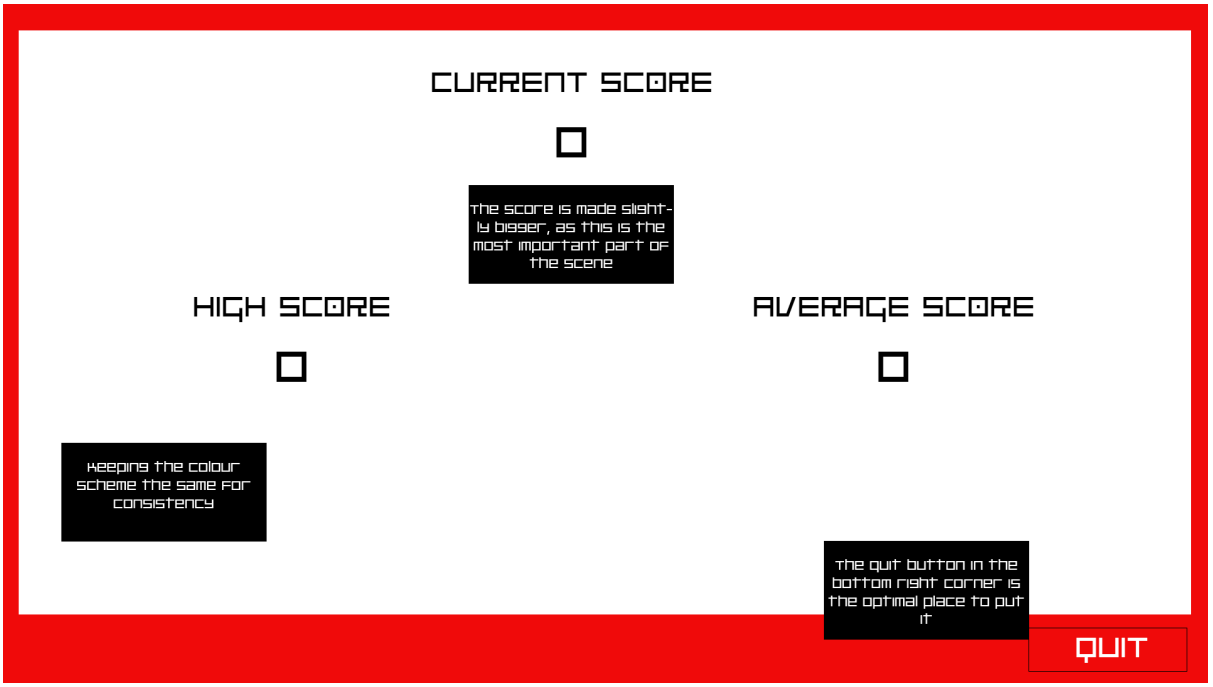
From the Stats Screen, I have massively changed the layout of this screen because I felt like that design was too complex and unworthy to allocate more time to make that style when I decided to not make use of graphs. The data in the new layout simply displays the hard statistics that are easy to understand even for beginners, and these counters will still help

them improve because they update each time the scene is loaded in. In addition to this, making the numbers consistent and simplistic allows people who aren't fluent in English to still understand if they are improving as in most countries, the English numbers are understood more than the language.

Stakeholder Feedback


buttons are rearranged for a new layout, the old one seemed a bit too overused and i wanted to try something new.
the logo is moved to the middle as it is the first thing that users generally see. this makes it more my game.
PLAY
AIM TRAINER
PROFILE
LEADERBOARD
SETTINGS
the quit button in the bottom right corner is the optimal place to put it
QUIT


the black and white aesthetic allows the user to react as optimal as possible because they are colours at the opposite ends of the spectrum.
i think making it 3d is a reasonable option because i think it would lack a lot of realism if it was only 2d. my stakeholders also agree, and my game engine supports doing it in 3d aswell.
i have made the circles into cubes and changed from 4x4 to 5x5 as i think it would be harder and be the correct difficulty.
in 3d aswell, the user will be able to position themselves in a more comfortable manner, and change their pov



With these screen designs, I have provided them for my stakeholders to look one final time before starting development. Generally, the feedback was positive and like my choice of choosing 3D, but I need to make sure the cubes are 3D and I need to be prepared if they were going to cause an issue with my raycast collision. Furthermore, I had some small issues such as that the logo in the login screen and main menu felt a bit too inconsistent with the curves and straight lines, and because I am going for a simplest design, the non-text logo would be better as the focus and the text needs to be more refined.

Overall, I will take these comments into consideration when I design it once again within the godot engine now because I am making it in 3D.

17/02/2023

Development + Testing

Phase Overview

In my development stage, I have split my project into different and unique phases, where Each of them are related to one or more points in my success criteria, where it allows me to keep track of what I want my game to have to be successful. Additionally, the order of the phases are ordered in level of importance because it will allow me to do more essential features that will only then allow me to proceed with further phases.

Phase Number	Title	Success Criteria
1	Login Screen and Main Menu	1, 3, 9
2	Game mode Creations	2, 4, 5, 6, 7, 8
3	Settings	9
4	Statistical Screen	2, 3

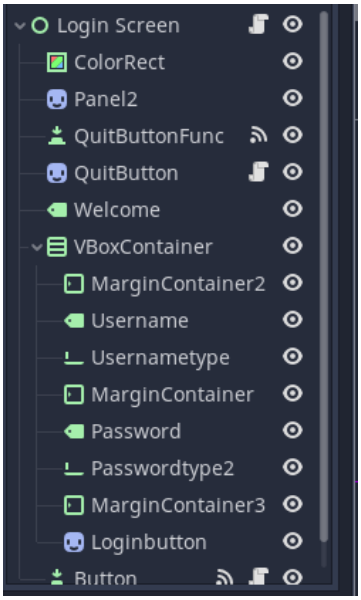
In each of these phases, I will demonstrate a good iterative and modular process. This is where I will be commenting on the relative sections of code, testing and debugging, and then most importantly reviewing by asking my stakeholders questions on how well it works and if it achieved my purpose of the phase. This is important because it will allow me to see if there are any code or features that could be improved to make the game run better for the user, and doing this iterative style of programming allows me to make sure I don't run into problems from earlier code later down the project.

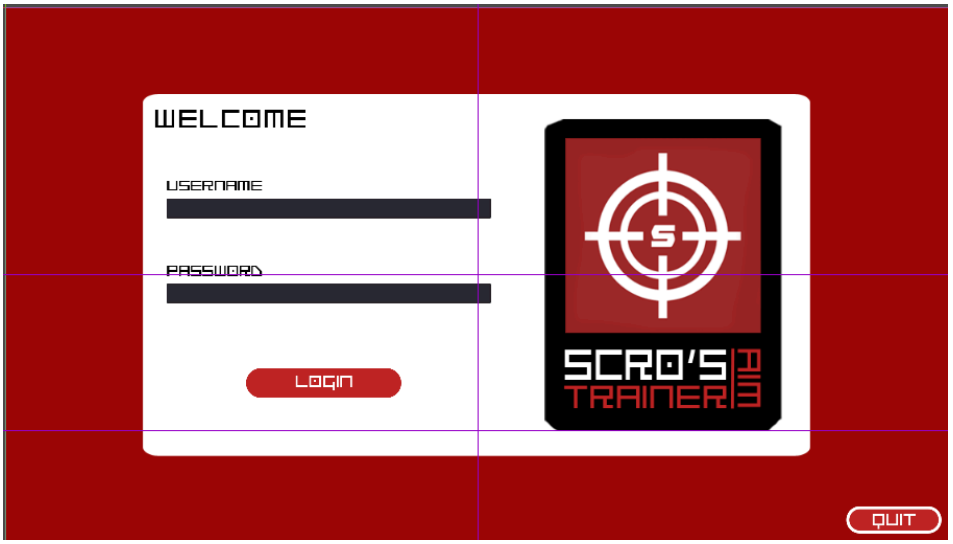
Phase 1 - Login Screen and Main Menu

In this first phase for my login screen, I plan to complete my initial idea of my Success Criteria 1 by making it easily functionable for the user, but also creating a database to safely store all the credentials that the user would use to sign into their own personal profile.

Step 1: Making the login screen template

To start, I had created my simple login screen with 3 simple colours of black, white and red, whilst also maintaining simplicity in the design, so I am able to keep it modular that will allow me to easily input different commands/code for each of my buttons.





As such, I have currently programmed my prototype versions of my quit and login buttons to do as mentioned so easily test and help validate my game (Test 1). Now that the basic functions are done, I will need to set up a database so then I will be able to save data and allocate it for use, such as verifying if the credentials are correct for the selected user.

Step 2: Creating the initial verification system and database, using only username

Firstly, I require a database so I am able to store and retrieve the data for verification or for further use in code. As such, I initially wanted to use FireBase which is something I used many times for previous games, but thinking back to my limitation section, I do not intend for the database or the game to go online. Therefore, I had decided to use SQlite with SQLite Browser as I had learnt about it during one of my components.

```
const SQLite = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns") # Before the game starts, preload before anything happens

var db # database
var db_name = "res://Data Storage/Player Credentials" # Path to database
var tableName = "PlayerCredentials"
```

Following a tutorial, I had set up a referral link between SQLite and Godot. As such, I will be able to access my data in my database while calling it in code during my scripts. Additionally, this was only possible with an import of the SQLite program using Assetlib, which I had to learn a new command list.

```
#Getting text from the user
func _on_Usernametype_text_changed(NameofUser):
    db.open_db()
    db.query("SELECT Username FROM " + tableName + " WHERE Username=" + NameofUser + ";")
    if db.query_result[0]["Username"] == true:
        print("True")
    else:
        print("False")
```

I have tried to initially test out the system by using a tutorial to try to see how to validate and search through the database. Using the command list from the plugin's official website on github of "Godot Sqlite" , I was unable to code the program to function correctly as I had made a mistake that I had fixed (Test 3). Then, the entire section of code was rewritten to implement the singletons/global variables shown below.

```
func _on_Button_button_up():

    var Username = $VBoxContainer/Usernametype.text
    var Password = $VBoxContainer/Passwordtype.text

    db = SQLite.new()
    db.path = db_name
    db.open_db()
    db.query("SELECT * FROM " + tableName + " WHERE Username = '" + Username + "';")
    if db.query_result.size() == 0:
        print("No Matching Username Found.")
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Username:
        print("Found Username")
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] != Username:
        print("This should never happen")
    elif db.query_result.size() >= 2:
        print("This should never happen 2")

    else:
        print("This should also never happen")
```

Currently, I have made this in mainsystem.gd for my login screen, to allocate all of the variables such as the Username and Password from their corresponding inputs from the user. This is just a brief overview of what I'll need to do to compare and verify if the data is correct for the username, but I need to make sure that the code is modular and repeatable so it can be used similarly for the password. Additionally, the "This should never happen" statements will help me debug the program and the debugger will show me which line is causing a problem, which will help with the testing phase of the project. (Test 2)

Step 3: Implementing the password and validating the data

```
func _on_Button_button_up():

    var Username = $VBoxContainer/Usernametype.text
    var Password = $VBoxContainer/Passwordtype.text

    db = SQLite.new()
    db.path = db_name
    db.open_db()
    db.query("SELECT * FROM " + tableName + " WHERE Username = '" + Username + "' AND Password = '" + Password + "';")
    if db.query_result.size() == 0:
        print("No Matching Player Found.")
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] == Password:
        print("Found Player.")
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] != Username or db.query_result[0]["Password"] != Password:
        print("This should never happen")
    elif db.query_result.size() >= 2:
        print("This should never happen 2")
    else:
        print("This should also never happen")

```

With the Username function fully setup, I can now easily insert my required password and validate it within the database to be able to match with the username. This was achieved by using the condition of “and” and I have used the condition “or” to represent something that should not be happening. Also, as I created my program within a function, it allows me to reorganise the code (Test 4)

Step 4: Error messages on incorrect information

```
func _ready():
    if $VBoxContainer/error_label_msg.is_visible():
        $VBoxContainer/error_label_msg.hide()

```

```
func _on_Button_button_up():

    var Username = $VBoxContainer/Usernametype.text
    var Password = $VBoxContainer/Passwordtype.text

    db = SQLite.new()
    db.path = db_name
    db.open_db()
    db.query("SELECT * FROM " + tableName + " WHERE Username = '" + Username + "' AND Password = '" + Password + "';")
    if db.query_result.size() == 0:
        $VBoxContainer/error_label_msg.show()
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] == Password:
        get_tree().change_scene("res://MainMenu.tscn")
        db.close_db()
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] != Username or db.query_result[0]["Password"] != Password:
        print("ERROR CODE_1: INVALID RESULT")
    elif db.query_result.size() >= 2:
        print("ERROR CODE_2: MORE THAN ONE PROFILE")
    else:
        print("ERROR CODE_3: UNKNOWN")

```

To make it more user-friendly and letting the user know what is wrong, I have added an error message that would appear once the information inputting is correct. This was done by initially hiding the text using .hide() if it was visible as the game first load, and it would reveal once the inputting information was incorrect. If it is correct, the button would now send it towards the Main Menu without the use of the "_on_button_pressed" function that I had temporarily made. Furthermore, I have changed the print messages to a more specific area of what may have gone wrong, if there are to be any further problems in the future. (Test 5)

Reflection: Prototype and comments from the stakeholders

Once the login screen had been finished, I sent this prototype to my stakeholders and received critical feedback on the functionality of the login screen. From my primary stakeholder, he had mentioned that there was no way of knowing which part of the username or password was wrong, thus he could not login. As such, this can be simply done by changing the other error messages to "Username is incorrect" or "Password is incorrect" to fix this issue. From my secondary stakeholder, the issue he had was different. He wasn't able to sign up to make a profile in the first place, so no login he used worked. I planned to do this at the end as it will take a lot of time to implement, so this will be a feature if I have the time.

Step 5: Adding the stakeholder comments to profile

```
func _ready():
    if $VBoxContainer/error_label_msg.is_visible():
        $VBoxContainer/error_label_msg.hide()
        $VBoxContainer/error_username.hide()
        $VBoxContainer/error_password.hide()
        $VBoxContainer/error_not_found.hide()

func _on_Button_button_up():

    var Username = $VBoxContainer/Usernametype.text
    var Password = $VBoxContainer/Passwordtype.text

    db = SQLite.new()
    db.path = db_name
    db.open_db()
    db.query("SELECT * FROM " + tableName + " WHERE Username = '" + Username + "' AND Password = '" + Password + "';")
    if db.query_result.size() == 0:
        if $VBoxContainer/error_password.is_visible():
            $VBoxContainer/error_password.hide() # password hide
        if $VBoxContainer/error_username.is_visible():
            $VBoxContainer/error_username.hide() # username hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_not_found.show()
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] == Password:
        get_tree().change_scene("res://MainMenu.tscn")
        db.close_db()
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] != Password:
        if $VBoxContainer/error_not_found.is_visible():
            $VBoxContainer/error_not_found.hide() # login not found hide
        if $VBoxContainer/error_username.is_visible():
            $VBoxContainer/error_username.hide() # username hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_username.show()
    elif db.query_result.size() == 1 and db.query_result[0]["Username"] != Username and db.query_result[0]["Password"] == Password:
        if $VBoxContainer/error_not_found.is_visible():
            $VBoxContainer/error_not_found.hide() # login not found hide
        if $VBoxContainer/error_password.is_visible():
            $VBoxContainer/error_password.hide() # password hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_password.show()
```

As the stakeholder mentioned, I had begun to experiment with the development of the error messages and thought that using the same methods as my Step 4, it would function the same. However, whenever I ran and tested the program (Test 6), it strangely didn't change when I clicked the button once again, with the shown code below.

```
if db.query_result.size() == 0:
    if $VBoxContainer/error_password.is_visible():
        $VBoxContainer/error_password.hide() # password hide
    if $VBoxContainer/error_username.is_visible():
        $VBoxContainer/error_username.hide() # username hide
    $VBoxContainer/error_label_msg.show()
    $VBoxContainer/error_not_found.show()
```

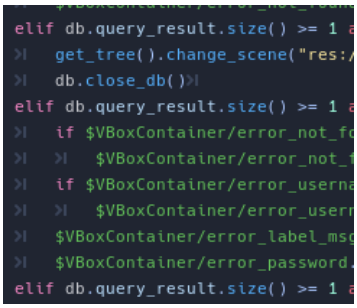
```
elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Usernam
    if $VBoxContainer/error_not_found.is_visible():
        $VBoxContainer/error_not_found.hide() # login not found hide
    if $VBoxContainer/error_username.is_visible():
        $VBoxContainer/error_username.hide() # username hide
    $VBoxContainer/error_label_msg.show()
    $VBoxContainer/error_username.show()
```

Nevertheless, the problem was finally solved when I realised that all the data my database was sending was only when there was a match to the database that had both username

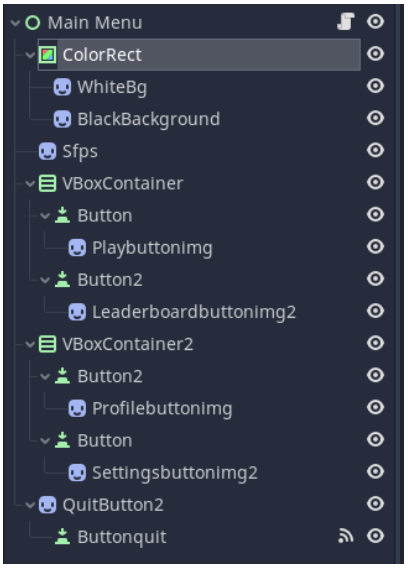
AND password correct. As such, I changed the AND operator to an OR to see if this fixes the problem.

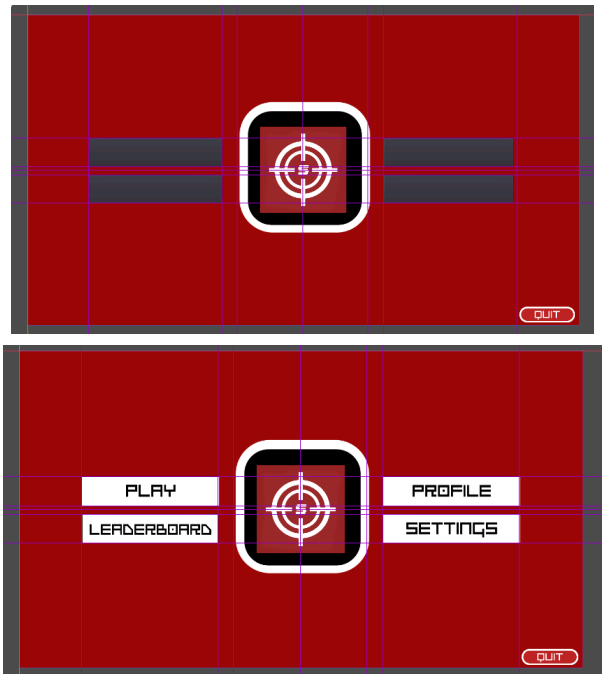
What happened after was that my username and password code was accidentally switched, so when the username was wrong the “password is incorrect” appeared. This was then easily replaced whilst using my debugger tools.

Unexpectedly, I had an issue with having a correct username and correct password, but they had different ID’s. To combat this, I removed the “size()>=2” error console message, and replaced the “size==1” if statements to a “size>=1”, which finally fixed the problem. (Test 7)



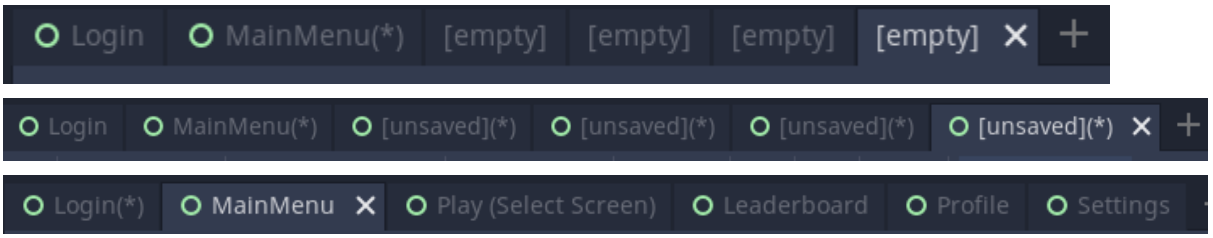
Step 6: The main menu template





Likewise in the login screen, I created a template for the main menu. Sticking with my simple colour scheme, I have only used the colours of black, white and red to make this main menu, with the same colour rect background as the login screen. Additionally, I made sure to put all of my variables into their own separate sections within the VBoxContainer1 and 2, so then I can easily assign scripts/code to each button to link to other scenes of my project.

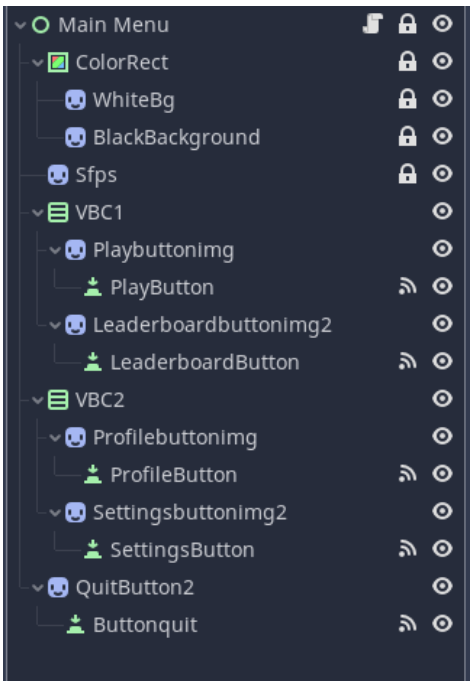
Step 7: Creating the main menu buttons and functionality



Now, I have created the corresponding scenes to match each of the buttons. These will be the last UI screens of the project, and making sure that they have the same dimensions as the previous 2 scenes so the user can navigate the game easier. Furthermore, this will help navigate through each scene with ease, as it separates scripts from one another, so then I will be able to understand faster and debug if there are any issues. I can still get variables from other scenes to take them into another, so there are no potential issues.

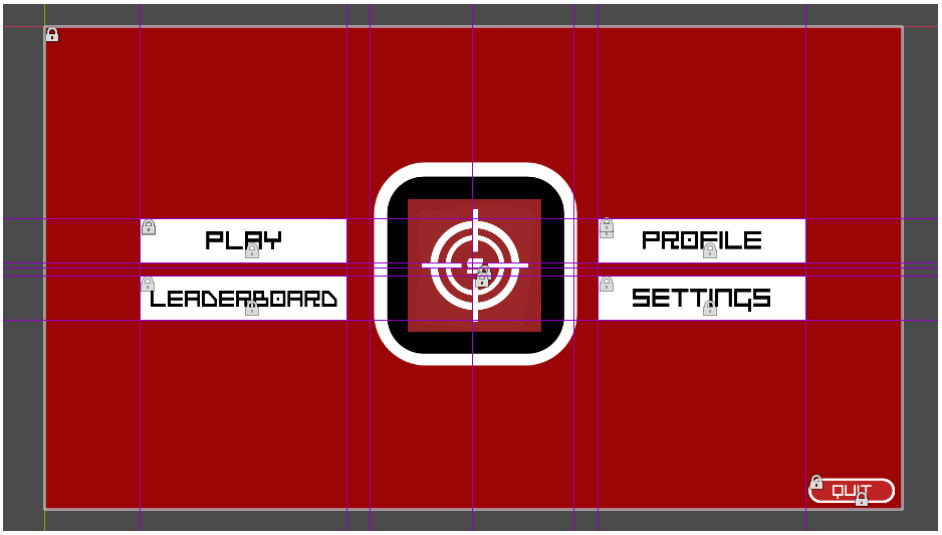
```
1  extends Control
2
3  # Declare member variables here. Examples:
4  # var a = 2
5  # var b = "text"
6
7  func _on_Buttonquit_pressed():
8      get_tree().quit()
9
10 func _on_PlayButton_button_up():
11     get_tree().change_scene("res://.import/Play (Select Screen).tscn")
12
13 func _on_LeaderboardButton_button_up():
14     get_tree().change_scene("res://.import/Leaderboard.tscn")
15
16 func _on_ProfileButton_button_up():
17     get_tree().change_scene("res://.import/Profile.tscn")
18
19 func _on_SettingsButton_button_up():
20     get_tree().change_scene("res://.import/Settings.tscn")
21
```

For each of the buttons, I have now made different functions to change the scenes when the button is clicked. However, when I go to test this, it does not change the scene. With problem solving, I initially think it is due to the image. This is because I found out that godot makes the bottom layer the accessible layer instead of the top layer.



Due to this, I have swapped the image layer and the button layer, which I should have done judging from the quit button. But, the problem was not fixed.

From using debugging tools and following through the algorithm, I found out that the project I was currently working on was the “import” version of the Main Menu, instead of the real version that the Login menu had changed the scene towards. Thus, no matter what I clicked and added, it did not add to the actual scene that I wanted, which really confused me.



Now, the main menu is fully functionable as intended, with locking each of the layers to prevent them from moving in the editor.

Testing Phase 1

<table>
<tr><th>ID (T)</th><th>Description</th><th>Code/Input</th></tr>
<tr><td rowspan="4">1</td><td>Functions created to easily test and see if the nodes I created are acting as intended.</td><td><pre>func _on_Button_pressed():
    get_tree().quit()</pre><pre>func _on_Button_pressed():
    get_tree().change_scene("res://MainMenu.tscn")</pre></td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2">First function successfully quits the game.

Second function changes scenes to my main menu successfully.</td></tr>
<tr><td>Fix?</td><td>N/A</td></tr>
<tr><td rowspan="3">2</td><td>Running the program and inputting correct and incorrect credentials to see if it runs successfully.</td><td>Code of success:

<pre>elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Username:
    print("Found Username")</pre>Code of failure:

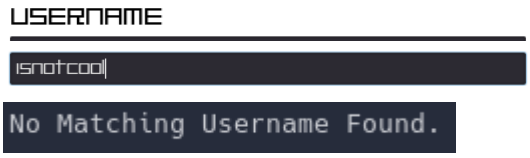
<pre>if db.query_result.size() == 0:
    print("No Matching Username Found.")</pre><pre>elif db.query_result.size() == 1 and db.query_result[0]["Username"] != Username:
    print("This should never happen")
elif db.query_result.size() >= 2:
    print("This should never happen 2")

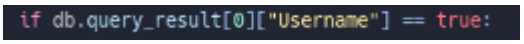
else:
    print("This should also never happen")</pre></td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2">Success Output:

  </td></tr>
</table>

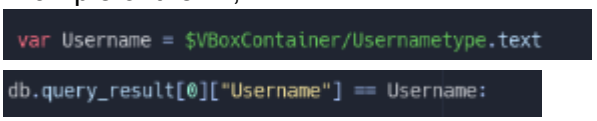
<table>
<tr><td></td><td colspan="2">PASS

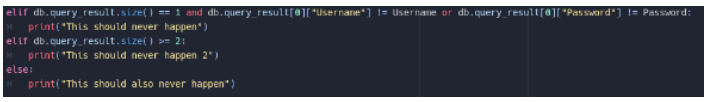
Failure Output:

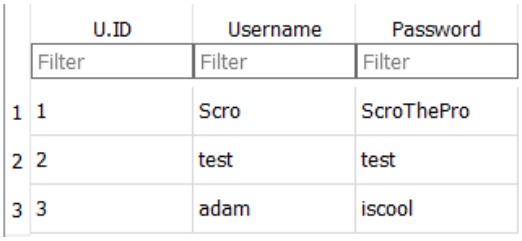
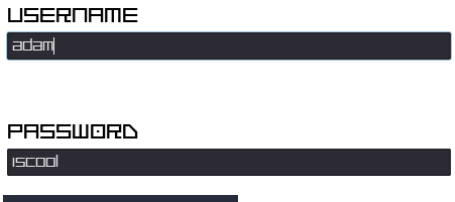
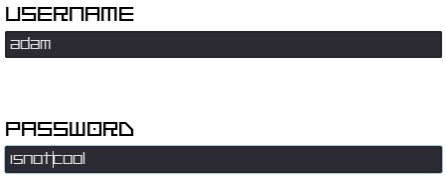
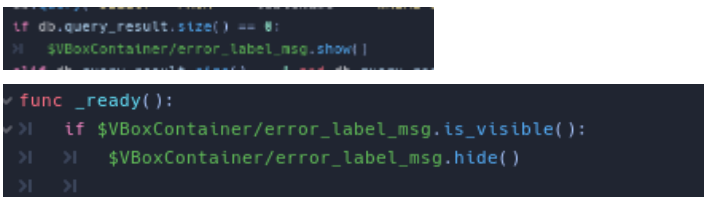
 

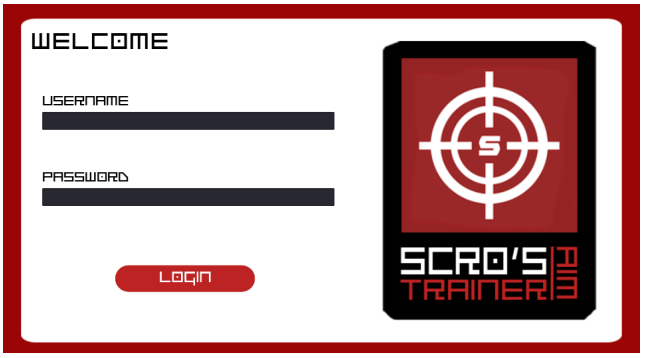
FAIL</td></tr>
<tr><td></td><td>Fix?</td><td>N/A</td></tr>
<tr><td rowspan="4">3</td><td>trying to set the string “db.query_result[0][“Username”]” to a different data type of boolean “true”.</td><td> 

Expected failure.</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td>Fix?</td><td>I need to change the bool operator into a variable that holds the string. I had initially set it to true because I thought this line would return true or false when there is a value in that row, but it actually returns back the string value in that row, so setting a string to be equivalent to a boolean operator is invalid.

Example of the fix,
 
</td></tr>
<tr><td rowspan="3">4</td><td>Testing to see if the password acts the same as the username algorithm.</td><td>Success Output:
 

Failure Output:
 
</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2">Database:</td></tr>
</table>

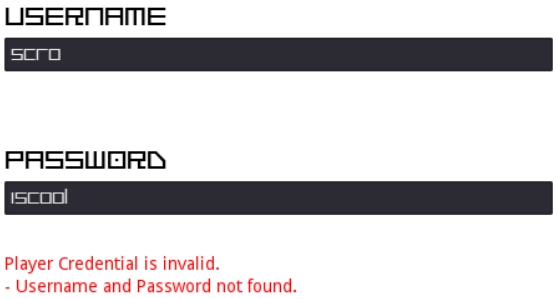
	 Success Output:  PASS Fail Output:  FAIL	
	Fix?	N/A
5	Creating a simple error message once the input is not data in the database.	
	Result	

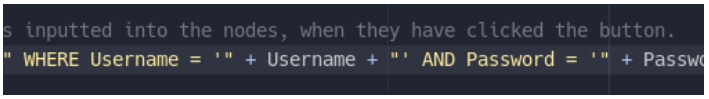


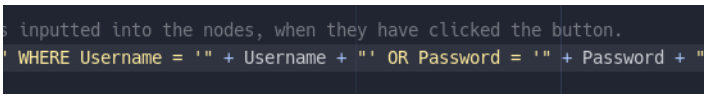
	Fix?	N/A

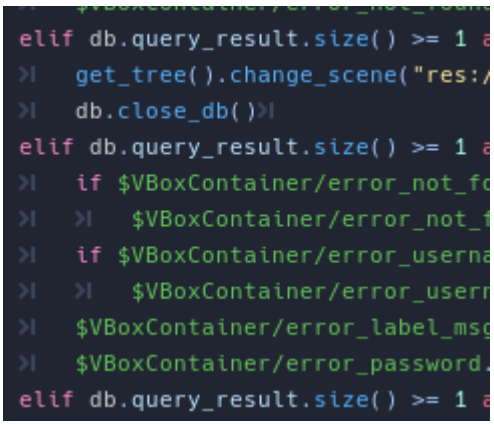
6	Test by changing each if statement would trigger different error messages so the user knows what they inputted wrong	`if db.query_result.size() == 0:`
		`  if $VBoxContainer/error_password.is_visible():`
		`  $VBoxContainer/error_password.hide() # password hide`
		`  if $VBoxContainer/error_username.is_visible():`
		`  $VBoxContainer/error_username.hide() # username hide`
		`  $VBoxContainer/error_label_msg.show()`
		`  $VBoxContainer/error_not_found.show()`
		
		`elif db.query_result.size() == 1 and db.query_result[0]["Username"] == Usernam`
		`  if $VBoxContainer/error_not_found.is_visible():`
		`  $VBoxContainer/error_not_found.hide() # login not found hide`
		`  if $VBoxContainer/error_username.is_visible():`
		`  $VBoxContainer/error_username.hide() # username hide`
		`  $VBoxContainer/error_label_msg.show()`
		`  $VBoxContainer/error_username.show()`
		
		`s inputted into the nodes, when they have clicked the button.`
		`" WHERE Username = '" + Username + "' AND Password = '" + Passwo`

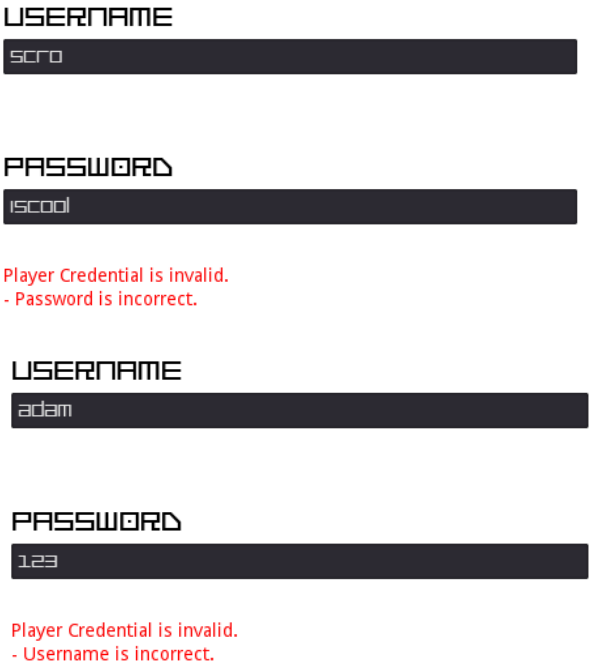
	Result	
	 PASS	

<table>
<tr><td rowspan="3"></td><td colspan="2"> 

FAIL</td></tr>
<tr><td>Fix?</td><td>Nevertheless, the problem was finally solved when I realised that all the data my database was sending through the query statement was only when there was a match to the database that had both username AND password correct. As such, I changed the AND operator to OR to see if this fixes the problem.
 

To
 

Now, running the code, it fixed this issue, but it was not retrieving the correct information. (Continued in Test 7)</td></tr>
<tr><td>7</td><td>Test to see why the query was not retrieving the correct information.</td><td> 
</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td></td><td colspan="2"></td></tr>
</table>

<table>
<tr><td rowspan="2"></td><td colspan="2"> 
</td></tr>
<tr><td>Fix?</td><td>Unexpectedly, I had an issue with having a correct username and a correct password, but they had different ID’s. To combat this, I removed the “size()>=2” error console message, by replacing the “size()==1” if statements to a “size()>=1”, which finally fixed the problem. This is because both the username and password would still need to match a particular ID to access the successful login if statement.
 
</td></tr>
</table>

Review

Overall, I have gotten a deeper level of understanding of GDscript by watching tutorials and command lists. By doing the login screen, I have learnt how to implement how to get the SQlite program to function correctly in godot, thus giving me experience with databases before using it in my main game. With using the debugger and problem solving the code, I learnt the basics of using this new IDE to create my game, and I plan to do a sign up menu at the end if I have enough time.

Additionally, I think I need to allocate my time more efficiently because I do not know if I will have enough time to spend on the more important phases of the game, as a login screen and main menu would not be the focal point. As a reminder, I need to remember to reference

my success criteria and requirements I have made in my analysis, so I can remove or add features that make it suitable for my game.

Final Code

LoginScreen.gd:

```
extends Control

const SQLite = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns") # Before the game starts, preload before anything happens

var db # database
var db_name = "res://Data Storage/Player Credentials" # Path to database
var tableName = "PlayerCredentials"


func _ready():
    if $VBoxContainer/error_label_msg.is_visible():
        $VBoxContainer/error_label_msg.hide()
        $VBoxContainer/error_username.hide()
        $VBoxContainer/error_password.hide()
        $VBoxContainer/error_not_found.hide()
func _on_Button_button_up():

    var Username = $VBoxContainer/Usernametype.text
    var Password = $VBoxContainer/Passwordtype.text

    db = SQLite.new()
    db.path = db_name
    db.open_db()
    db.query("SELECT * FROM " + tableName + " WHERE Username = '" + Username + "' OR Password = '" + Password + "';")
    if db.query_result.size() == 0:
        if $VBoxContainer/error_password.is_visible():
            $VBoxContainer/error_password.hide() # password hide
        if $VBoxContainer/error_username.is_visible():
            $VBoxContainer/error_username.hide() # username hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_not_found.show()
    elif db.query_result.size() >= 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] == Password:
        get_tree().change_scene("res://Scenes/MainMenu.tscn")
        db.close_db()
    elif db.query_result.size() >= 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] != Password:
        if $VBoxContainer/error_not_found.is_visible():
            $VBoxContainer/error_not_found.hide() # login not found hide
        if $VBoxContainer/error_username.is_visible():
            $VBoxContainer/error_username.hide() # username hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_password.show()
    elif db.query_result.size() >= 1 and db.query_result[0]["Username"] != Username and db.query_result[0]["Password"] == Password:
```

```
44          if $VBoxContainer/error_not_found.is_visible():
45              $VBoxContainer/error_not_found.hide() # login not found hide
46          if $VBoxContainer/error_password.is_visible():
47              $VBoxContainer/error_password.hide() # password hide
48          $VBoxContainer/error_label_msg.show()
49          $VBoxContainer/error_username.show()
50      else:
51          print("ERROR CODE_1: UNKNOWN")
52
53  func _on_QuitButtonFunc_pressed():
54      get_tree().quit()
55
56  """
57  #SIGN UP FUNC - DO AT END
58  func commitDataToDB(): # Function to call the database
59      db = SQLite.new() # Creates a new command
60      db.path = db_name # Links the SQLite database to godot for use
61      db.open_db() # Opening the database
62      var tableName = "PlayerCredentials" # Declares variable to the table in the pre-existing db
63      var dict : Dictionary = Dictionary() # Declares variable for godot to call the fields of the db
64      dict["Username"] = "Scro" # Username Example
65      dict["Password"] = "ScroThePro" # Password Example
66
67      db.insert_row(tableName,dict) # Inserts the new data into the corresponding fields
68
69  """
70
71  """
72  #Getting text from the user
73  func _on_Usernametype_text_changed(NameofUser):
```

MainMenu.gd:

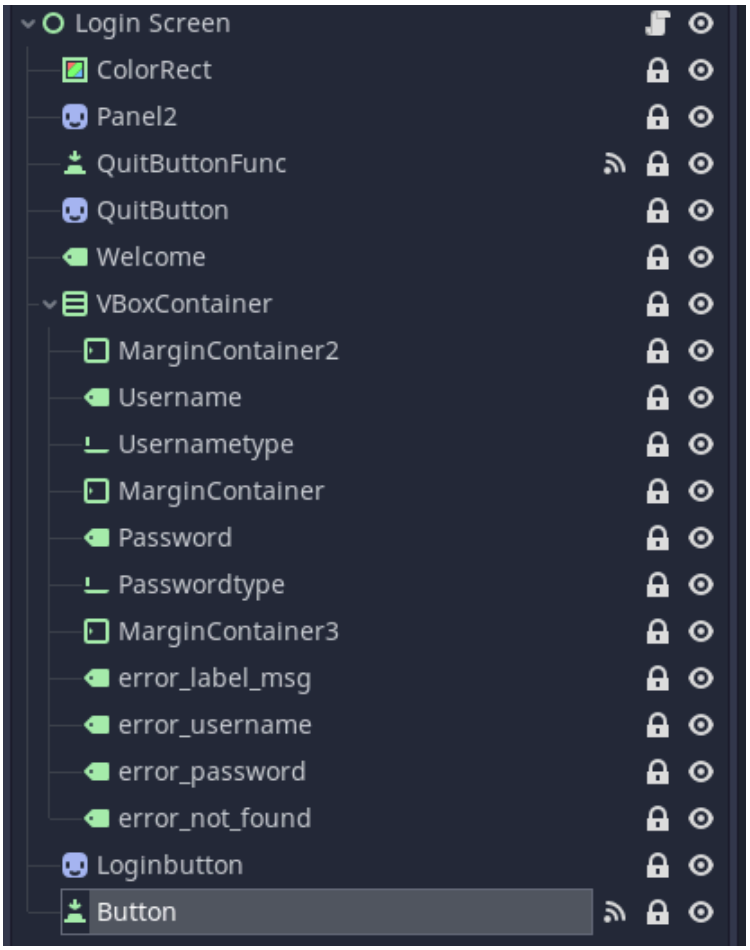
```
extends Control

func _ready():
    pass # Replace with function body.

func _on_Buttonquit_pressed():
    get_tree().quit()

func _on_PlayButton_button_up():
    get_tree().change_scene("res://Scenes/Play.tscn")

func _on_LeaderboardButton_button_up():
    get_tree().change_scene("res://Scenes/Leaderboard.tscn")

func _on_ProfileButton_button_up():
    get_tree().change_scene("res://Scenes/Profile.tscn")

func _on_SettingsButton_button_up():
    get_tree().change_scene("res://Scenes/Settings.tscn")

```

QuitButton.gd

```
extends Sprite


func _ready():
    pass # Replace with function body.


func _on_Button_pressed():
    get_tree().quit()

```

2D UI:

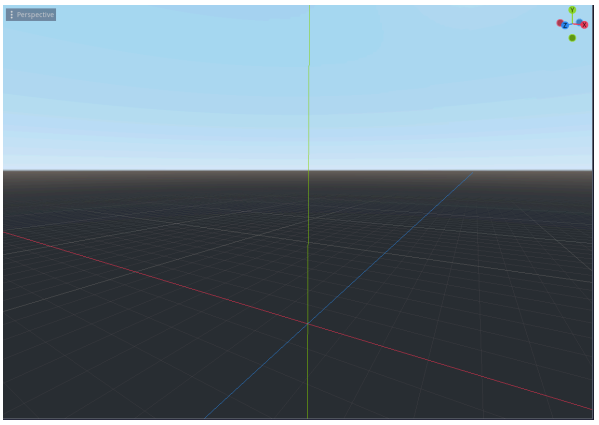
Parent and Children Nodes (of the Scene):

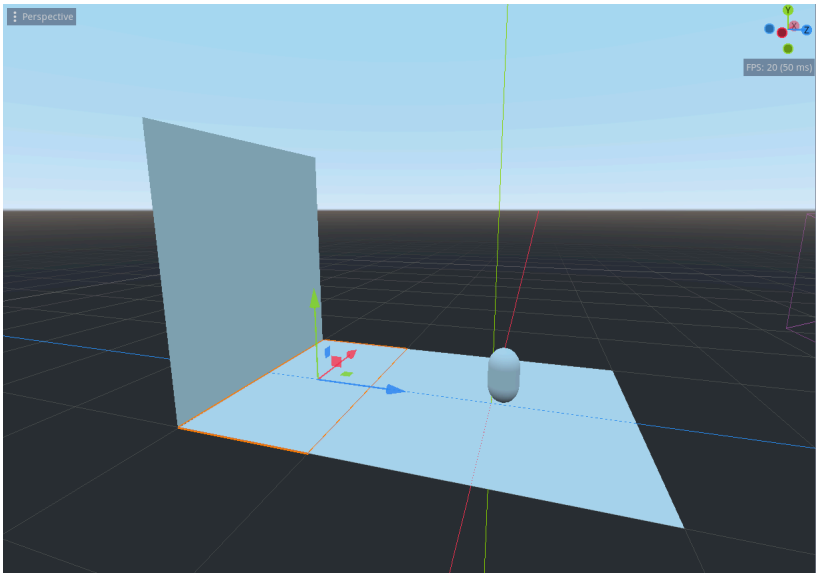


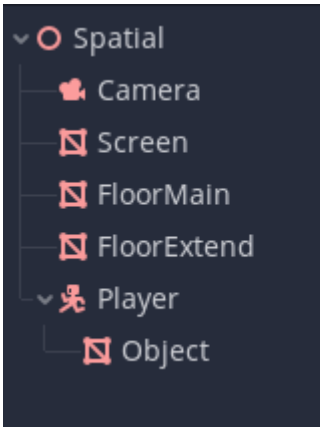
Phase 2 - Game mode Creations

Before proceeding on with the development, I need to take into consideration my limitations of what I will be able to achieve. As mentioned in my analysis stage, I will only plan to complete 2 different Aim training algorithms, using the same template background, cubes, and point of view of the character. I will need to also make sure the aim trainer would obviously be playable, but I don't think that backgrounds and scenery would be necessary because it provides no use in the purpose of the game.

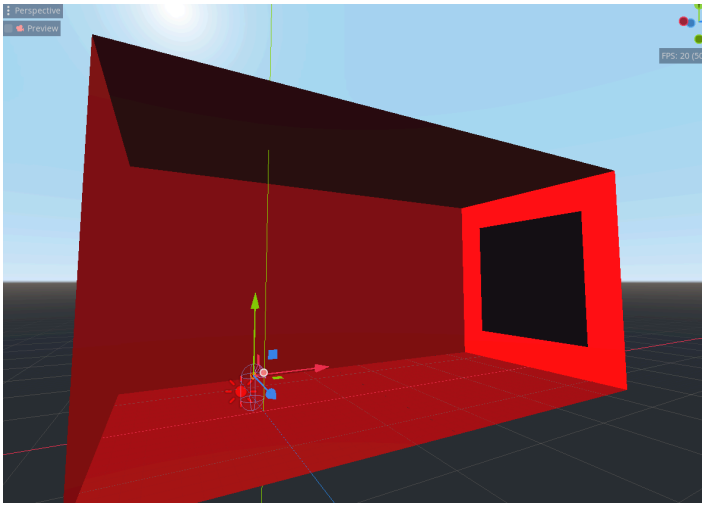
Step 1: Making the 3D playing area (Background)

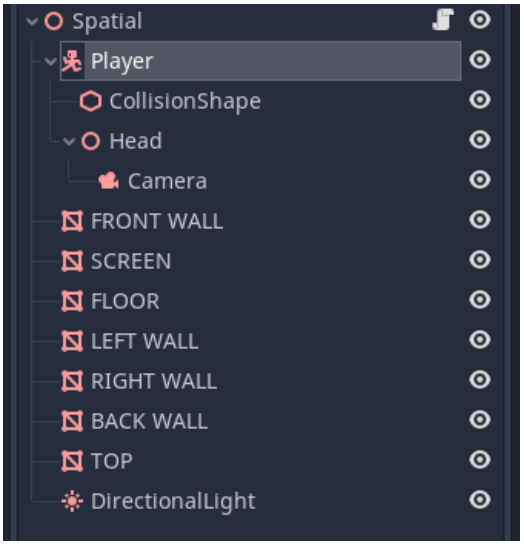


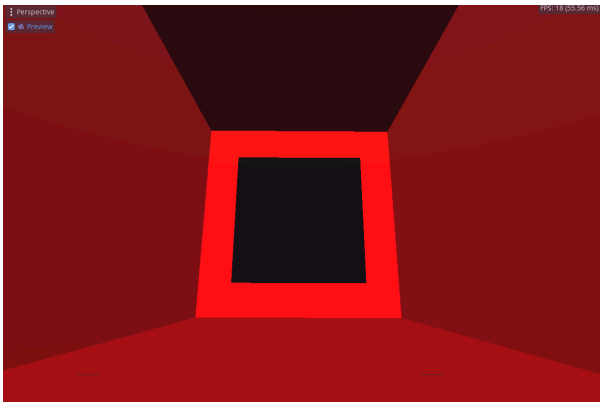




Firstly, I created my game scene which creates the overall base prototype for all of the different game modes. Now I will be able to add my hit objects that will appear in logical positions on the screen, with potentially different difficulties/boosters that might give the player a multiplier on their score.



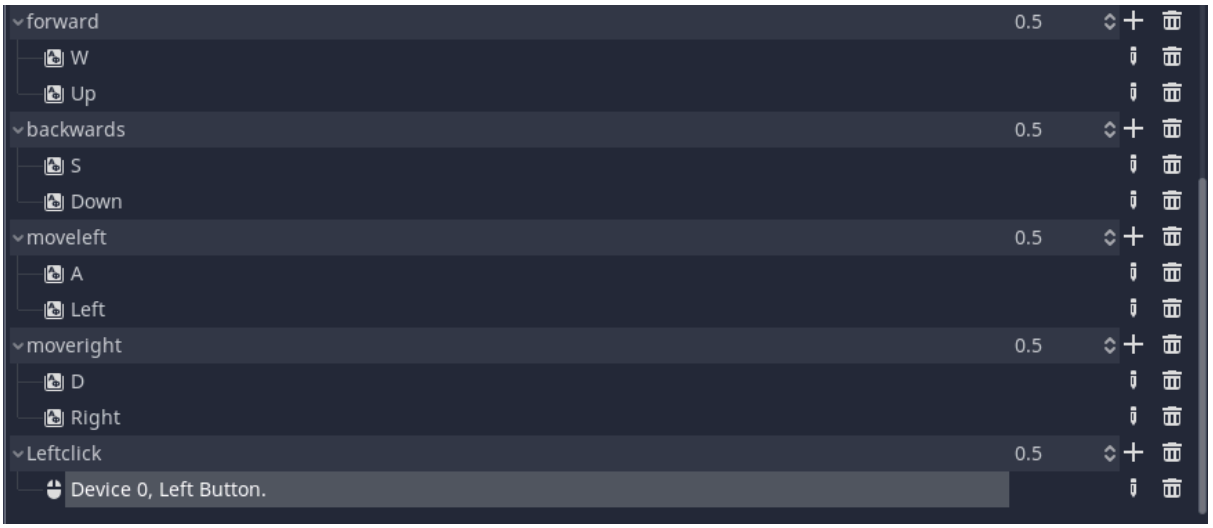




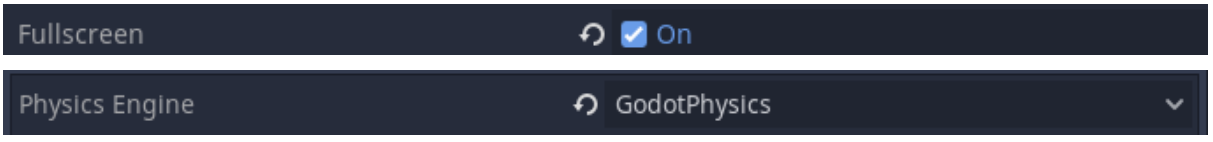
As this game is going for functionality, I think the best course of action would be to place the player within a cube, so their focus would be on the black screen in front of them, with white objects to hit for the complementary effect to make the players hit the objects with maximum efficiency. Additionally, with the collision shape, I will be able to send a signal from the player parent node to the main script, so I will be able to code the player to not phase through the walls and be able to hit the objects on the screen. I have used this video as reference.

Step 2- Setting up the movement mechanics and customising project settings

For godot, there is a separate input map that is within the project settings, which allows you to call the action name, to move correctly within the project. For example, in pygame, it would require a “event.key == pygame.K_UP:”, but in godot, this can be called in a function with the variable “forward” in this case.



This allows me to simply add the WASD and UP, Down, Left and Right keys on the keyboard, where both of the inputs will do the same thing, so this is just for preference for the user.



Step 3 - Adding the camera and player movement

```gdscript
extends KinematicBody

var look = Vector3.ZERO
onready var HEAD = $Head/Camera
export var sensitivity = 0.1
export var max_angle = 90
export var min_angle = -75

func _ready():
    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED) # Captures where the mouse is on the screen

func _physics_process(delta):
    HEAD.rotation_degrees.x = look.x # rotation for x direction
    HEAD.rotation_degrees.y = look.y # rotation for y direction

func _input(event):
    if event is InputEventMouseMotion:
        look.y -= (event.relative.x * sensitivity) #Having the y vector - the x relative to the screen for each input
        look.x -= (event.relative.y * sensitivity) #Having the x vector - the y relative to the screen for each input
        look.x = clamp(look.x, min_angle, max_angle) #Setting a cap to how far the player can look in either y direction (up/down)

```

Developing and adapting towards my program, I have used this to help me develop my code to allow the player to move the camera with a normal sensitivity. Additionally, I made sure to comment on the code to remind myself when debugging so I can remember in the future if I ever need to fix a problem. The datatype of Vector3 represents the position within the 3D space by doing mathematical calculations. This allows me to change the camera angle by moving my mouse due to the declaring the var “look” to move accordingly depending on the position of the mouse.

```gdscript
func _physics_process(delta):
    HEAD.rotation_degrees.x = look.x # rotation for x direction
    HEAD.rotation_degrees.y = look.y # rotation for y direction
    input_dir = Vector3(Input.get_action_strength("moveright") - Input.get_action_strength("moveleft"),
    0,
    Input.get_action_strength("backward") - Input.get_action_strength("forward")).normalized().rotated(Vector3.UP, rotation.y)

    speed.x = input_dir.x * rateofs
    speed.z = input_dir.z * rateofs

    speed = move_and_slide(speed)
```

Adding the movement, this section of code helped move the character depending on what input the user is doing. However, if I change my camera position, it does not correspond with what the user is inputting, therefore I found a problem. (Test 1)

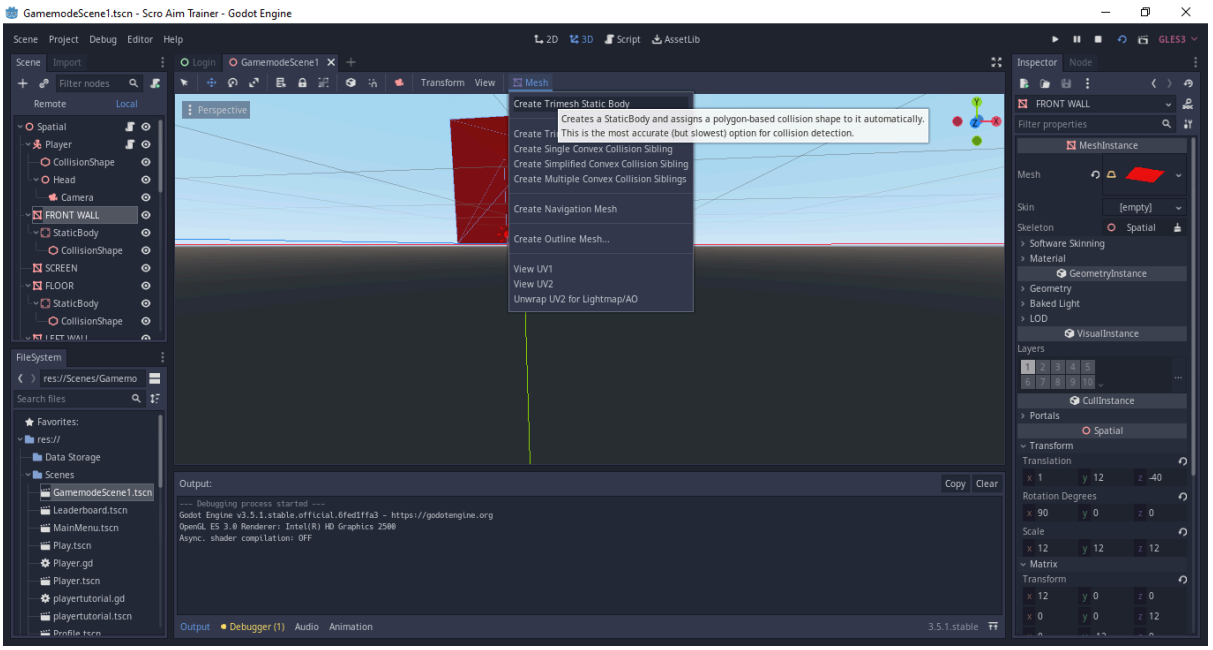
To fix this, I would need to require a different method that is different from recording the addition and subtraction of the x and z-axis. For this, I simplified the calculation to get the axis from the get_axis() method, and make use of the transform methods that I learned from this video.

```
var input_dir := Input.get_vector("moveleft", "moveright", "forward", "backward")

var direction = (CAMERA.transform.basis.rotated(Vector3.UP, rotation.y) * Vector3(input_dir.x, 0, input_dir.y)).normalized()

if direction:
    velocity.x = direction.x * rateofs
    velocity.z = direction.z * rateofs

else:
    velocity.x = move_toward(velocity.x, 0, rateofs)
    velocity.z = move_toward(velocity.z, 0, rateofs)

velocity = move_and_slide(velocity, Vector3.UP)
```

I have changed the HEAD variable to be known as the CAMERA variable now because it makes more sense within the program and debugging. Additionally, I simplified the statement that was retrieving the input data to a simple get_vector() method, as I can do the calculations later within the statements. The .transform.basis is a godot feature that determines the orientation of the camera and position. As such, I was able to link the camera rotation to the input value, enabling movement in that particular direction. With the move_and_slide() function, it now gives access for my body to move along the vector axis'.

Step 4 - Collision with walls/exiting the game

Right now, my character can phase through the walls, which they shouldn't be able to. As such, I need to create a function to calculate collision detection.

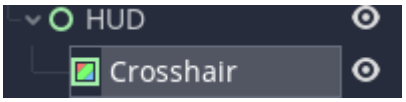


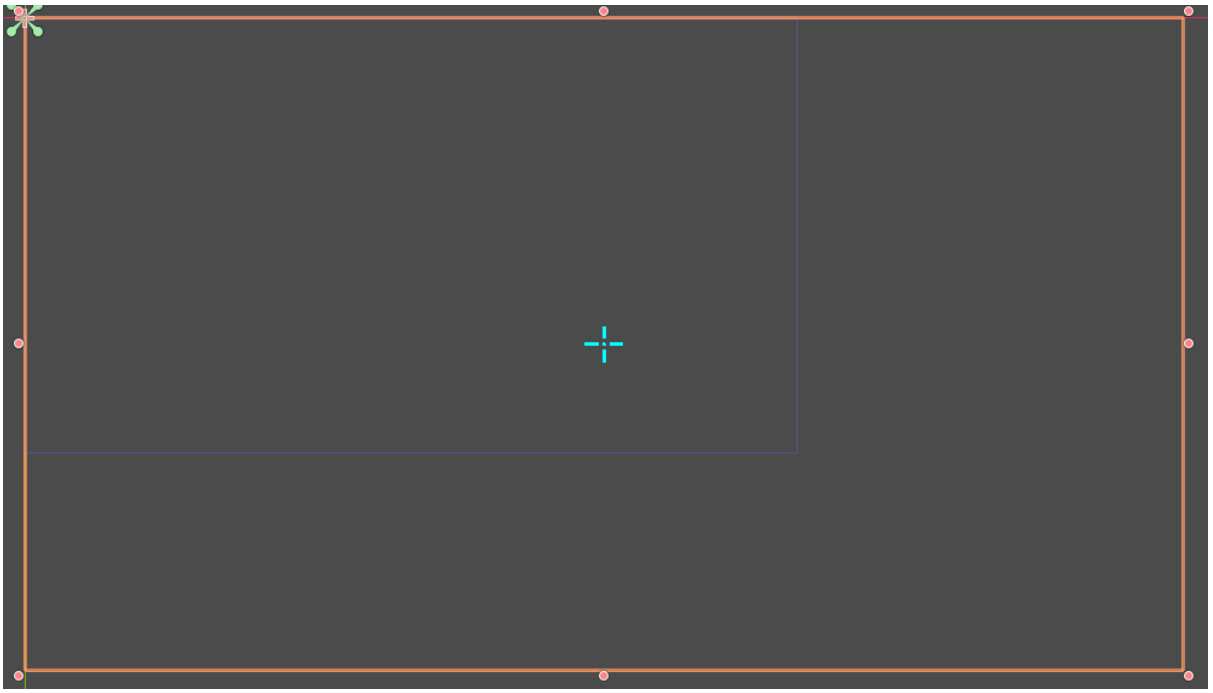
Thus, I applied a trimmesh static body to each of the individual plane meshes, so the player's collision body can collide with the collision body of the plane to stop it from going through the wall.

As I'm currently in my testing phase, I need to create a quick and easy exit that utilises the "Esc" key on the keyboard.

```
if Input.is_action_just_pressed("esc"):
    get_tree().quit()
```

Step 5 - Brief Crosshair UI and Confirmations

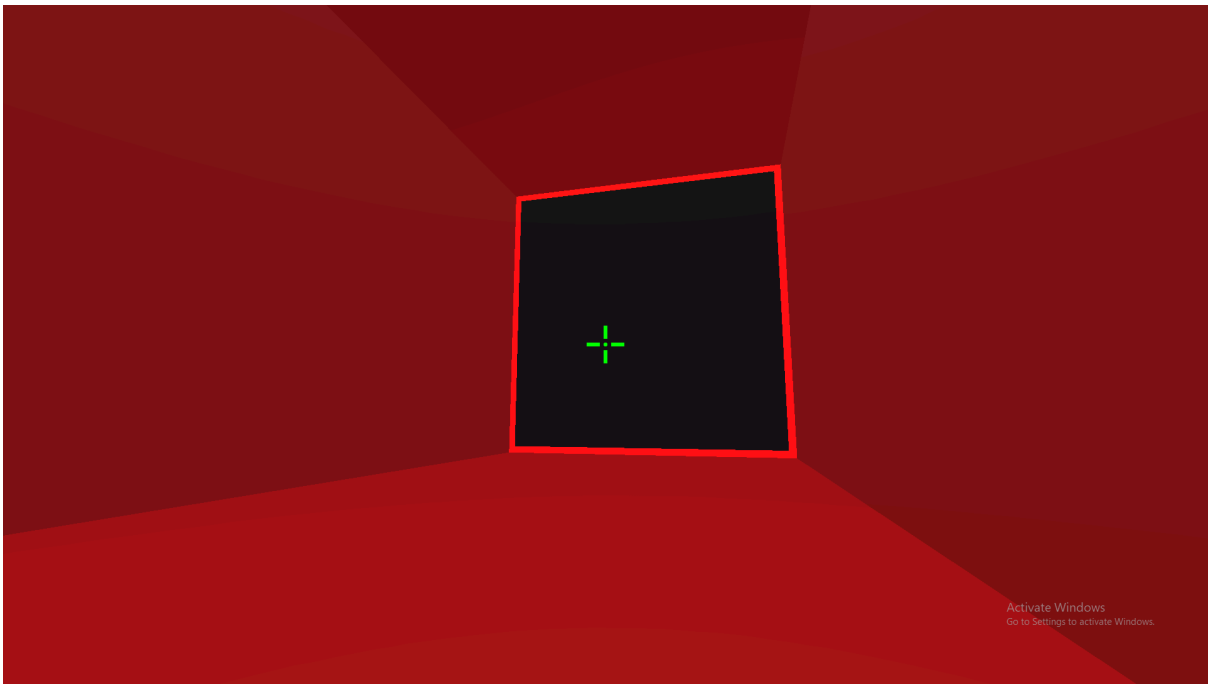




By utilising the crosshair section of this provided, I customise a 2D HUD for my crosshair, which will be changeable within my settings section. With this, I created a new UI node to hold my crosshair and the colourect object is using the shader material that was also given by the script. To make it show up on the screen of my camera, I will need to call it within a function of my player script with reference to the camera using singletons and variables.

```
func _update_hud():
    var cursor = $Head/Camera/RayCast.get_collider()
    if cursor:
        $HUD/Crosshair.material.set_shader_param("color_id", 0)
```

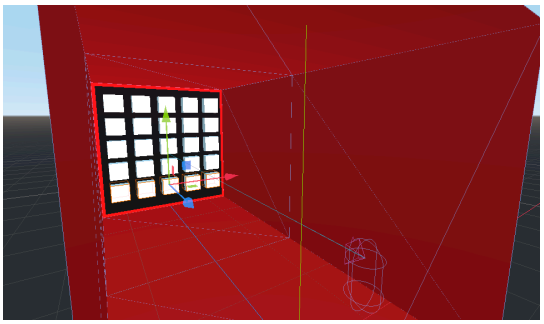
```
func _physics_process(delta): #delta
    
    _update_hud()
    
```

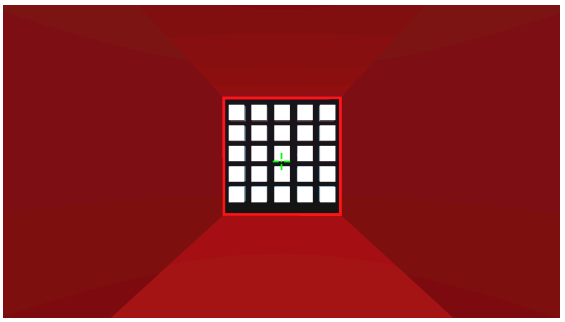


For my first game mode, I will not require to change the colours or shape of the crosshair, but I will need to require collision detection between the cubes that will appear on the screen and the crosshair to function. As such, I only need to set up one crosshair to test out the functionality of hitting the objects right now - the rest of the crosshair's features will be mentioned in the settings phase.

Step 6 - Objects and hit detection

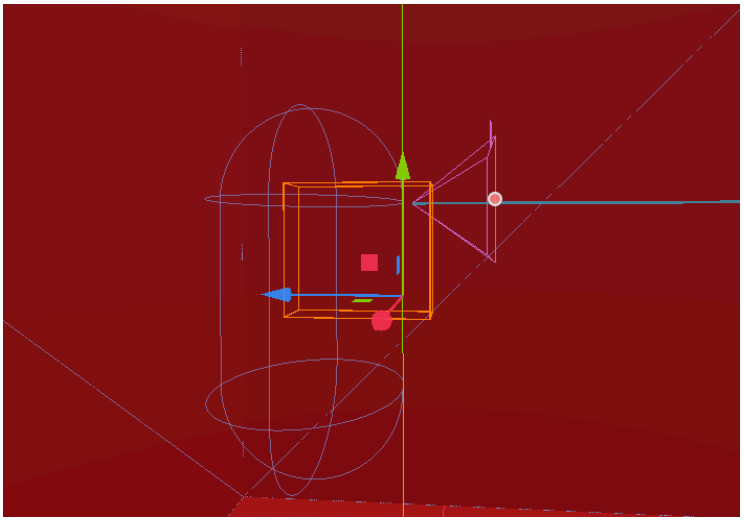
So now that we have done the foundation of all of the player's movement and playing space, I will now implement the objects that the user would "collide" with using the crosshair. In this game, I am going to use cubes that have 2 layers of hit detection, where the centre of the cube would give more points (like a bullseye) and the outside content would give less.





```
if Input.is_action_just_pressed("Leftclick"):
    if aimcast.is_colliding():
        var target = aimcast.get_collider()
        if target.is_in_group("Cubes"):
            print("registering")
            target.queue_free()
```

I made the gridshot layout from my design section and added the “left click” to my input map, to represent the left click of the mouse. Also, allocating all of the cubes into a singular group named “Cubes”, allows me to detect when the aimcast is colliding with the cubes - The aim cast is the raycast feature which allows collision detection in a 3D space. However, when I ran the code shown above, there was no error or input of the cubes into the code. I thought this initially meant one of 2 things, the cubes aren’t placed into the group correctly or the raycast isn’t colliding with the cubes. But after doing some tests (Test 2), I concluded that the issue wasn’t either of the problems I mentioned previously. It was much simpler. The raycast was colliding with the Player collision shape first, so the cubes were not even in the function in the first place. I fixed this simply with moving my camera forward, so the ray cast would never hit my collision shape as seen below. (The collision shape is still important so the player does not go through the red walls or the floor)

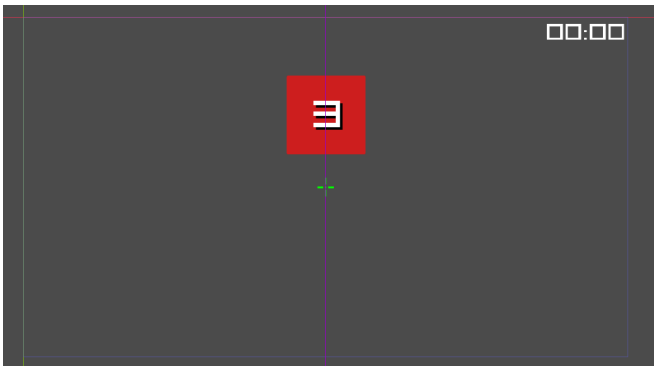


Now, the cubes are disappearing as they are intended to from the left click input.

Step 7 - Timer and score importance in the algorithm

To make the algorithm, I would need a changing multiplier that can be used to differentiate people who are fast and accurate at clicking the cubes, from the people who are clicking them slow and accurately. Because this is the gridshot game mode, reaction time and speed are the essential aspects in which the player is trying to get better at. As such, the multiplier will be the time taken from the user to click **another** cube from the last one (the first cube multiplier will be set at a constant of 1), and if takes less than 0.01 seconds, then it is a +100 to the constant, if its less than 0.025 seconds, then its a +50, and so on for each individual cube the user clicks on within 60 seconds.

2D preview:

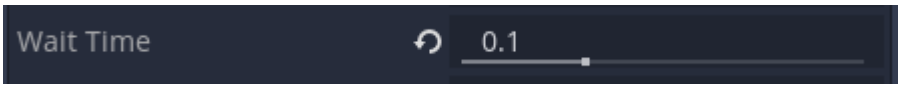


In-game preview:

I have created some graphics so that the user will be able to see the timer and countdown. I plan to have the countdown timer (in the middle) countdown from 3 to 1, and the timer in the top right corner will countdown from 60.

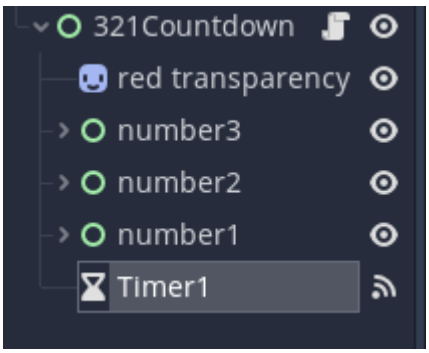

```gdscript
extends RichTextLabel

var ms = 0
var s = 60

func _process(delta):

    if ms < 1:
        s -= 1
        ms = 9

    set_text(str(s)+":"+str(ms))

func _ready():
    pass
```

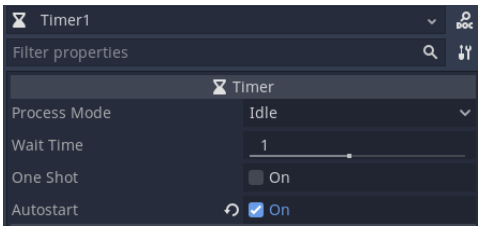
Now attaching a script to the playing timer gives me the ability to change the text of the playing timer to countdown from 60. However, I ran into a problem. The video that I was using to get inspiration from, only helps when I'm making a usual timer counting up, but not when it is counting down. As such, I began to rewrite the code. (Test 3).



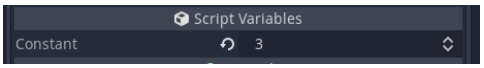
```
extends Control

export (int) var seconds = 0
var dsec = 0

func _ready():
    pass

func _physics_process(delta):

    if seconds > 0 and dsec <= 0:
        seconds -= 1
        dsec = 10

    if seconds >= 10:
        $Seconds.set_text(str(seconds))
    else:
        $Seconds.set_text("0"+str(seconds))

    if dsec >= 10:
        $DSeconds.set_text(str(dsec))
    else:
        $DSeconds.set_text("0"+str(dsec))


func _on_Timer_timeout():
    dsec -= 1
```

In this new section of code, I made the seconds have an export value so I can customise the seconds in the inspector instead of changing the code. Then, I separated the seconds and milliseconds into 2 separate objects so I can reference them using singletons to act as the visual timer. Now, using the timeout function, I can continuously loop the value being subtracted by a millisecond, and this timer will only start when I allow it to through the main game scene script. For this, I will need to implement code into my 3,2,1 Countdown to start the game.






```
27
28 func _on_Timer_timeout():
29     constant -= 1
30
```

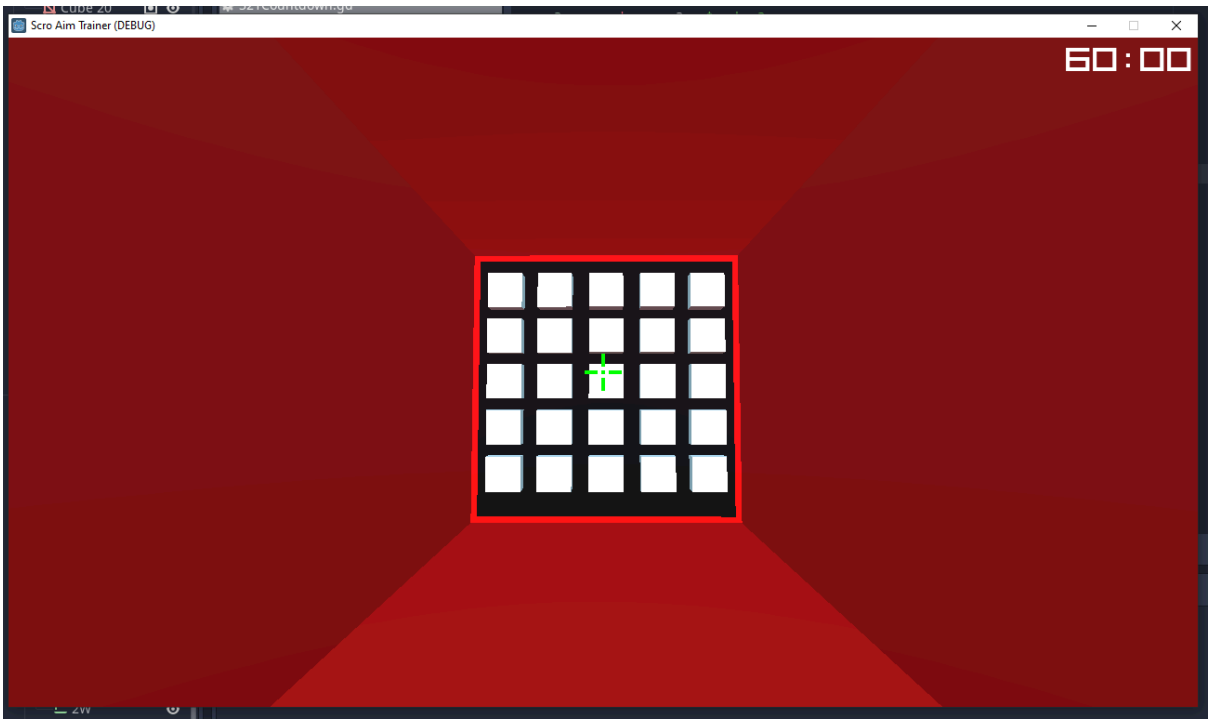



```
1  extends Control
2
3  onready var n3 = $number3
4  onready var n2 = $number2
5  onready var n1 = $number1
6  export var constant = 0
7
8  func _ready():
9      n3.hide()
10     n2.hide()
11     n1.hide()
12
13 func _process(delta):
14
15     if constant == 3:
16         n3.show()
17     elif constant == 2:
18         n3.hide()
19         n2.show()
20     elif constant == 1:
21         n2.hide()
22         n1.show()
23     elif constant <= 0:
24         n1.hide()
25         $"red transparency".hide()
26         $Timer1.stop()
27
28 func _on_Timer_timeout():
29     constant -= 1
```

In this script, I have assigned the variables of each of the countdown numbers and I created a variable named “constant”. This will help save time in the future because if I need another countdown, I can use this same script/code and just change the constant value to countdown from 5 or 10, meaning that this is reusable. This process is set to “autostart”, meaning once the scene has been opened, it will automatically activate this countdown timer through a bunch of loops by utilising the constant value. Additionally, I made sure to stop the timer when the constant value is less than or equal to 0 because it will prevent a waste in memory space. However, when I ran the code, it didn’t work.

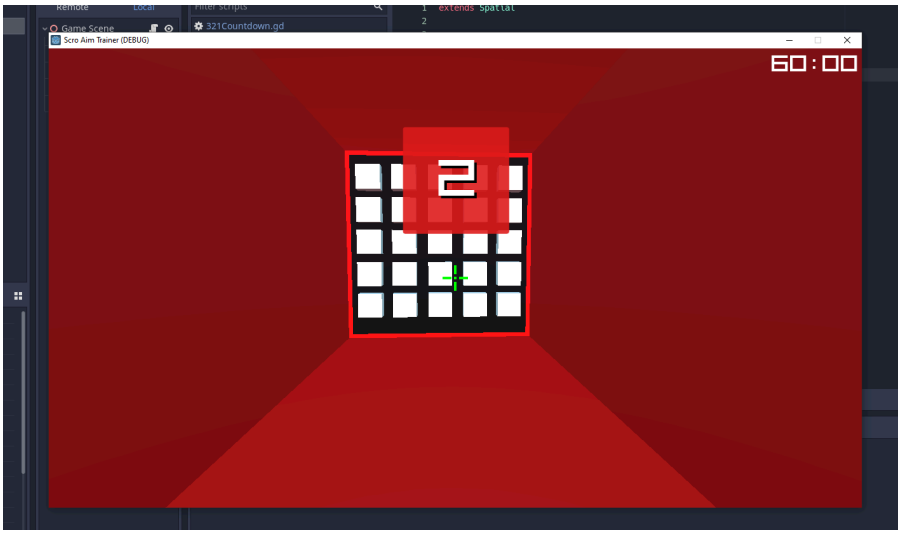
```
extends Control

export (int) var seconds = 0
var dsec = 0
var counter = 18.0
var n = 1.0

func _ready():
    $Timer2.wait_time = 3.0
    $Timer2.start()

func _physics_process(delta):
    if counter - n * $Timer2.wait_time >= 0.0:
        $Seconds.set_text(str(60))
        $DSeconds.set_text("0"+str(0))
        n += 1.0
    else:
        $Timer2.wait_time = 0.1
        calc()

func calc():
    if seconds > 0 and dsec <= 0:
        seconds -= 1
        dsec = 10

    if seconds >= 10:
        $Seconds.set_text(str(seconds))
```

Now after writing the 2 scripts for the timers, I need to make sure that the PlayingTimer starts after the 321 Countdown script. Additionally, at this moment the text displays as “59.9” for the PlayingTimer, which is due the calc() function in the PlayingTimer happening whilst the countdown in happening To prevent this, I made up a simple if statement that does a loop consisting of setting the wait_time of the clock to 3.0 seconds, to sync up with the end of the countdown timer.



I was really perplexed by this because I had just reorganised all of my scripts and removed all the unnecessary items that I previously used, but not anymore. I detached and reattached the scripts to move the scripts and scenes to the correct location, but this was my critical flaw. If the script is removed, the exported values in the inspector section do not save. This means that all I needed to do was to make the constant value 3 again and it all worked fine and handy. (Test 4)

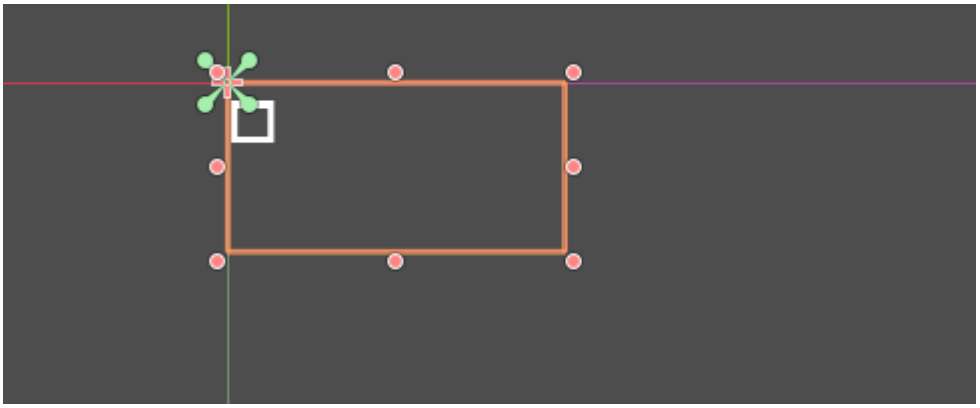
Reflection: Prototype and comments from the stakeholders



As I've finished my timer, I have shared this with my stakeholders. From my secondary stakeholder, he had thought of what would happen when the playing timer hit 0. Because of the long duration, I had not thought of this but since I let the timer run, it began going into negative time. To fix this, currently I have wrote this line of code:

```
if seconds == 0 and dsec == 0:
    get_tree().quit() #this should change to the result page later on
```

Right now as it is during the development stage, I only need to let it run for 60 seconds to test everything is running correctly, so I only need to make the program quit at this moment..



Now, I have done the score counter for the user to visually see. This process was done in the same way the PlayingTimer objects were done, but this one was easier because there is only 1 number to update instead of 2.

```
var hit_target = aimcast.get_collider()

if hit_target.is_in_group("Cubes") and hit_target.is_visible():
    print ("HIT VISIBLE CUBE")
    hit_target.hide()
    var time_left = int(($PlayingTimer/Seconds.text) + "." + ($PlayingTimer/DSeconds.text))
    if vartimer - time_left <= 0.1:
        score += 100
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    elif vartimer - time_left <= 0.25:
        score += 50
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    elif vartimer - time_left <= 0.5:
        score += 10
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    else:
        score += 1
        $Score/ScoreCount.set_text(str(score))
```

```
export (int) var vartimer = 59.1
```

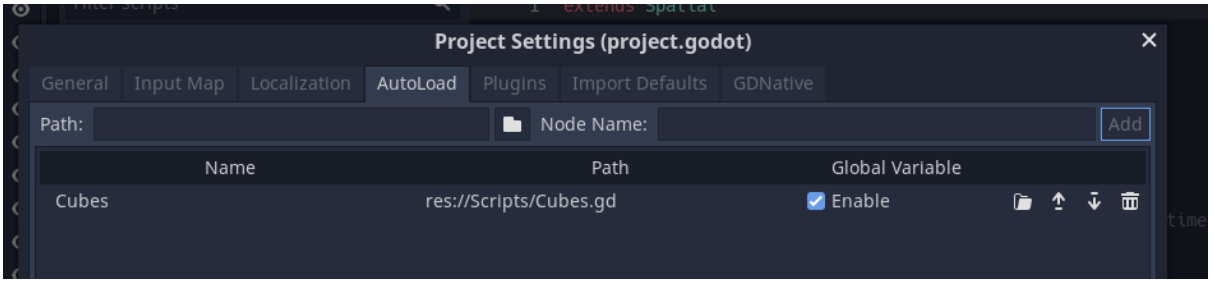
This is the algorithm that I have done within the Player script to increment the score by the factor of if the player has hit the cubes fast enough. I have made new variables called “vartimer”, “time_left”, and “score” to hold the values that are constantly changing in the game. The reason why the vartimer is set to 59.1 is because my time_left variable would start at “59.09”, so 59.1 is 60 seconds in this case. However, after a while it would stop getting the big values of the score when I was clicking quickly which tells me that the vartimer is not changing how I intended it to do. (Test 5)

```
if hit_target.is_in_group("Cubes") and hit_target.is_visible():
    print ("HIT VISIBLE CUBE")
    hit_target.hide()
    var time_left = float(($PlayingTimer/Seconds.text) + "." + ($PlayingTimer/DSeconds.text))
    if vartimer - time_left <= 0.01:
        score += 100
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    elif vartimer - time_left <= 0.025:
        score += 50
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    elif vartimer - time_left <= 0.05:
        score += 25
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    else:
        score += 10
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
```

After doing some debugging, I found out that the vartimer was not changing at all when it was hitting the lowest value. This results in me finding out that there was no new assignment to change the value of “vartimer”, so my vartimer would always be the same after that.

Step 8 - Making the game mode “Gridshot”

To finish off the game mode, I will need to have the cubes hide and be visible repeatedly once the player has collided and left clicked them. For this, I will have only 3 cubes displayed at one time, like my inspiration of “Aim Labs”, at random so I will need to make a randomizer and make a separate Cubes script. However, I cannot simply reference Cubes in my Player script because the Cubes cannot be a child of the Player node as they will move when the player also moves, which is not ideal. As I thought of ways around this, I thought all I need to do is make my Cubes script local with the global variable feature within the project settings.



As seen above, I have added my Cubes script to my Project Settings. Now, I can reference them from wherever I want within the project, especially in the Player script.

However, this method did not work for what I was planning to do. (Test 6) This makes the “Cubes” node have a complete relationship with the Gamescene node, so whenever I tried to run it, It showed this message.

```
0:00:00.457        get_node: (Node not found: "Cube10/Cube 10" (relative to "/root/Cubes").)
```

This means that my program was unable to detect the Cubes, so this way was unnecessary in the first place. As such, I realised that I did not need to reference it in the Player Script, I could just do this algorithm in the main Game Scene Script separately.

```
extends Spatial

onready var AllCubes = [$Cubes/Cube1,$Cubes/Cube2,$Cubes/Cube3,$Cubes/Cube4,$Cubes/Cube5,$Cubes/Cube6,$Cubes/Cube7,$Cubes/Cube8,
$Cubes/Cube9,$Cubes/Cube10,$Cubes/Cube11,$Cubes/Cube12,$Cubes/Cube13,$Cubes/Cube14,$Cubes/Cube15,$Cubes/Cube16,$Cubes/Cube17,$Cubes/Cube18,
$Cubes/Cube19,$Cubes/Cube20,$Cubes/Cube21,$Cubes/Cube22,$Cubes/Cube23,$Cubes/Cube24,$Cubes/Cube25]
onready var rng = RandomNumberGenerator.new()
var value = 0
```

```
func _process(delta):
    if $"Player/321Countdown/Timer1".is_stopped():
        rng.randomize()
        value = rng.randi_range(0, 24)
        if AllCubes[value].is_visible():
            pass
        else:
            AllCubes[value].show()
```

I have currently created an initial draft of my algorithm using the random feature that godot has, where it randomises one of my cube numbers in the array in this case, and I have made it repeat continuously throughout the game. However, I don't think this is what I intended to create in my success criteria. I wanted to only have a few cubes visible at one time, so the player has to utilise the entire area of the screen. In this case, I think I will code it so it would only show another cube once a cube that is visible is clicked. This means that I will need to create a new “buffer” variable.

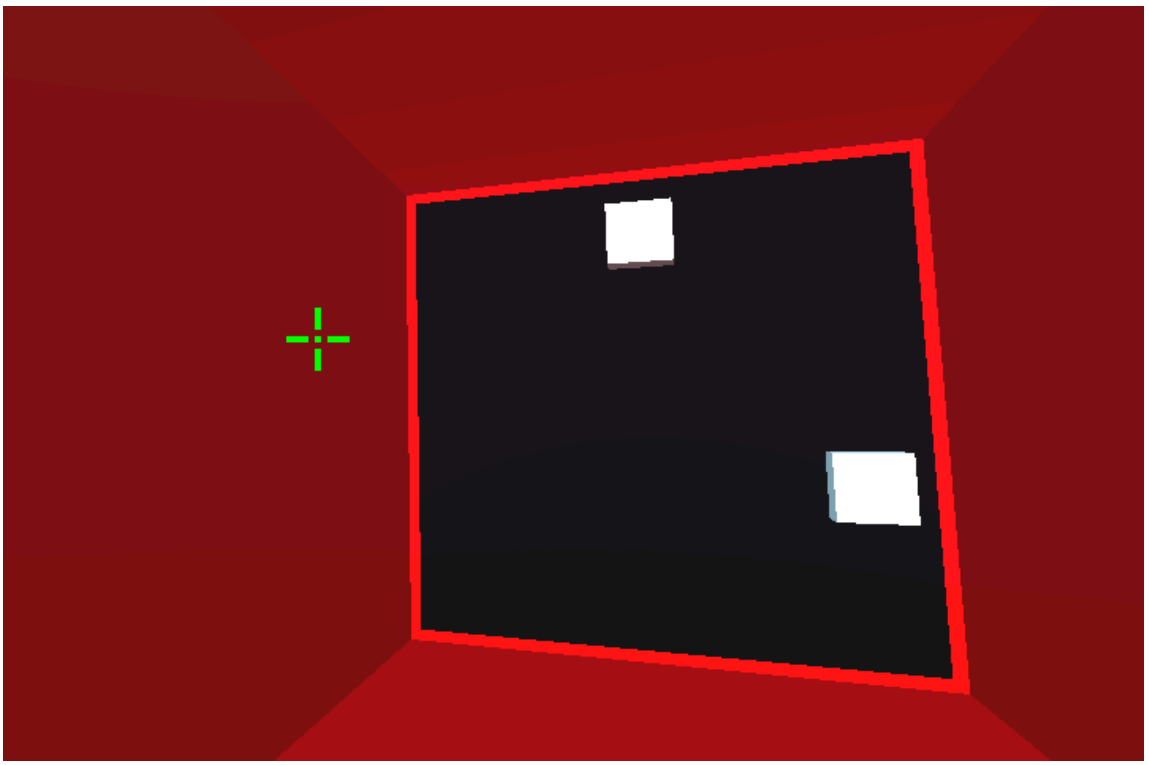
```
if Input.is_action_just_pressed("Leftclick"):
    if aimcast.is_colliding() and $"321Countdown/Timer1".is_stopped():
        var hit_target = aimcast.get_collider()
        if hit_target.is_in_group("Cubes") and hit_target.is_visible():
            print ("HIT VISIBLE CUBE")
            get_parent().buffer = true
```

In my Player script, the buffer variable is retrieved from its parent node’s script which is called using the “get_parent()” feature that godot has.

```
var repeatvalue = 0
var buffer = false

func _process(delta):
    if $"Player/321Countdown/Timer1".is_stopped() and buffer == true:
        rng.randomize()
        repeatvalue = rng.randi_range(0, 24)
        if AllCubes[repeatvalue].is_visible():
            pass
        else:
            AllCubes[repeatvalue].show()
            buffer = false
```

Since the buffer value in the Main Game Scene script is set to false, it won't be called until the player has clicked onto a cube successfully. The buffer variable is now implemented into the algorithm, where if buffer == true, the RandomNumberGenerator will randomise and generate a number between 0 and 24. If the Cube is already visible, it will pass and the if statement will repeat because the buffer variable is still set to true. If the cube is not, then it is a hidden cube, so it will show and set the buffer to false.

```
func _ready():

    rng.randomize()
    AllCubes[rng.randi_range(0, 24)].show()
    rng.randomize()
    AllCubes[rng.randi_range(0, 24)].show()
    rng.randomize()
    AllCubes[rng.randi_range(0, 24)].show()
```



A mini-problem did arise when I was testing the program as there were cases of only 2 or even 1 cube spawning at the start. This is a problem because it would put that particular user at a disadvantage compared to other cases, so this must be fixed. (Test 7)

```
	rng.randomize()
	value1 = AllCubes[rng.randi_range(0, 24)]
	value1.show()

	rng.randomize()
	value2 = AllCubes[rng.randi_range(0, 24)]
	while value2 == value1:
		rng.randomize()
		value2 = AllCubes[rng.randi_range(0, 24)]
	value2.show()

	rng.randomize()
	value3 = AllCubes[rng.randi_range(0, 24)]
	while value3 == value2 or value3 == value1:
		rng.randomize()
		value3 = AllCubes[rng.randi_range(0, 24)]
	value3.show()
```

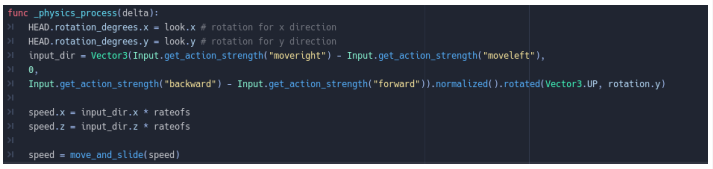
```
7  var value1 = 0
8  var value2 = 0
9  var value3 = 0
```

All I needed to do was to put some while loops to compare the new values to the old values, to see if they were the same. If there was, it would continue to loop until the value is different, where it would end the while loop and then show that value of the cube. Now, it has no way of ever showing less or more cubes than 3.

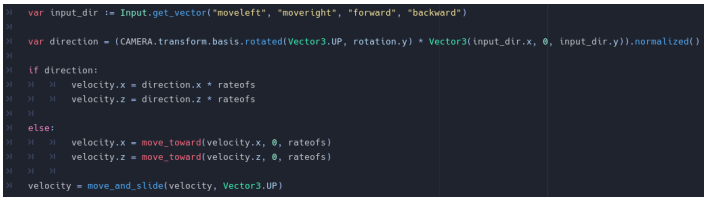
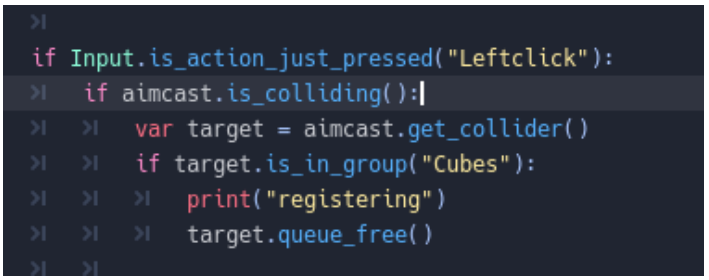
```
if aimcast.is_colliding() and $"321Countdown/Timer1".is_stopped():
```

Additionally, the “left click” input collision will only happen once the countdown has finished. This means that the player can freely roam in the 3 seconds they have to a preferred position of theirs.

Testing Phase 2

ID (T)	Description	Code/Input
1	Moving the character forward and backwards, according to the calculations done by the engine.	
	Result	
	 PASS	

```
func _physics_process(delta):
    HEAD.rotation_degrees.x = look.x # rotation for x direction
    HEAD.rotation_degrees.y = look.y # rotation for y direction
    input_dir = Vector3(Input.get_action_strength("moveright") - Input.get_action_strength("moveleft"),
    0,
    Input.get_action_strength("backward") - Input.get_action_strength("forward")).normalized().rotated(Vector3.UP, rotation.y)

    speed.x = input_dir.x * rateofs
    speed.z = input_dir.z * rateofs

    speed = move_and_slide(speed)
```

	Fix?	To fix this, I would need to require a different method that is different from recording the addition and subtraction of the x and z-axis. For this, I simplified the calculation to get the axis from the get_axis() method, and made use of the transform methods that I learned from this video and applied it to my program. New Code:  I have changed the HEAD variable to be known as the CAMERA variable now because it makes more sense within the program and debugging. Additionally, I simplified the statement that was retrieving the input data to a simple get_vector() method, as I can do the calculations later within the statements. The .transform.basis is a godot feature that determines the orientation of the camera and position. As such, I was able to link the camera rotation to the input value, enabling movement in that particular direction. With the move_and_slide() function, it now gives access for my body to move along the vector axis'.
2	Left click should remove the cubes, but nothing is sent to the console or changed.	 FAIL
	Result	

```
if Input.is_action_just_pressed("Leftclick"):
    if aimcast.is_colliding():
        var target = aimcast.get_collider()
        print("registering")
        if target.is_in_group("Cubes"):
            print("registering")
            target.queue_free()
```

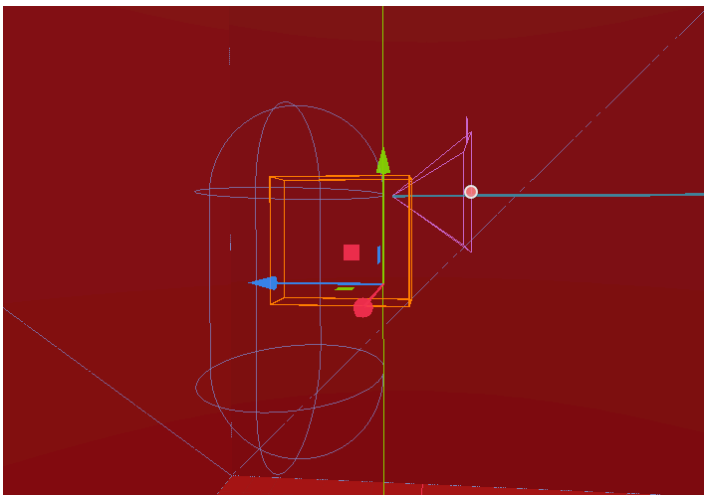
Added another print statement to see if the game is recognising my left click.

```
registering
registering
registering
registering
registering
registering
registering
registering
registering
--- Debugging process stopped ---
```

This makes the aimcast valid.

PASS

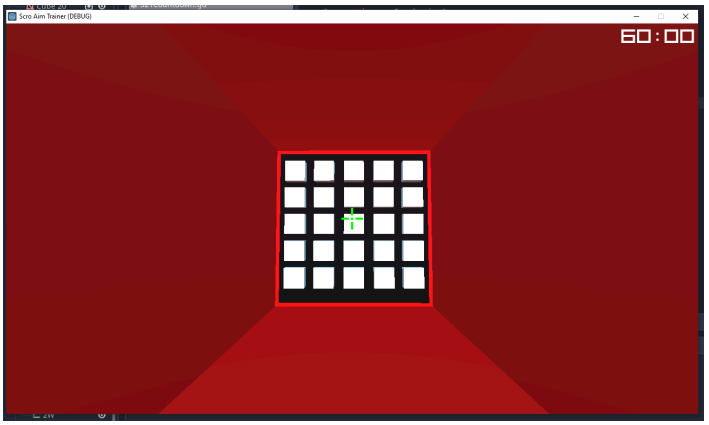
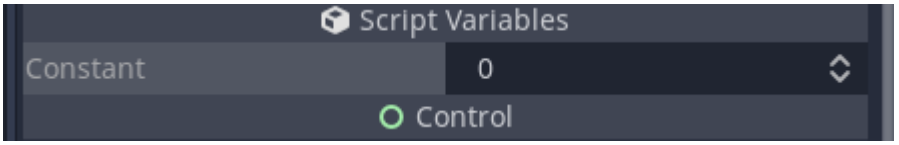
Fix?	I solved this by changing the placement of my aimcast, as it was interfering with the collision shape of itself, which made it not a part of the group Cubes. The raycast was colliding with the Player collision shape first, so the cube objects were not even in the function in the first place. I fixed this simply by moving my camera's x-position forward, so the ray cast would never hit the user's own collision shape as seen below. (The collision shape is still important so the player does not go through the red walls or the floor)

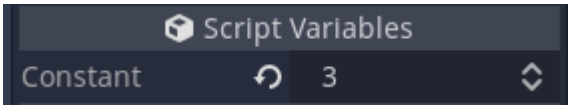
		
3	This timer only helped me when I was counting down so I need to change it to be different because my timer is not starting at 0.	1 extends RichTextLabel 2 3 var ms = 0 4 var s = 60 5 6 func _process(delta): 7 8 if ms < 1: 9 s -= 1 10 ms = 9 11 12 set_text(str(s)+":"+str(ms)) 13 14 func _ready(): 15 pass FAIL
	Result	

```
extends Control

export (int) var seconds = 0
var dsec = 0

func _ready():
    pass

func _physics_process(delta):

    if seconds > 0 and dsec <= 0:
        seconds -= 1
        dsec = 10

    if seconds >= 10:
        $Seconds.set_text(str(seconds))
    else:
        $Seconds.set_text("0"+str(seconds))

    if dsec >= 10:
        $DSeconds.set_text(str(dsec))
    else:
        $DSeconds.set_text("0"+str(dsec))


func _on_Timer_timeout():
    dsec -= 1
```

PASS

Fix?	In this new section of code, I made the seconds have an export value so I can customise the seconds in the inspector instead of changing the code. Then, I separated the seconds and milliseconds into 2 separate objects so I can reference them using singletons to act as the visual timer. Now, using the timeout function, I can continuously loop the value being subtracted by a millisecond, and this timer will only start when I allow it to through the main game scene script. For this, I will need to implement code into my 3,2,1 Countdown to start the game.

<table>
<tr><td></td><td></td><td> 
</td></tr>
<tr><td rowspan="4">4</td><td>Whenever I tried to play the game, it didn’t do the countdown menu. It was like the whole script worked, but didn’t visually show what it was doing,</td><td> 
</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td>Fix?</td><td>I was really perplexed by this because I had just reorganised all of my scripts and removed all the unnecessary items that I previously used, but not anymore. I detached and reattached the scripts to move the scripts and scenes to the correct location, but this was my critical flaw. If the script is removed, the exported values in the inspector section do not save. This means that all I needed to do was to make the constant value 3 again and it all worked fine and handy.

 
</td></tr>
</table>

<table>
<tr><td rowspan="4">5</td><td>Once I had a low value added to my score, the score would only add the lowest value each and every time no matter how fast I clicked on it.</td><td>

```
if Input.is_action_just_pressed("Leftclick"):
    if aimcast.is_colliding():

        var hit_target = aimcast.get_collider()

        if hit_target.is_in_group("Cubes") and hit_target.is_visible():
            print ("HIT VISIBLE CUBE")
            hit_target.hide()
            var time_left = float(($PlayingTimer/Seconds.text) + "." + ($PlayingTimer/DSeconds.text))
            if vartimer - time_left <= 0.1:
                score += 100
                $Score/ScoreCount.set_text(str(score))
                vartimer = time_left
            elif vartimer - time_left <= 0.25:
                score += 50
                $Score/ScoreCount.set_text(str(score))
                vartimer = time_left
            elif vartimer - time_left <= 0.5:
                score += 25
                $Score/ScoreCount.set_text(str(score))
                vartimer = time_left
            else:
                score += 10
                $Score/ScoreCount.set_text(str(score))
```

</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2"> 

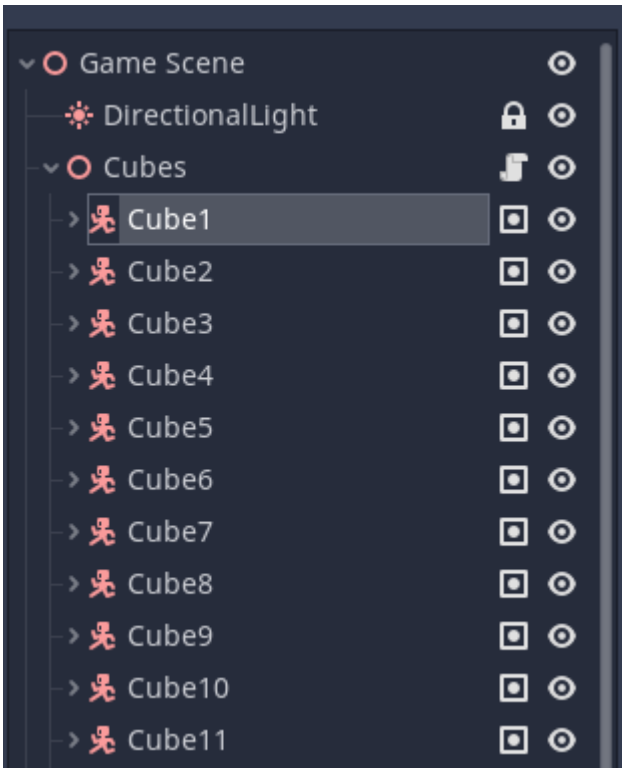
PASS</td></tr>
<tr><td>Fix?</td><td>After doing some debugging, I found out that the vartimer was not changing at all when it was hitting the lowest value. This results in me finding out that there was no new assignment to change the value of “vartimer”, so my vartimer would always be the same after that.

To fix this I added vartimer = time_left in the last line of each if statement, so the vartimer would change to the new time left.</td></tr>
</table>

<table>
<tr><td></td><td></td><td>

```
if hit_target.is_in_group("Cubes") and hit_target.is_visible():
    print ("HIT VISIBLE CUBE")
    hit_target.hide()
    var time_left = float(($PlayingTimer/Seconds.text) + "." + ($PlayingTimer/DSeconds.text))
    if vartimer - time_left <= 0.01:
        score += 100
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    elif vartimer - time_left <= 0.025:
        score += 50
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    elif vartimer - time_left <= 0.05:
        score += 25
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
    else:
        score += 10
        $Score/ScoreCount.set_text(str(score))
        vartimer = time_left
```

</td></tr>
<tr><td rowspan="4">6</td><td rowspan="2">A failure message was outputted in the terminal where it showed that my references to the Cubes weren't being found.</td><td>



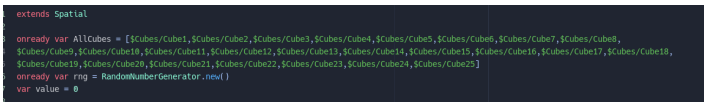
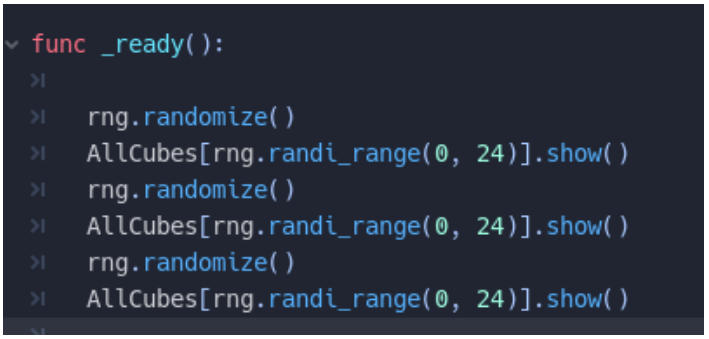
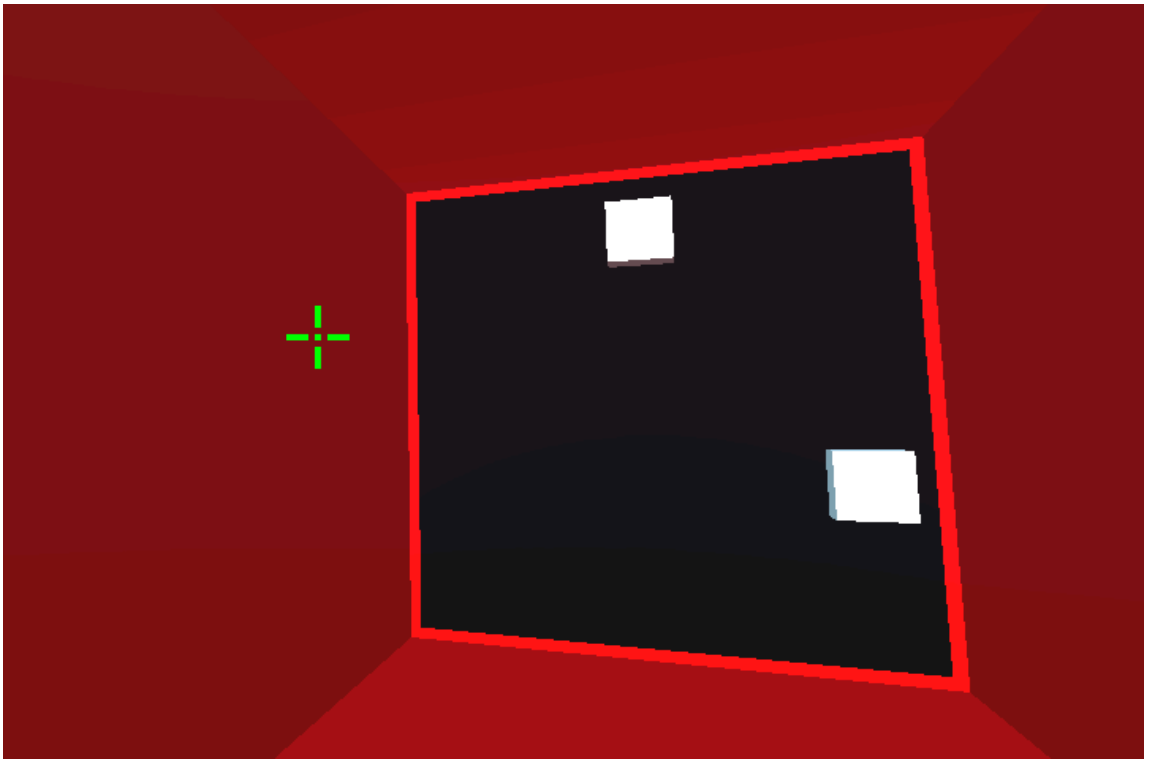

```
2
3  onready var Cube1 = $"Cube10/Cube 10"
4
5
```

</td></tr>
<tr><td>Result</td></tr>
<tr><td colspan="2">

```
0:00:00.449   The argument 'delta' is never used in the function '_process'. If this is intended, prefi
0:00:00.457   get_node: (Node not found: "Cube10/Cube 10" (relative to "/root/Cubes").)
```

PASS

</td></tr>
<tr><td>Fix?</td><td>This means that my program was unable to detect the Cubes, so this way was unnecessary in the first</td></tr>
</table>

<table>
<tr><td></td><td></td><td>place. As such, I realised that I did not need to reference it in the Player Script, I could just do this algorithm in the main Game Scene Script separately. Thus, I unloaded the autoload from my project settings.
  </td></tr>
<tr><td rowspan="3">7</td><td>Sometimes, only 2 cubes would be visible when there should always be 3.</td><td>  </td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2"> 
PASS</td></tr>
</table>

	Fix?	All I needed to do was to put some while loops to compare the new values to the old values, to see if they were the same. If there was, it would continue to loop until the value is different, where it would end the while loop and then show that value of the cube. Now, it has no way of ever showing less or more cubes than 3.
		
		`var value1 = 0`
		`var value2 = 0`
		`var value3 = 0`
		
		`rng.randomize()`
		`value1 = AllCubes[rng.randi_range(0, 24)]`
		`value1.show()`
		
		`rng.randomize()`
		`value2 = AllCubes[rng.randi_range(0, 24)]`
		`while value2 == value1:`
		`  rng.randomize()`
		`  value2 = AllCubes[rng.randi_range(0, 24)]`
		`value2.show()`
		
		`rng.randomize()`
		`value3 = AllCubes[rng.randi_range(0, 24)]`
		`while value3 == value2 or value3 == value1:`
		`  rng.randomize()`
		`  value3 = AllCubes[rng.randi_range(0, 24)]`
		`value3.show()`

Review

After completing this one game mode, I feel like the best course of action would be to adapt from project into a more agile methodology, where I would rather finish my entire project first before working on more game modes or the sign up feature that I planned to add. This will allow me to have a completed game that runs well, instead of one that is half finished.

In this main phase, I have learnt a lot on the coding side of things, where I finally realise the complexity of the code behind any simple game like this project, so this gives me more knowledge of how much time I will spend on each section of the game. Furthermore, having access to use command lists and the step-by-step debugger allows me to solve problems, by showing me visually what is wrong with the code instead of displaying it in text-form.

Final Code

Player.gd:

```gdscript
extends KinematicBody


onready var CAMERA = $Head/Camera
onready var aimcast = $Head/Camera/AimCast
export (float) var vartimer = 59.1
export var sensitivity = 0.05
export var max_angle = 90
export var min_angle = -75
export var rateofs = 15
export var score = 0
var look = Vector3.ZERO
var velocity = Vector3.ZERO
var collision = Vector3.ZERO


func _ready():
    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED) # Captures where the mouse is on the screen


func _physics_process(delta): #delta is unused because I do not require gravity.

    _update_hud()

    _process_input(delta)
```

```gdscript
    CAMERA.rotation_degrees.x = look.x # rotation for x direction
    CAMERA.rotation_degrees.y = look.y # rotation for y direction

    var input_dir := Input.get_vector("moveleft", "moveright", "forward", "backward")

    var direction = (CAMERA.transform.basis.rotated(Vector3.UP, rotation.y) * Vector3(input_dir.x, 0, input_dir.y)).normalized()

    if direction:
            velocity.x = direction.x * rateofs
            velocity.z = direction.z * rateofs
    else:
            velocity.x = move_toward(velocity.x, 0, rateofs)
            velocity.z = move_toward(velocity.z, 0, rateofs)

    velocity = move_and_slide(velocity, Vector3.UP)

func _process_input(_delta):

    if Input.is_action_just_pressed("esc"):
        get_tree().quit()

    if Input.is_action_just_pressed("Leftclick"):
        if aimcast.is_colliding() and $"321Countdown/Timer1".is_stopped():
            var hit_target = aimcast.get_collider()
            if hit_target.is_in_group("Cubes") and hit_target.is_visible():
                print ("HIT VISIBLE CUBE")
                get_parent().buffer = true
```

```
                hit_target.hide()
                var time_left = float(($PlayingTimer/Seconds.text) + "." + ($PlayingTimer/DSeconds.text))
                if vartimer - time_left <= 0.01:
                    score += 100
                    $Score/ScoreCount.set_text(str(score))
                    vartimer = time_left
                elif vartimer - time_left <= 0.025:
                    score += 75
                    $Score/ScoreCount.set_text(str(score))
                    vartimer = time_left
                elif vartimer - time_left <= 0.05:
                    score += 50
                    $Score/ScoreCount.set_text(str(score))
                    vartimer = time_left
                else:
                    score += 10
                    $Score/ScoreCount.set_text(str(score))
                    vartimer = time_left
            elif hit_target.is_in_group("Cubes"):
                print("HIT HIDDEN CUBE")
            else:
                pass
        else:
            pass
```

```
func _input(event):
    if event is InputEventMouseMotion:
        look.y -= (event.relative.x * sensitivity) #Having the y vector - the x relative to the screen for each input
        look.x -= (event.relative.y * sensitivity) #Having the x vector - the y relative to the screen for each input
        look.x = clamp(look.x, min_angle, max_angle) #Setting a cap to how far the player can look in either y direction (Up/Down)

func _update_hud():
    $HUD/Crosshair.material.set_shader_param("color_id", 0)
```

321Countdown.gd:

```
extends Control

onready var n3 = $number3
onready var n2 = $number2
onready var n1 = $number1
export var constant = 0

func _ready():
    n3.hide()
    n2.hide()
    n1.hide()

func _process(_delta):

    if constant == 3:
        n3.show()
    elif constant == 2:
        n3.hide()
        n2.show()
    elif constant == 1:
        n2.hide()
        n1.show()
    elif constant <= 0:
        n1.hide()
        $"red transparency".hide()
        $Timer1.stop()
```

```
28 func _on_Timer_timeout():
29     constant -= 1
```

PlayingCountdown.gd:

```
extends Control

export (int) var seconds = 0
var dsec = 0
var counter = 18.0
var n = 1.0

func _ready():
    $Timer2.wait_time = 3.0
    $Timer2.start()

func _physics_process(_delta):
    if counter - n * $Timer2.wait_time >= 0.0:
        $Seconds.set_text(str(60))
        $DSeconds.set_text("0"+str(0))
        n += 1.0
    else:
        $Timer2.wait_time = 0.1
        calc()

func calc():
    if seconds > 0 and dsec <= 0:
        seconds -= 1
        dsec = 10

    if seconds >= 10:
        $Seconds.set_text(str(seconds))
```

```
    else:
        $Seconds.set_text("0"+str(seconds))

    if dsec >= 10:
        $DSeconds.set_text(str(dsec))
    else:
        $DSeconds.set_text("0"+str(dsec))

    if seconds == 0 and dsec == 0:
        $Timer2.stop()
        get_tree().quit() #this should change to the result page later on

func _on_Timer2_timeout():
    dsec -= 1
```

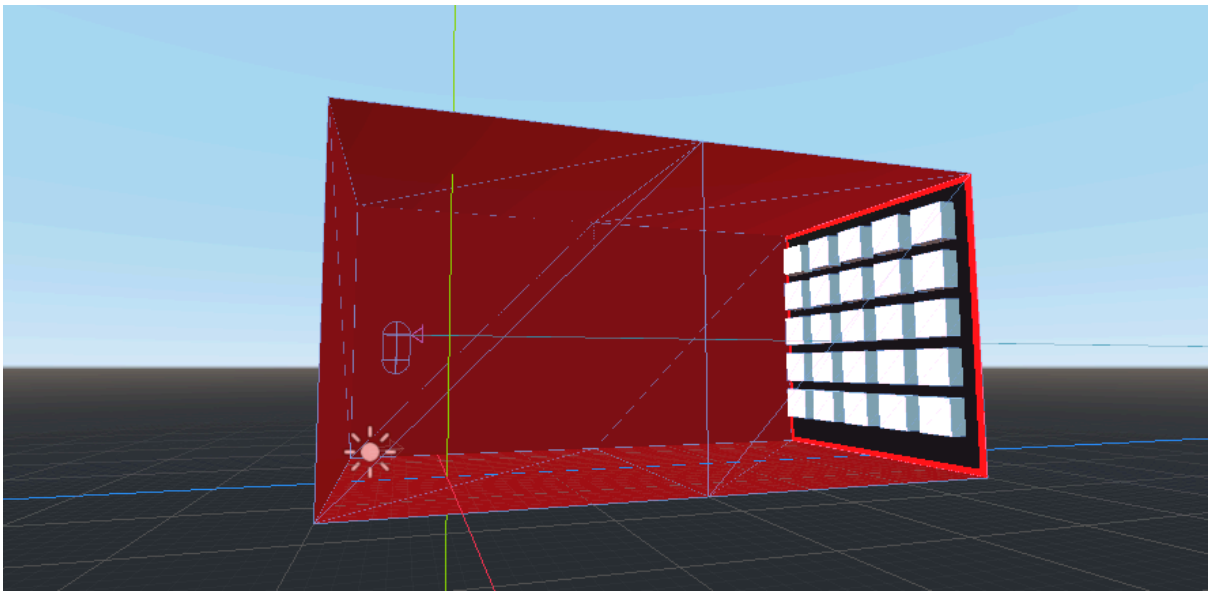
Gridshot.gd:

```
extends Spatial

onready var AllCubes = [$Cubes/Cube1,$Cubes/Cube2,$Cubes/Cube3,$Cubes/Cube4,$Cubes/Cube5,$Cubes/Cube6,$Cubes/Cube7,$Cubes/Cube8,
$Cubes/Cube9,$Cubes/Cube10,$Cubes/Cube11,$Cubes/Cube12,$Cubes/Cube13,$Cubes/Cube14,$Cubes/Cube15,$Cubes/Cube16,$Cubes/Cube17,$Cubes/Cube18,
$Cubes/Cube19,$Cubes/Cube20,$Cubes/Cube21,$Cubes/Cube22,$Cubes/Cube23,$Cubes/Cube24,$Cubes/Cube25]
onready var rng = RandomNumberGenerator.new()
var value1 = 0
var value2 = 0
var value3 = 0
var repeatvalue = 0
var buffer = false

func _process(_delta):
    if $"Player/321Countdown/Timer1".is_stopped() and buffer == true:
        rng.randomize()
        repeatvalue = rng.randi_range(0, 24)
        if AllCubes[repeatvalue].is_visible():
            pass
        else:
            AllCubes[repeatvalue].show()
            buffer = false

func _ready():
    
    $Cubes/Cube1.hide()
    $Cubes/Cube2.hide()
    $Cubes/Cube3.hide()
```

```
    $Cubes/Cube4.hide()
    $Cubes/Cube5.hide()
    $Cubes/Cube6.hide()
    $Cubes/Cube7.hide()
    $Cubes/Cube8.hide()
    $Cubes/Cube9.hide()
    $Cubes/Cube10.hide()
    $Cubes/Cube11.hide()
    $Cubes/Cube12.hide()
    $Cubes/Cube13.hide()
    $Cubes/Cube14.hide()
    $Cubes/Cube15.hide()
    $Cubes/Cube16.hide()
    $Cubes/Cube17.hide()
    $Cubes/Cube18.hide()
    $Cubes/Cube19.hide()
    $Cubes/Cube20.hide()
    $Cubes/Cube21.hide()
    $Cubes/Cube22.hide()
    $Cubes/Cube23.hide()
    $Cubes/Cube24.hide()
    $Cubes/Cube25.hide()
```

```
50
51      rng.randomize()
52      value1 = AllCubes[rng.randi_range(0, 24)]
53      value1.show()
54
55      rng.randomize()
56      value2 = AllCubes[rng.randi_range(0, 24)]
57      while value2 == value1:
58          rng.randomize()
59          value2 = AllCubes[rng.randi_range(0, 24)]
60      value2.show()
61
62      rng.randomize()
63      value3 = AllCubes[rng.randi_range(0, 24)]
64      while value3 == value2 or value3 == value1:
65          rng.randomize()
66          value3 = AllCubes[rng.randi_range(0, 24)]
67      value3.show()
68
```

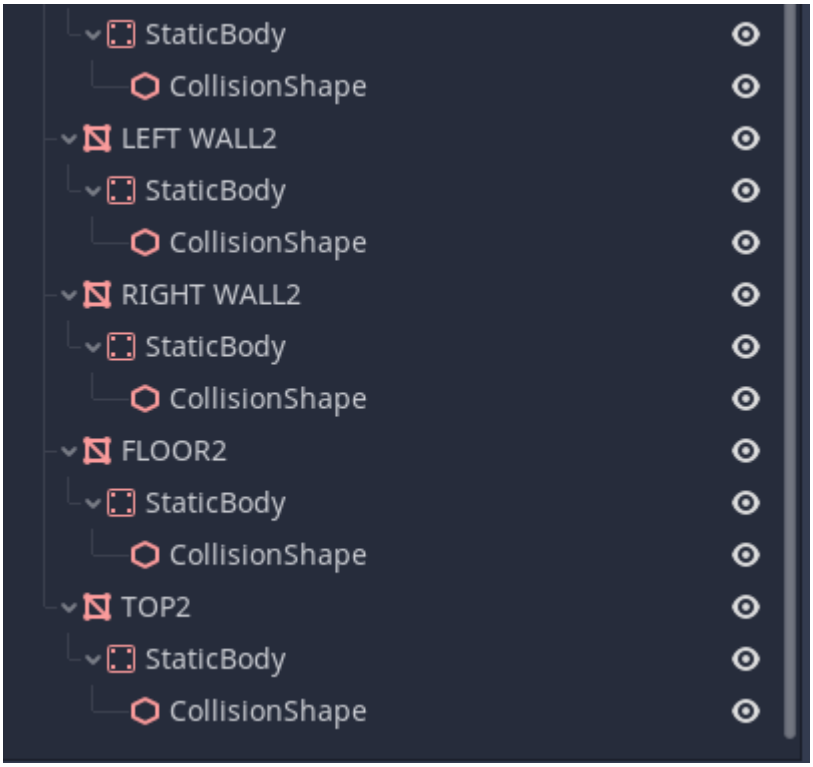
3D Game Scene:



Parents and Child Nodes (of the scene):


Game Scene
DirectionalLight
Cubes
Cube1
Cube2
Cube3
Cube4
Cube5
Cube6
Cube7
Cube8
Cube9
Cube10
Cube11
Cube12
Cube13
Cube14
Cube15
Cube16
Cube17
Cube18
Cube19
Cube20
Cube21
Cube22
Cube23
Cube24
Cube25
Player
321Countdown
red transparency
number1
number2
number3

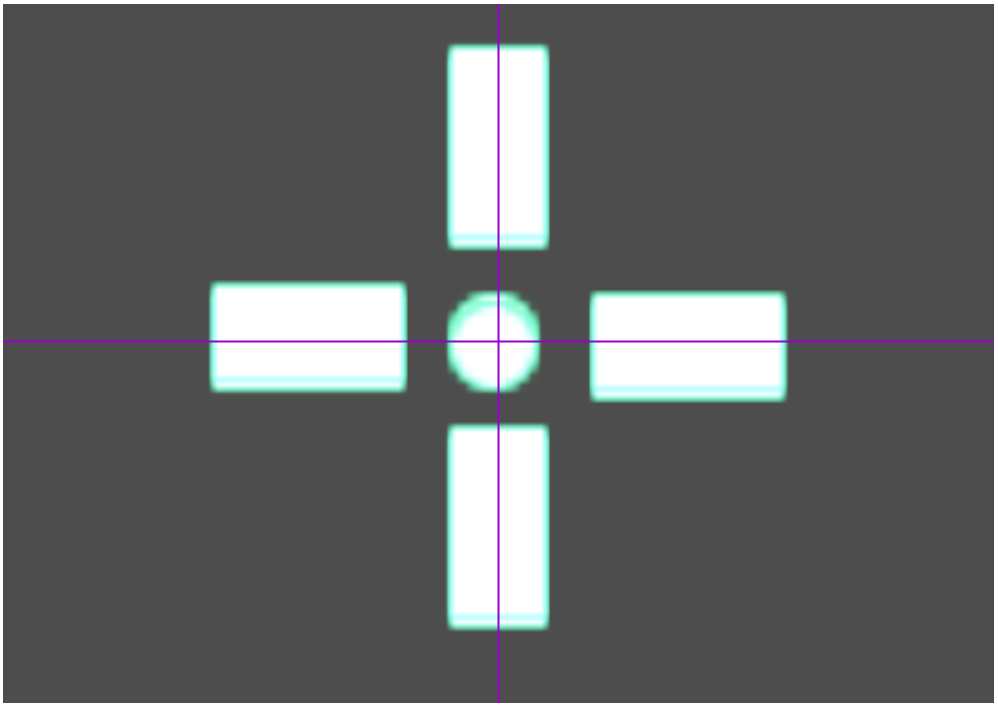

Timer1
PlayingTimer
Seconds
SemiColon
DSeconds
Timer2
HUD
Crosshair
CollisionShape
Head
Camera
AimCast
Score
ScoreCount
Walls
FRONT WALL
SCREEN
FLOOR
StaticBody
CollisionShape
LEFT WALL
StaticBody
CollisionShape
RIGHT WALL
StaticBody
CollisionShape
BACK WALL
StaticBody
CollisionShape
TOP


StaticBody
CollisionShape
LEFT WALL2
StaticBody
CollisionShape
RIGHT WALL2
StaticBody
CollisionShape
FLOOR2
StaticBody
CollisionShape
TOP2
StaticBody
CollisionShape

Phase 3 - Settings

This settings section allows the user to choose which crosshair in particular they would prefer to have. As being able for the user to customise their crosshair would take a long time, I have removed the current HUD script that I had imported, and decided to keep it simple by giving the user a selection type of 3 different crosshairs, with 3 different colours for each.

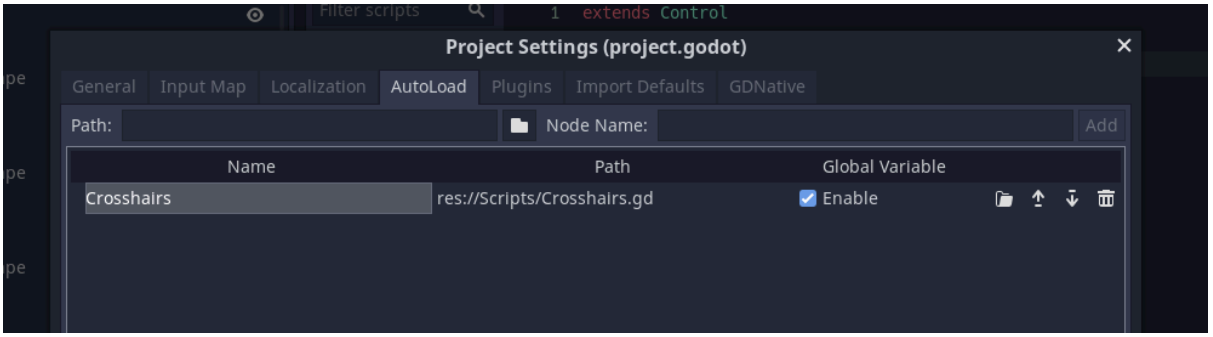
Step 1: Creating the basic prototype of each of the customizable section



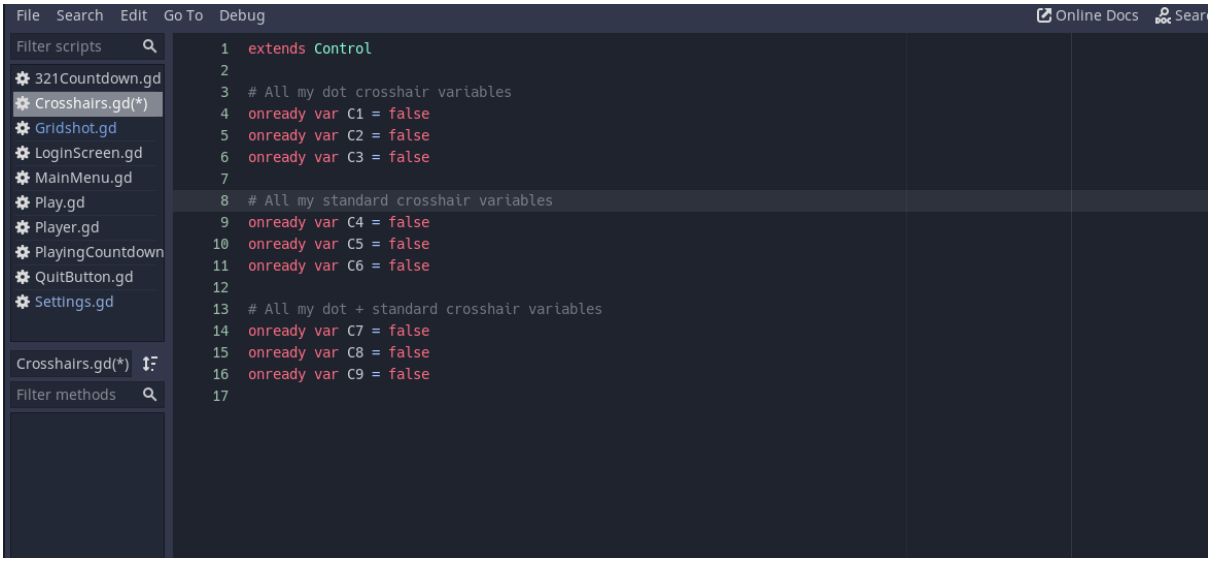
I have made the 9 different crosshairs in photoshop and imported them into the project with all of them having the HUD node as the parent. Additionally, I think moving the HUD outside

the Player node would help because I plan to set it to be global as I will be using it in my Settings scene.

Step 2: Global Variable referencing

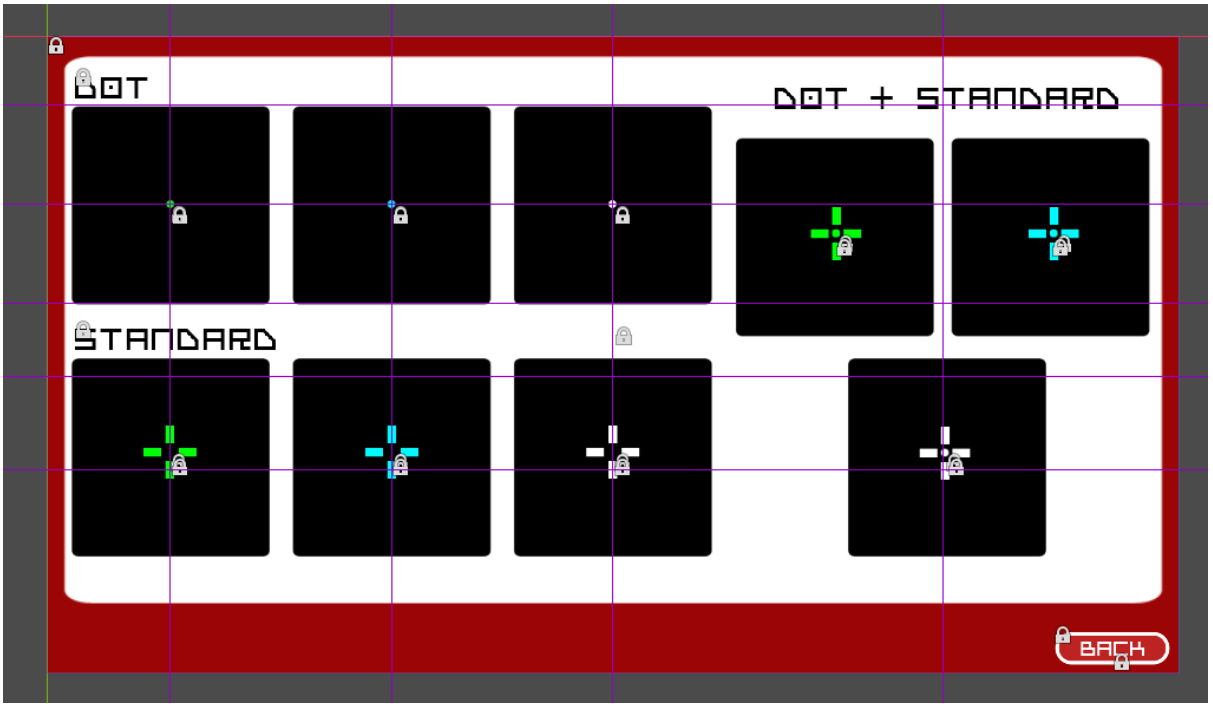


Now, in this case, I have added it correctly because now I can use it in different scenes of my choice by putting Crosshairs.(action), as “Crosshairs” is the node and in this script, I will label each of the crosshairs to a separate variable.

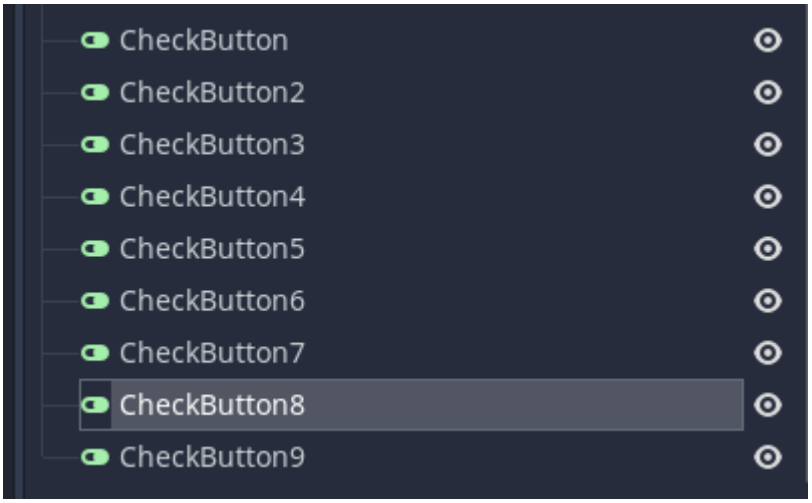


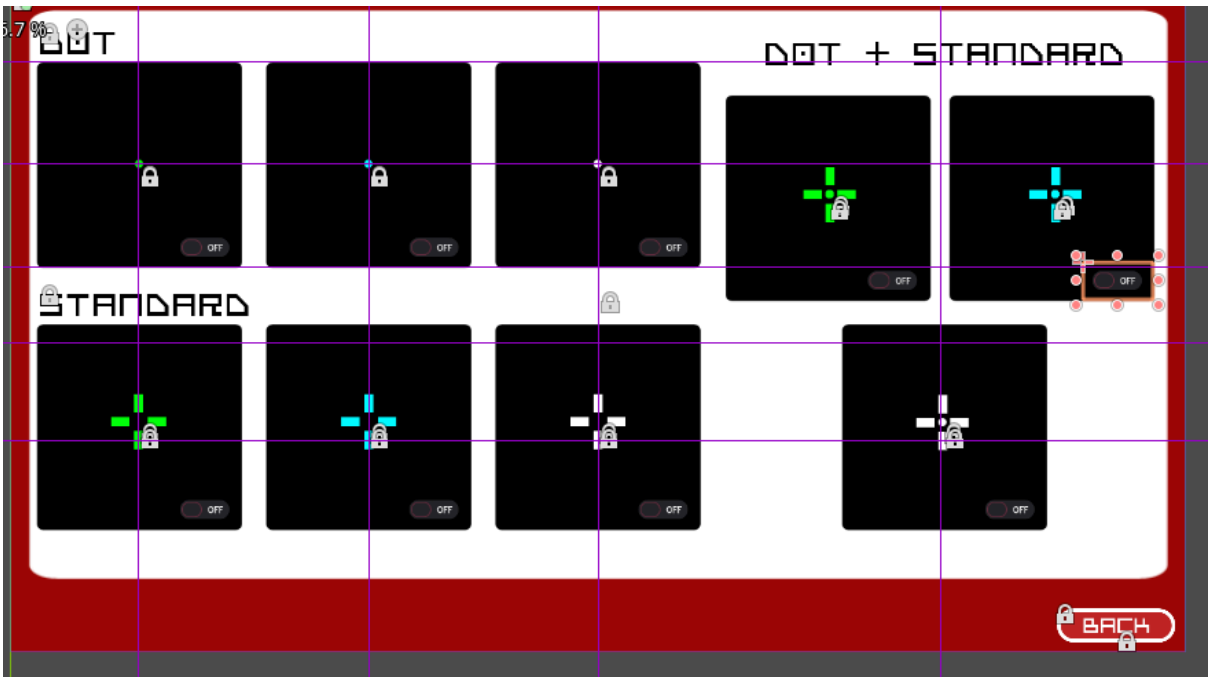
By learning from my previous mistake, I cannot reference the image that I want to directly change as it will not recognise the path, but these global variables can change to what the programmer wants it to do by settings being able to fit a condition. In this case, it will be false to hide() and true to show().

Step 3: The Settings Scene

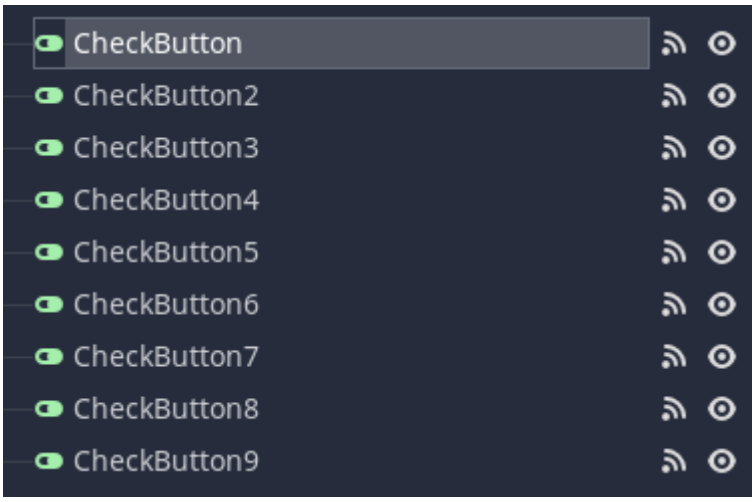


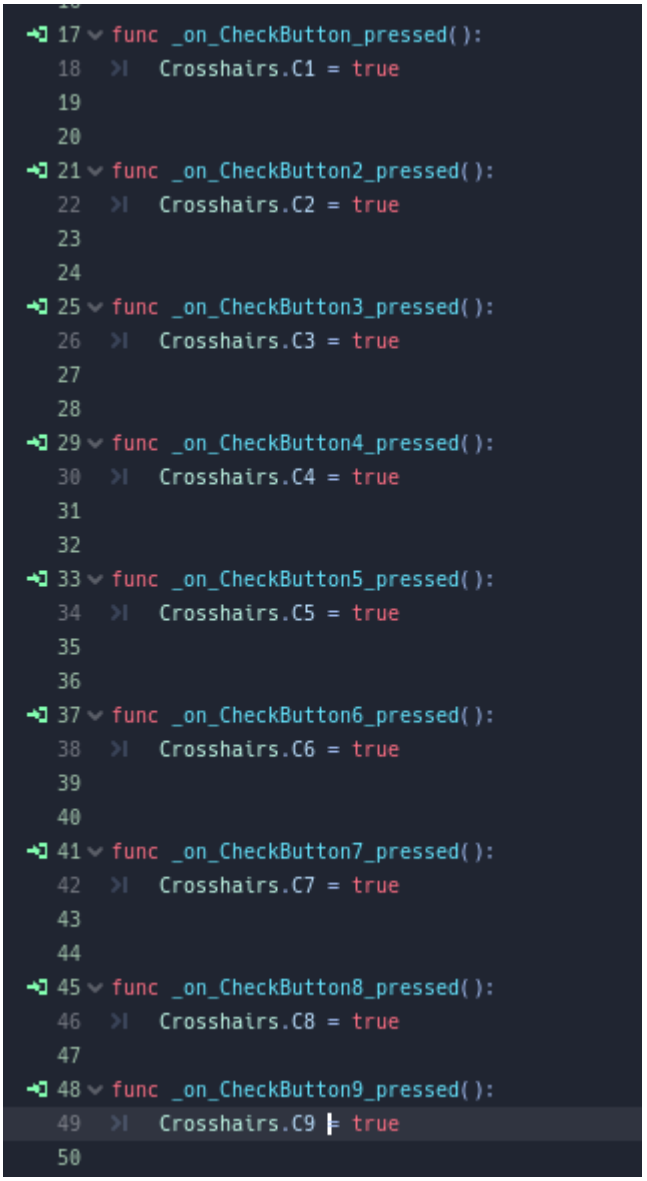
I have made a quick rendition of the Settings design that I previously constructed and all I need to do now is the buttons that go along with it. For this, I will be using a feature implemented into godot called the check button.





Place them in the bottom right corner of each crosshair preview.





```
17 func _on_CheckButton_pressed():
18     Crosshairs.C1 = true
19
20
21 func _on_CheckButton2_pressed():
22     Crosshairs.C2 = true
23
24
25 func _on_CheckButton3_pressed():
26     Crosshairs.C3 = true
27
28
29 func _on_CheckButton4_pressed():
30     Crosshairs.C4 = true
31
32
33 func _on_CheckButton5_pressed():
34     Crosshairs.C5 = true
35
36
37 func _on_CheckButton6_pressed():
38     Crosshairs.C6 = true
39
40
41 func _on_CheckButton7_pressed():
42     Crosshairs.C7 = true
43
44
45 func _on_CheckButton8_pressed():
46     Crosshairs.C8 = true
47
48 func _on_CheckButton9_pressed():
49     Crosshairs.C9 = true
50
```

I have linked all of the buttons up so when the user presses the on switch button. The global variable that I have made would be set to true. However, this doesn't take into account whether the checkbutton is toggled on or toggled off, so the game cannot distinguish between them. This means that I should have done the _toggled instead of _pressed.

```
16 func _on_CheckButton_toggled(button_pressed):
17     if $CheckButton.is_toggle_mode() == true:
18         Crosshairs.C1 = true
19     elif $CheckButton.is_toggle_mode() == false:
20         Crosshairs.C1 = false
21
22
23
24 func _on_CheckButton2_toggled(button_pressed):
25     if $CheckButton2.is_toggle_mode() == true:
26         Crosshairs.C2 = true
27     elif $CheckButton2.is_toggle_mode() == false:
28         Crosshairs.C2 = false
29
```

Now, because godot automatically deletes all the data of a scene when changing to another scene, I will need to make sure that the data is the same even when I change back to the scene. But, this didn't actually record my actions on the slider in the first place. I misunderstood the toggle_mode() feature, as it really meant turning the feature of toggling off or not. (Test 1)

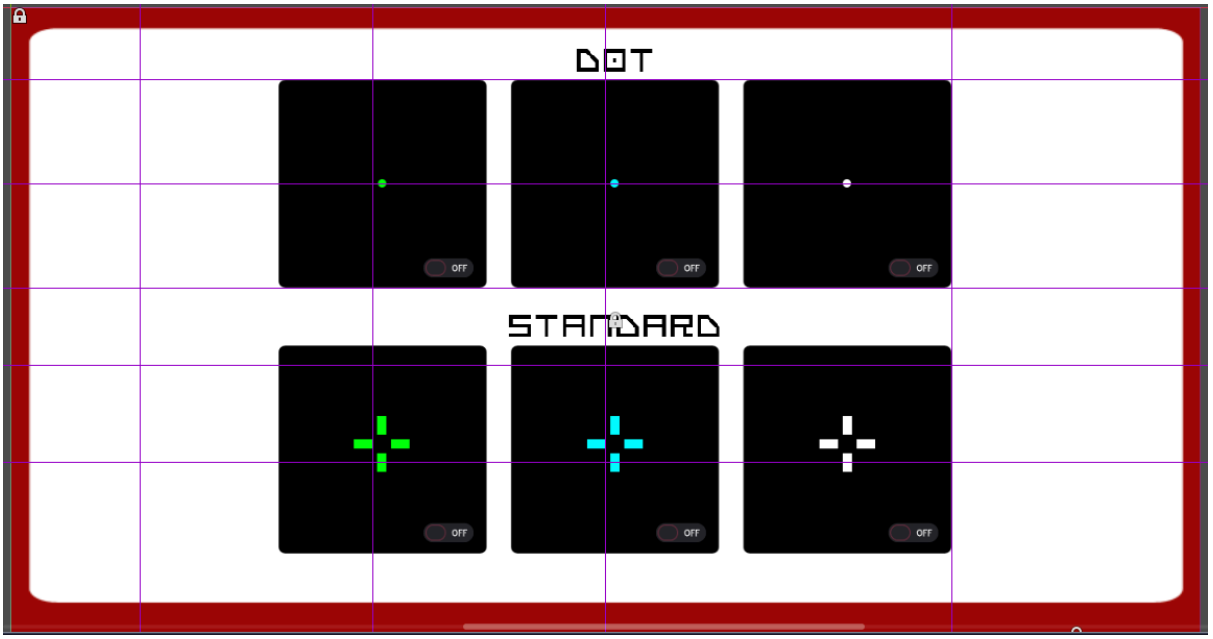
```
func _ready():
    if Crosshairs.C1 == true:
        $CheckButton.pressed = true
    else:
        $CheckButton.pressed = false
```

```
19 func _on_CheckButton_toggled(button_pressed):
20     if button_pressed:
21         Crosshairs.C1 = true
22     else:
23         Crosshairs.C1 = false
24
```

By putting it in the _ready() section, everytime I load up the Settings scene, it will load the data from the global variable I set on/off in my toggled function. This was simply done by the "pressed" command to turn it on for the user at the beginning of every scene change.

```
func _ready():
    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED)
    _update_hud()
```

With this, I put the _update_hud() function into the _ready() one because that scene would only need to call it once to check which crosshairs are on or off.



Changed the layout by removing Checkbutton 7, 8, and 9 as they were now redundant in the UI and the code.

Testing Phase 3

<table>
<tr><th>ID (T)</th><th>Description</th><th>Code/Input</th></tr>
<tr><td rowspan="4">1</td><td>Toggling on and off did not work the program as intended.</td><td>

```
16 func _on_CheckButton_toggled(button_pressed):
17     if $CheckButton.is_toggle_mode() == true:
18         Crosshairs.C1 = true
19     elif $CheckButton.is_toggle_mode() == false:
20         Crosshairs.C1 = false
21
22
23
24 func _on_CheckButton2_toggled(button_pressed):
25     if $CheckButton2.is_toggle_mode() == true:
26         Crosshairs.C2 = true
27     elif $CheckButton2.is_toggle_mode() == false:
28         Crosshairs.C2 = false
29
```

</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2">

```
19 func _on_CheckButton_toggled(button_pressed):
20     if button_pressed:
21         Crosshairs.C1 = true
22     else:
23         Crosshairs.C1 = false
24
```

```
func _ready():
    if Crosshairs.C1 == true:
        $CheckButton.pressed = true
    else:
        $CheckButton.pressed = false
```

PASS

</td></tr>
<tr><td>Fix?</td><td>By putting it in the _ready() section, everytime I load up the Settings scene, it will load the data from the global variable I set on/off in my toggled function. This was simply done by the “pressed” command to turn it on for the user at the beginning of every scene change.</td></tr>
</table>

Reflection and Review

This and my last phase of development will be very short since these are important aspects of the game, but it doesn't take too long to code or document. I have achieved success on this in only a few steps since in reality, I had to research a ton online to get to the conclusion to make the on and off switches actually work.

From my stakeholders, this is what I intended to do from the design phase, as the data is stored to a global variable that checks each time when it's loaded, whether the crosshair is active or not. This makes it easy to confirm to the user which combination of crosshair they would like to have through iterative testing in the game space.

Additionally, I have learnt quite a bit about the parameters that are associated with each function as I had not known that the "button_pressed" parameter that is from the signal is representing the action of left clicking, so this was much easier than I anticipated because I thought I had to create a entire new input system like I did within my second phase.

Some simple quality of life changes or features that I might make in the future would be to add more customisable crosshairs that the user can make. This would be extremely beneficial for the most experienced users that have a particular crosshair that they perform well on. However, this might take too long and considering I still have a phase remaining and documentation to look through, it might not be possible to achieve.

Final Code

Settings.gd:

```
extends Control

func _ready():
    if Crosshairs.C1:
        $CheckButton.pressed = true
    else:
        $CheckButton.pressed = false
    if Crosshairs.C2:
        $CheckButton2.pressed = true
    else:
        $CheckButton2.pressed = false
    if Crosshairs.C3:
        $CheckButton3.pressed = true
    else:
        $CheckButton3.pressed = false
    if Crosshairs.C4:
        $CheckButton4.pressed = true
    else:
        $CheckButton4.pressed = false
    if Crosshairs.C5:
        $CheckButton5.pressed = true
    else:
        $CheckButton5.pressed = false
    if Crosshairs.C6:
        $CheckButton6.pressed = true
    else:
        $CheckButton6.pressed = false
```

```
30 func _on_Backbutton_button_up():
31     get_tree().change_scene("res://Scenes/MainMenu.tscn")
32
33
34 func _on_CheckButton_toggled(button_pressed):
35     if button_pressed:
36         Crosshairs.C1 = true
37     else:
38         Crosshairs.C1 = false
39
40 func _on_CheckButton2_toggled(button_pressed):
41     if button_pressed:
42         Crosshairs.C2 = true
43     else:
44         Crosshairs.C2 = false
45
46
47 func _on_CheckButton3_toggled(button_pressed):
48     if button_pressed:
49         Crosshairs.C3 = true
50     else:
51         Crosshairs.C3 = false
52
53
54 func _on_CheckButton4_toggled(button_pressed):
```

```
62     if button_pressed:
63         Crosshairs.C5 = true
64     else:
65         Crosshairs.C5 = false
66
67
68 func _on_CheckButton6_toggled(button_pressed):
69     if button_pressed:
70         Crosshairs.C6 = true
71     else:
72         Crosshairs.C6 = false
73
74
75
```

Player.gd:

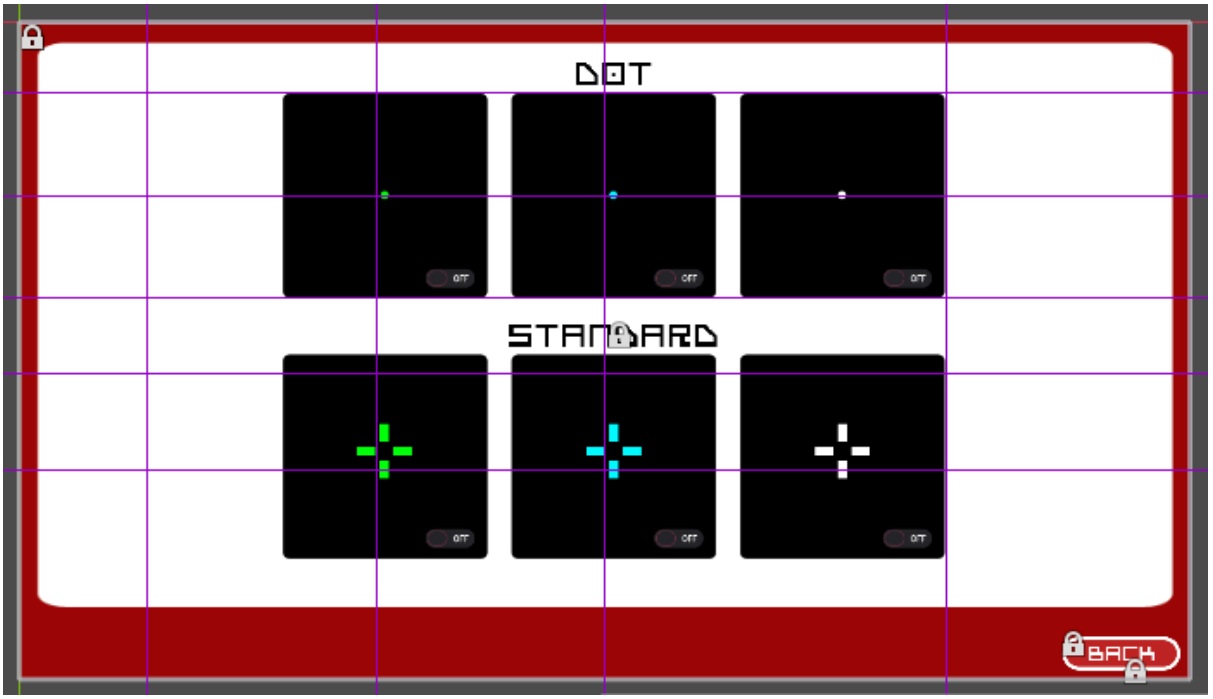
```
func _ready():
    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED) #
    _update_hud()
```

```
82  func _update_hud():
83      if Crosshairs.C1 == true:
84          $HUD/CrosshairDot.show()
85      else:
86          pass
87      if Crosshairs.C2 == true:
88          $HUD/CrosshairDotCyan.show()
89      else:
90          pass
91      if Crosshairs.C3 == true:
92          $HUD/CrosshairDotWhite.show()
93      else:
94          pass
95      if Crosshairs.C4 == true:
96          $HUD/CrosshairStandard.show()
97      else:
98          pass
99      if Crosshairs.C5 == true:
100         $HUD/CrosshairStandardCyan.show()
101     else:
102         pass
103     if Crosshairs.C6 == true:
104         $HUD/CrosshairStandardWhite.show()
105     else:
106         pass
```

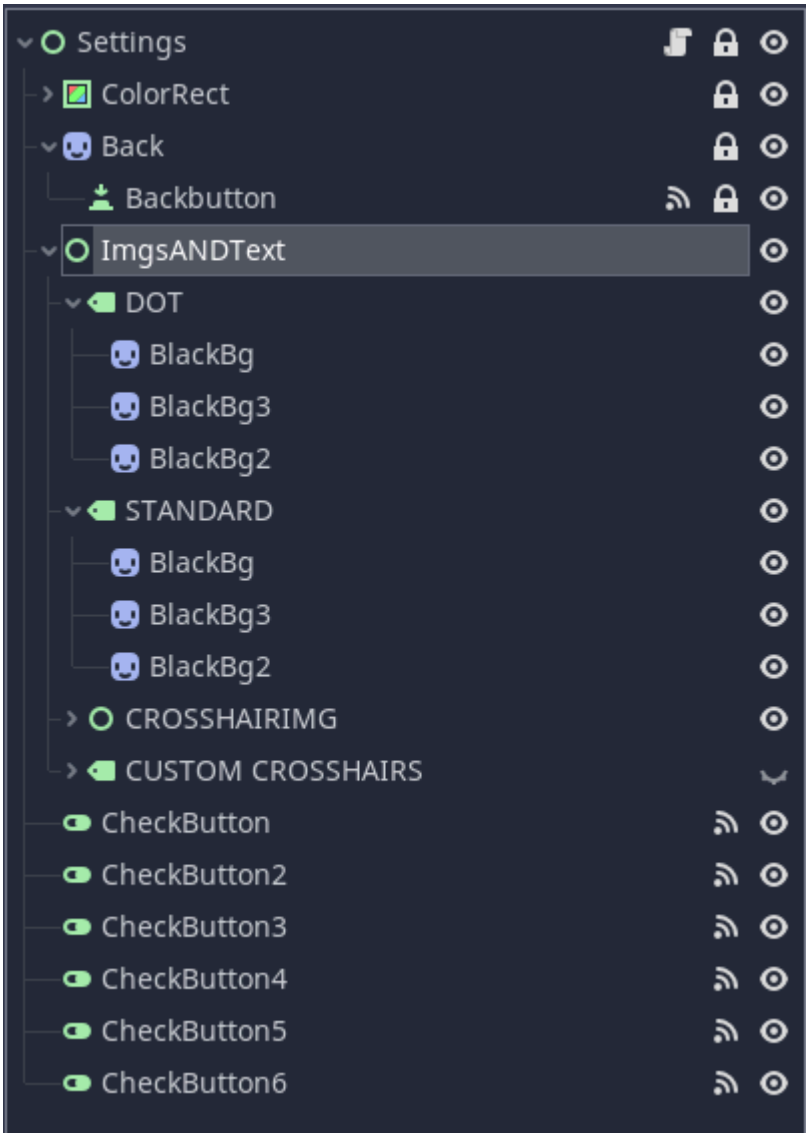
Crosshairs.gd (global variables):

```gdscript
extends Control

# All my dot crosshair variables
onready var C1 = false
onready var C2 = false
onready var C3 = false

# All my standard crosshair variables
onready var C4 = false
onready var C5 = false
onready var C6 = false
```

2D UI:



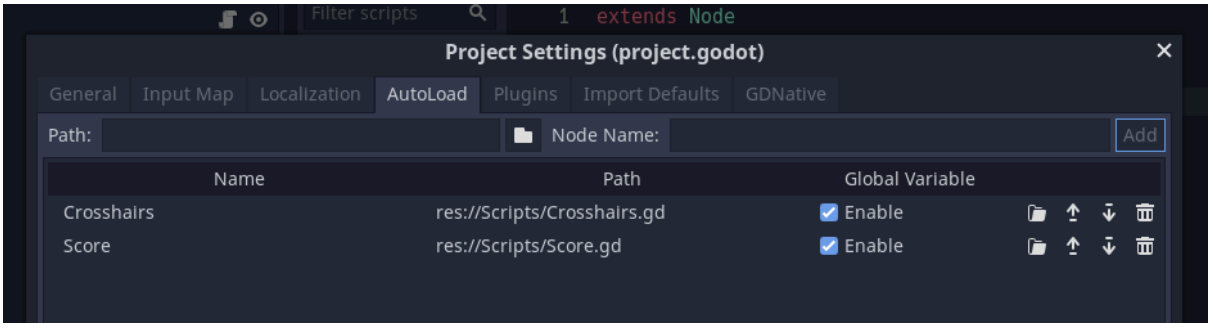
Parents and Child Nodes (of the scene):


Settings
ColorRect
Back
Backbutton
ImgsANDText
DOT
BlackBg
BlackBg3
BlackBg2
STANDARD
BlackBg
BlackBg3
BlackBg2
CROSSHAIRIMG
CUSTOM CROSSHAIRS
CheckButton
CheckButton2
CheckButton3
CheckButton4
CheckButton5
CheckButton6

Phase 4 - Statistical Screen

At the end of each round of gameplay, I will need a way to store the values to later compare them with the leaderboard. This will probably be through a database system, and luckily for me, I have used one previously to store the credentials of the login.

Step 1: Creating the global score script



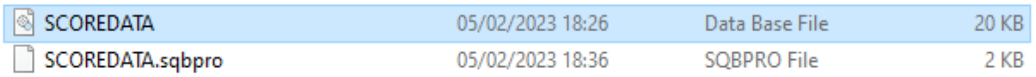

```
1  extends Node
2
3
4  onready var current_score = 0
5  onready var high_score = 0
6  onready var average_score = 0
```

Adding the Score global script to the autoload allows me to use the variables throughout different scenes to give the function the information that it will require.

```
1  extends Control
2
3  func _on_Backbutton_button_up():
4      get_tree().change_scene("res://Scenes/MainMenu.tscn")
5
6  func _ready():
7      $YOURSCORE/COUNTER.set_text(str(Score.current_score))
8      $HIGHESTSCORE/COUNTER.set_text(str(Score.high_score))
9      $AVERAGESCORE/COUNTER.set_text(str(Score.average_score))
10
```

Now, within my Settings script, I can set the text of the current, high and average score, as I will have changed the variable's value to their respective values before changing to this scene.

Step 2: Making the database system and connections



SCOREDATA	05/02/2023 18:26	Data Base File	20 KB
SCOREDATA.sqbpro	05/02/2023 18:36	SQBPRO File	2 KB

```
elif db.query_result.size() >= 1 and db.query_result[0]["Username"] == Username and db.query_res
    Score.ID = db.query_result[0]["U.ID"]
    get_tree().change_scene("res://Scenes/MainMenu.tscn")
    db.close_db()
```

```
7  onready var ID = 0
8
```

In the login screen script, when the user has successfully logged in, the ID of the user will be stored into the global variable. This is due to the fact that each individual score that the user will get, it will be stored to their respective ID database, so high scores can be set correctly and not be muddled within a singular one.

```
extends Control

const SQLite = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns")

var db # database
var db_name = "res://Data Storage/SCOREDATA" # Path to database
var tableName = str(Score.ID)

func _on_Backbutton_button_up():
    get_tree().change_scene("res://Scenes/MainMenu.tscn")

func _ready():

    db = SQLite.new()
    db.path = db_name
    db.open_db()

    $YOURSCORE/COUNTER.set_text(str(Score.current_score))
    $HIGHESTSCORE/COUNTER.set_text(str(Score.high_score))
    $AVERAGESCORE/COUNTER.set_text(str(Score.average_score))
```

Step 3: Storing and retrieving the variables

```
	if seconds == 0 and dsec == 0:
		$Timer2.stop()
		db = SQLite.new()
		db.path = db_name
		db.open_db()
		var current_score = $Score/ScoreCount.text
		var dict : Dictionary = Dictionary()
		dict["SCORE"] = int(current_score)
		
		db.insert_row(tableName,dict)
		
		get_tree().change_scene("res://Scenes/Stats.tscn")
	
```

To store and retrieve variables, I will need to make it so the current_score is stored within the database. I have done this with the Dictionary() feature that the sqlite library has provided, as this will allow me to write a new line with the assistance of the insert_row command. However, when I ran it, it did not save into the database, but since there was an error relating to the “tableName” variable, it must have been because of that. (Test 1)

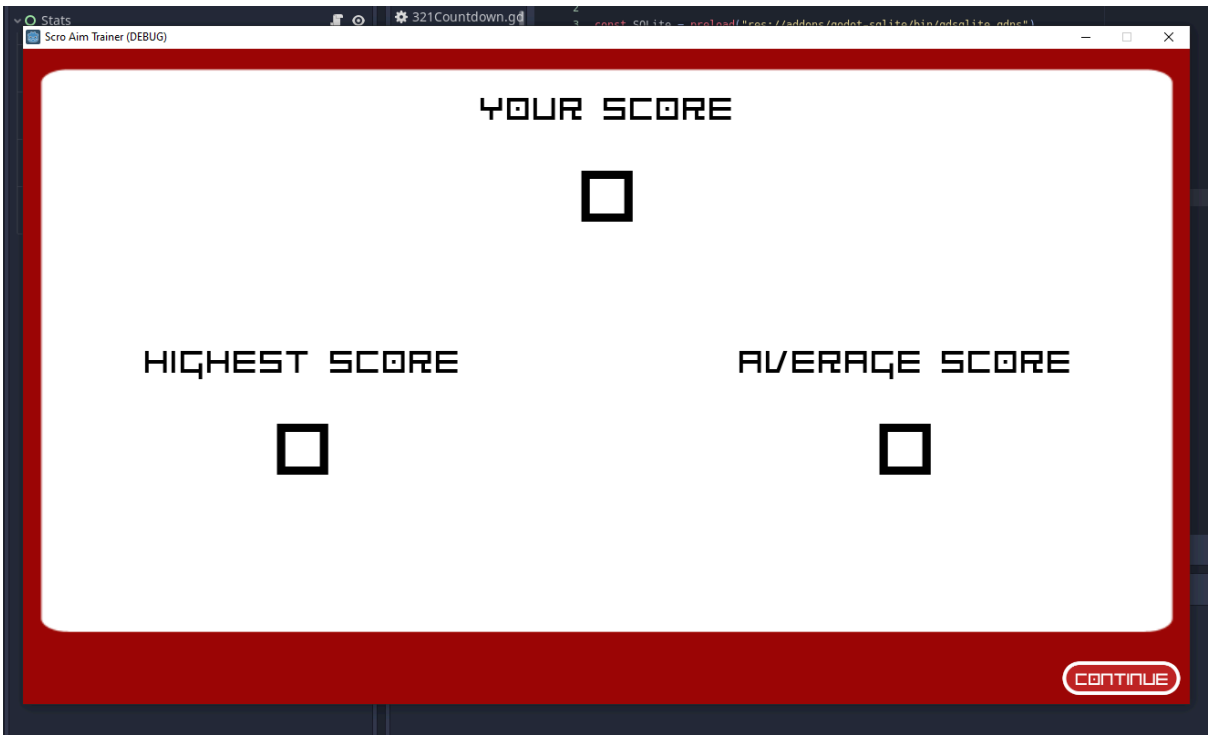
```
var tableName = str(Score.ID)
```

```
34		Score.ID = "UID"+str(db.query_result[0]["U.ID"])
35		get_tree().change_scene("res://Scenes/MainMenu.tscn")
```

I figured out that SQLite did not recognise table names as integer numbers, even though I made the names of the tables in the database those integer numbers. This is easily fixed by adding a “UID” in front of the ID number in the login screen.

```
42	if seconds == 0 and dsec == 0:
43		$Timer2.stop()
44		db = SQLite.new()
45		db.path = db_name
46		db.open_db()
47		var tableName = Score.ID
48		Score.current_score = int($Score/ScoreCount.text)
49		var dict : Dictionary = Dictionary()
50		dict["SCORE"] = Score.current_score
51		
52		db.insert_row(tableName,dict)
53		
54		get_tree().change_scene("res://Scenes/Stats.tscn")
```

Now, it functions as intended, and saves the score to the global variable script and to the database. Nevertheless, I could not use my cursor when I changed it to the statistics screen. (Test 2)




```
func _ready():
    Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)
```

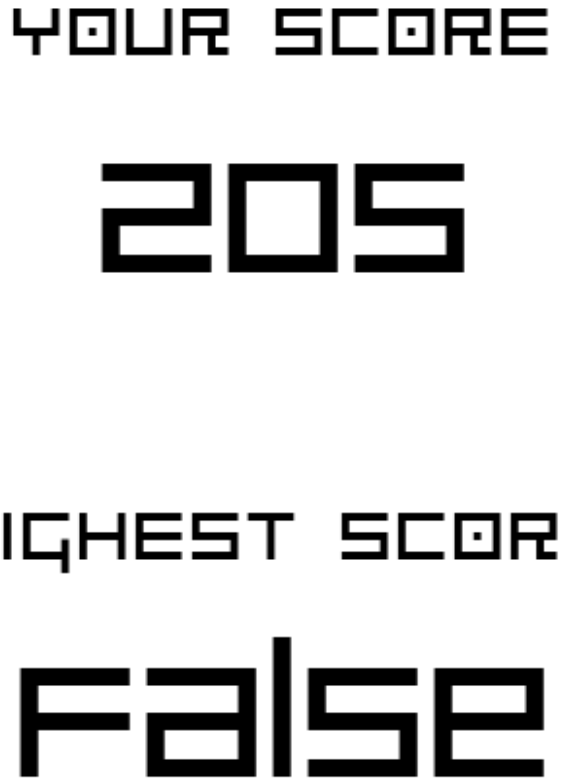
To re-enable the cursor, all I need to do is change the mouse-mode back to the cursor mode when it changes to this scene.

Step 4: Calculating the high score and average score

. For this, I will first calculate the highest score from the database and display it out on the stats screen.

```
52        Score.high_score = db.query("SELECT MAX(SCORE) FROM" + Score.ID + ";")
```

Utilising SQL syntax, the max operator calculates the max value from the user's score database. I thought this would be simple but, remembering from the login screen, I had a similar problem.



Since my global variable has the value of 0, the query is comparing the value of the max(score) to 0, instead of assigning the variable the value. So, false is set to Score.high_score. However, this is easily fixable because all I need to use is the query_result command.

```
52  db.query("SELECT MAX(SCORE) FROM " + tableName + ";") #queries to the sql database to get the maximum score
53  Score.high_score = db.query_result[0]["MAX(SCORE)"] #sets the first value of the "max(score) to the gobal of Score.high_score
```

```
57  db.query("SELECT AVG(SCORE) FROM " + tableName + ";") #queries to the sql database to get the average score
58  Score.average_score = int(db.query_result[0]["AVG(SCORE)"]) #sets the first value of the "avg(score)" to the gobal of Score.average_score
```

Average_score is the same premise but with a different operator in SQL, so it was pretty similar to implement. My game is now complete.

Testing Phase 4

ID (T)	Description	Code/Input

<table>
<tr><td rowspan="5">1</td><td>There was an error relating to the variable of tableName. The current_score was not saved either.</td><td>

```
if seconds == 0 and dsec == 0:
    $Timer2.stop()
    db = SQLite.new()
    db.path = db_name
    db.open_db()
    var current_score = $Score/ScoreCount.text
    var dict : Dictionary = Dictionary()
    dict["SCORE"] = int(current_score)

    db.insert_row(tableName,dict)

    get_tree().change_scene("res://Scenes/Stats.tscn")
```

</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2">

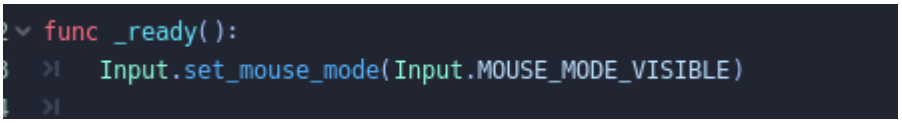
```
var tableName = str(Score.ID)
```

```
34      Score.ID = "UID"+str(db.query_result[0]["U.ID"])
35      get_tree().change_scene("res://Scenes/MainMenu.tscn")
```

PASS

</td></tr>
<tr><td>Fix?</td><td>I figured out that SQLite did not recognise table names as integer numbers, even though I made the names of the tables in the database those integer numbers. This is easily fixed by adding a “UID” in front of the ID number in the login screen.</td></tr>
<tr><td rowspan="4">2</td><td>When I change the scene, the cursor is not active and I cannot click the continue button.</td><td> 

(No cursor active)</td></tr>
<tr><td colspan="2">Result</td></tr>
<tr><td colspan="2"></td></tr>
</table>

	 PASS	
	Fix?	After looking at online forums, I found out that the cursor needs to be re-enabled. When the scene is loaded, the ready() function is run first so all I need to do is put a line in to set the cursor back to visible.

Final Reflection and Review

In this final review, I have made many changes to my design to make it better and intuitively design, but I have tried to keep it similar. Additionally, I wasn't able to create the leaderboard or the other game modes, but these are external features that can be simply added in the future and they don't impact any of the current features in the game. After talking with my stakeholder, this is a good place to stop the development as I have shown my modular design as well, so if I wanted to expand by adding a few more game modes or a difficulty changer for example, then the code I have written would be set as a template, and I can slightly change it to whatever the game mode desires to do. This is why I adopted the agile methodology, as high level code was produced which enables me to do simpler tasks a lot easier.

Linking back to the success criteria and requirements, I have completed all of them except for the ones linking back to my 3 game mode option, such as requirement 1, 3, and 4 and success criteria 4, 6, and 7. This is where me and my stakeholders have changed our opinions on this matter because the game modes I thought of during the analysis section were simple and non-modular algorithms that wouldn't be effective long-term. Now, I only have 1 game mode, but it has modular design, database integrity, referencing using global/local just to name a few, which I am much more proud of doing instead.

Final Project Code (with correct naming conventions and commented code)

LoginScreen.gd:

```
extends Control

# Before the game starts, preload before anything happens
const SQLite = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns")

var db # database
var db_name = "res://Data Storage/Player Credentials" # Path to database
var table_name = "PlayerCredentials" # table name


func _ready():
    if $VBoxContainer/error_label_msg.is_visible():
        # on start of the program, if the error_label is visible, set all to hide
        $VBoxContainer/error_label_msg.hide()
        $VBoxContainer/error_username.hide()
        $VBoxContainer/error_password.hide()
        $VBoxContainer/error_not_found.hide()
    
func _on_Button_button_up():

    # set the text inputted from the user to a variable using singletons, in this case, username and password
    var Username = $VBoxContainer/Usernametype.text
    var Password = $VBoxContainer/Passwordtype.text

    db = SQLite.new() # creating a new instance of the database to use
    db.path = db_name # putting a path towards the file location
    db.open_db() # open db

    # search the database of what the user has inputted into the nodes, when they have clicked the button.
    db.query("SELECT * FROM " + table_name + " WHERE Username = '" + Username + "' OR Password = '" + Password + "';")
```

```
    if db.query_result.size() == 0: # if the database returns empty, no username/password is found.
        if $VBoxContainer/error_password.is_visible():
            $VBoxContainer/error_password.hide() # password hide
        if $VBoxContainer/error_username.is_visible():
            $VBoxContainer/error_username.hide() # username hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_not_found.show() # user has not been found.
    elif db.query_result.size() >= 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] == Password:
        # database has found the userID related to the username and password correctly.
        Score.ID = "UID"+str(db.query_result[0]["U.ID"])
        get_tree().change_scene("res://Scenes/MainMenu.tscn")
        db.close_db()
    elif db.query_result.size() >= 1 and db.query_result[0]["Username"] == Username and db.query_result[0]["Password"] != Password:
        # database has found the username, but the password is incorrect.
        if $VBoxContainer/error_not_found.is_visible():
            $VBoxContainer/error_not_found.hide() # login not found hide
        if $VBoxContainer/error_username.is_visible():
            $VBoxContainer/error_username.hide() # username hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_password.show() # password is incorrect.
    elif db.query_result.size() >= 1 and db.query_result[0]["Username"] != Username and db.query_result[0]["Password"] == Password:
        # database has found the password, but the username is incorrect.
        if $VBoxContainer/error_not_found.is_visible():
            $VBoxContainer/error_not_found.hide() # login not found hide
        if $VBoxContainer/error_password.is_visible():
            $VBoxContainer/error_password.hide() # password hide
        $VBoxContainer/error_label_msg.show()
        $VBoxContainer/error_username.show() # username is incorrect.
    else:
        print("ERROR CODE_1: UNKNOWN") # when there is a hidden error not accounted for.

```

```
func _on_QuitButtonFunc_pressed(): #closes the application when pressed.
    get_tree().quit()
```

Gridshot.gd:

```gdscript
extends Spatial

# Array of all the cube nodes in one variable.
onready var AllCubes = [$Cubes/Cube1,$Cubes/Cube2,$Cubes/Cube3,$Cubes/Cube4,$Cubes/Cube5,$Cubes/Cube6,$Cubes/Cube7,$Cubes/Cube8,
$Cubes/Cube9,$Cubes/Cube10,$Cubes/Cube11,$Cubes/Cube12,$Cubes/Cube13,$Cubes/Cube14,$Cubes/Cube15,$Cubes/Cube16,$Cubes/Cube17,$Cubes/Cube18,
$Cubes/Cube19,$Cubes/Cube20,$Cubes/Cube21,$Cubes/Cube22,$Cubes/Cube23,$Cubes/Cube24,$Cubes/Cube25]
onready var rng = RandomNumberGenerator.new() # create a new instance of RNG andd assign it to the variable rng.
var value_1 = 0 # first cube
var value_2 = 0 # second cube
var value_3 = 0 # third cube
var repeat_value = 0 # if visible, repeat this
var buffer = false # condition of if the cube is hidden or visible.

func _process(_delta):
    # when the countdown timer has stopped, and when the cube hit is visible, show a new cube that was hidden. All down the chance.
    if $"Player/321Countdown/Timer1".is_stopped() and buffer == true:
        rng.randomize()
        repeat_value = rng.randi_range(0, 24) # number of cubes
        if AllCubes[repeat_value].is_visible():
            pass # pass if the cube generated is already visible (repeat function)
        else:
            AllCubes[repeat_value].show() # show the new cube
            buffer = false

func _ready():
    #hide all the cubes
    $Cubes/Cube1.hide()
    $Cubes/Cube2.hide()
    $Cubes/Cube3.hide()
    $Cubes/Cube4.hide()
    $Cubes/Cube5.hide()
    $Cubes/Cube6.hide()
    $Cubes/Cube7.hide()
    $Cubes/Cube8.hide()
    $Cubes/Cube9.hide()
    $Cubes/Cube10.hide()
    $Cubes/Cube11.hide()
    $Cubes/Cube12.hide()
    $Cubes/Cube13.hide()
    $Cubes/Cube14.hide()
    $Cubes/Cube15.hide()
    $Cubes/Cube16.hide()
    $Cubes/Cube17.hide()
    $Cubes/Cube18.hide()
    $Cubes/Cube19.hide()
    $Cubes/Cube20.hide()
    $Cubes/Cube21.hide()
    $Cubes/Cube22.hide()
    $Cubes/Cube23.hide()
    $Cubes/Cube24.hide()
    $Cubes/Cube25.hide()

    rng.randomize() # randomize
    value_1 = AllCubes[rng.randi_range(0, 24)]
    value_1.show() # show a random cube
```

```gdscript
    rng.randomize()
    value_2 = AllCubes[rng.randi_range(0, 24)]
    while value_2 == value_1: #repeat until value_2 != value_1 to show a new cube.
        rng.randomize()
        value_2 = AllCubes[rng.randi_range(0, 24)]
    value_2.show()

    rng.randomize()
    value_3 = AllCubes[rng.randi_range(0, 24)]
    while value_3 == value_2 or value_3 == value_1: #repeat until value_3 != value_1 or value_3 != value_2, to show a new cube.
        rng.randomize()
        value_3 = AllCubes[rng.randi_range(0, 24)]
    value_3.show()
```

Score.gd:

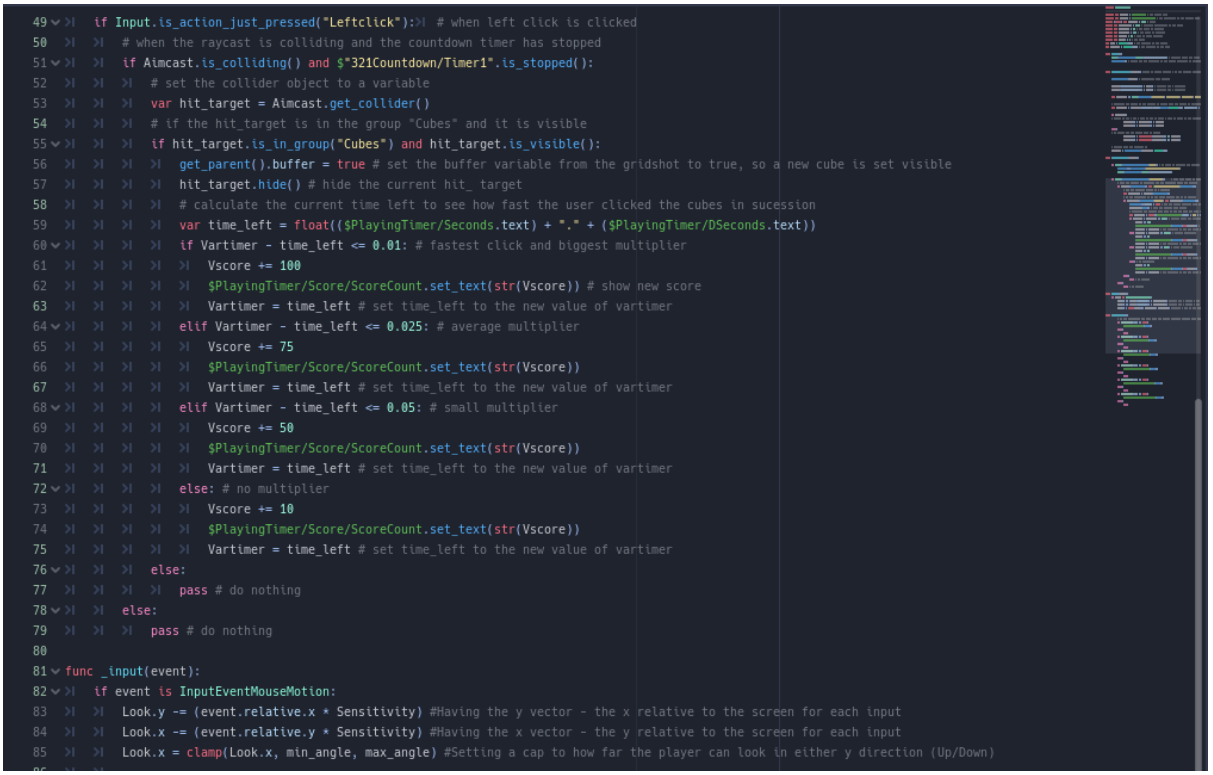
```gdscript
extends Node


onready var current_score = 0 # the score from the game round
onready var high_score = 0 # highest score
onready var average_score = 0 # calculated average score
onready var ID = "" # ID obtained from the login screen
```

Crosshairs.gd:

```gdscript
extends Control

# All my dot crosshair variables, false = hidden
onready var c_1 = false
onready var c_2 = false
onready var c_3 = false

# All my standard crosshair variables, false = hidden
onready var c_4 = false
onready var c_5 = false
onready var c_6 = false
```

Player.gd:

```gdscript
extends KinematicBody

onready var Vcamera = $Head/Camera # the camera node
onready var Aimcast = $Head/Camera/AimCast # the projectory of the raycast from the camera
export (float) var Vartimer = 59.1 # timer
export var Sensitivity = 0.05 # personal sensitivity of the mouse
export var max_angle = 90 # max angle of rotation
export var min_angle = -75 # min angle of rotation
export var Rateofs = 15 # rate of speed increase
export var Vscore = 0 # the score
var Look = Vector3.ZERO # the rotation of the camera
var Velocity = Vector3.ZERO # the movement of the user

func _ready():
    Input.set_mouse_mode(Input.MOUSE_MODE_CAPTURED) # Captures where the mouse is on the screen
    _update_hud() # update the hud depending on the global variables in Crosshairs.gd


func _physics_process(_delta): #delta is unused because I do not require gravity.

    _process_input(_delta) # interacting with objects

    Vcamera.rotation_degrees.x = Look.x # rotation for x direction
    Vcamera.rotation_degrees.y = Look.y # rotation for y direction

    var input_dir := Input.get_vector("moveleft", "moveright", "forward", "backward") # WASD movement

    # transform the camera so the raycast is moving when the camera is rotating
    var Direction = (Vcamera.transform.basis.rotated(Vector3.UP, rotation.y) * Vector3(input_dir.x, 0, input_dir.y)).normalized()

    if Direction:
    # moving on the x and z axis in the 3D space * rate of speed in the postion of the camera
            Velocity.x = Direction.x * Rateofs
            Velocity.z = Direction.z * Rateofs
    else:
    # if camera has not moved, move as normal
            Velocity.x = move_toward(Velocity.x, 0, Rateofs)
            Velocity.z = move_toward(Velocity.z, 0, Rateofs)

    # setting what the velocity is
    Velocity = move_and_slide(Velocity, Vector3.UP)

func _process_input(_delta):

    if Input.is_action_just_pressed("esc"): # if "esc" is pressed, the program is sent back to the main menu.
        get_tree().change_scene("res://Scenes/MainMenu.tscn")
        Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE)
```



```gdscript
if Input.is_action_just_pressed("Leftclick"):
    # when left click is clicked
    # when the raycast is colliding and the countdown timer has stopped
    if Aimcast.is_colliding() and $"321Countdown/Timer1".is_stopped():
        # set the collider object to a variable
        var hit_target = Aimcast.get_collider()
        # if the hit_target is in the group "Cubes" and it is visible
        if hit_target.is_in_group("Cubes") and hit_target.is_visible():
            get_parent().buffer = true # set the buffer variable from the gridshot to true, so a new cube is set visible
            hit_target.hide() # hide the current hit_ target
            # calculate the current time left to see how fast the user clicked the cubes in sucession
            var time_left = float(($PlayingTimer/Seconds.text) + "." + ($PlayingTimer/DSeconds.text))
            if Vartimer - time_left <= 0.01: # fastest click and biggest multiplier
                Vscore += 100
                $PlayingTimer/Score/ScoreCount.set_text(str(Vscore)) # show new score
                Vartimer = time_left # set time_left to the new value of vartimer
            elif Vartimer - time_left <= 0.025: # average multiplier
                Vscore += 75
                $PlayingTimer/Score/ScoreCount.set_text(str(Vscore))
                Vartimer = time_left # set time_left to the new value of vartimer
            elif Vartimer - time_left <= 0.05: # small multiplier
                Vscore += 50
                $PlayingTimer/Score/ScoreCount.set_text(str(Vscore))
                Vartimer = time_left # set time_left to the new value of vartimer
            else: # no multiplier
                Vscore += 10
                $PlayingTimer/Score/ScoreCount.set_text(str(Vscore))
                Vartimer = time_left # set time_left to the new value of vartimer
            else:
                pass # do nothing
        else:
            pass # do nothing

func _input(event):
    if event is InputEventMouseMotion:
        Look.y -= (event.relative.x * Sensitivity) #Having the y vector - the x relative to the screen for each input
        Look.x -= (event.relative.y * Sensitivity) #Having the x vector - the y relative to the screen for each input
        Look.x = clamp(Look.x, min_angle, max_angle) #Setting a cap to how far the player can look in either y direction (Up/Down)
```

```gdscript
func _update_hud():
    # if any crosshairs are true from the global variable script, show them.
    if Crosshairs.c_1 == true:
        $HUD/CrosshairDot.show()
    else:
        pass
    if Crosshairs.c_2 == true:
        $HUD/CrosshairDotCyan.show()
    else:
        pass
    if Crosshairs.c_3 == true:
        $HUD/CrosshairDotWhite.show()
    else:
        pass
    if Crosshairs.c_4 == true:
        $HUD/CrosshairStandard.show()
    else:
        pass
    if Crosshairs.c_5 == true:
        $HUD/CrosshairStandardCyan.show()
    else:
        pass
    if Crosshairs.c_6 == true:
        $HUD/CrosshairStandardWhite.show()
    else:
        pass
```

321Countdown.gd:

```gdscript
extends Control

onready var n_3 = $number3 # "3" image
onready var n_2 = $number2 # "2" image
onready var n_1 = $number1 # "1" image
export var constant = 0

func _ready():
    # hide all the number images
    n_3.hide()
    n_2.hide()
    n_1.hide()

func _process(_delta):

    if constant == 3: # At 3, show the "3" image.
        n_3.show()
    elif constant == 2: # At 2, show the "2" image. hide the previous
        n_3.hide()
        n_2.show()
    elif constant == 1: # At 1, show the "1" image. hide the previous
        n_2.hide()
        n_1.show()
    elif constant <= 0: # At 0, hide the previous and bg, stop timer.
        n_1.hide()
        $"red transparency".hide()
        $Timer1.stop()

func _on_Timer_timeout(): # constant negative per second.
    constant -= 1

```

Mainmenu.gd:

```gdscript
extends Control

func _ready():
    pass # Replace with function body.

func _on_Buttonquit_pressed(): # close program when quit button is pressed
    get_tree().quit()

func _on_PlayButton_button_up(): # change to the game scene
    get_tree().change_scene("res://Scenes/Gridshot.tscn")

func _on_ProfileButton_button_up(): # change to the profile scene
    get_tree().change_scene("res://Scenes/Profile.tscn")

func _on_SettingsButton_button_up(): # change to the settings scene
    get_tree().change_scene("res://Scenes/Settings.tscn")

```

PlayingCountdown.gd:

```gdscript
extends Control

# Before the game starts, preload before anything happens
const SQLite = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns")

var db # database
var db_name = "res://Data Storage/SCOREDATA" # Path to database


export (int) var Seconds = 0 # generally 60 seconds of play time, but can change in inspector
var mil_seconds = 0 # milli seconds variable
var Counter = 18.0 # stall counter
var n = 1.0 # integer multiplier

func _ready():
	$Timer2.wait_time = 3.0 # set the wait time to 3 seconds
	$Timer2.start() # start the timer

func _physics_process(_delta):
	if Counter - n * $Timer2.wait_time >= 0.0: # Loop until value <= 0, now the timer can count down.
		$Seconds.set_text(str(60))
		$DSeconds.set_text("0"+str(0))
		n += 1.0
	else:
		$Timer2.wait_time = 0.1 # set the wait_time back to count every milli second
		calc() # start function calculation below

func calc():
	# if miliseconds go from 1 -> 0, and seconds arent less than 0, minus 1 from seconds
	if Seconds > 0 and mil_seconds <= 0:
		Seconds -= 1
		mil_seconds = 10

	if Seconds >= 10: # text placement
		$Seconds.set_text(str(Seconds))
	else:
		$Seconds.set_text("0"+str(Seconds))

	if mil_seconds >= 10: # text placement
		$DSeconds.set_text(str(mil_seconds))
	else:
		$DSeconds.set_text("0"+str(mil_seconds))
```

```gdscript
	if Seconds == 0 and mil_seconds == 0: # when the timer hits 0
		$Timer2.stop() # stop timer
		db = SQLite.new() # create a new instance of the database
		db.path = db_name # putting a path towards the file location
		db.open_db() # open db
		var table_name = Score.ID # assigning the score.ID to a new table name variable
		Score.current_score = int($Score/ScoreCount.text) # set the game scene score to the score global variable
		var dict : Dictionary = Dictionary() # add a new record to the table
		dict["SCORE"] = Score.current_score # set the parameters

		db.insert_row(table_name,dict) # insert the data to the database

		db.query("SELECT MAX(SCORE) FROM " + table_name + ";") #queries to the sql database to get the maximum score
		Score.high_score = int(db.query_result[0]["MAX(SCORE)"]) #sets the first value of the "max(score)" to the gobal of Score.high_score

		db.query("SELECT AVG(SCORE) FROM " + table_name + ";") #queries to the sql database to get the average score
		Score.average_score = int(db.query_result[0]["AVG(SCORE)"]) #sets the first value of the "avg(score)" to the gobal of Score.average_score

		get_tree().change_scene("res://Scenes/Stats.tscn") # change to the stat scene

func _on_Timer2_timeout(): # minus 1 millisecond every millisecond
	mil_seconds -= 1
```

Profile.gd:

```gdscript
extends Control

# Before the game starts, preload before anything happens
const SQLite = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns")

var db # database
var db_name = "res://Data Storage/SCOREDATA" # Path to database
var table_name = str(Score.ID) # table name

# Called when the node enters the scene tree for the first time.
func _ready():
	db = SQLite.new() # creating a new instance of the database to use
	db.path = db_name # putting a path towards the file location
	db.open_db() # open db

	db.query("SELECT MAX(SCORE) FROM " + table_name + ";") #queries to the sql database to get the maximum score
	Score.high_score = int(db.query_result[0]["MAX(SCORE)"]) #sets the first value of the "max(score)" to the gobal of Score.high_score

	db.query("SELECT AVG(SCORE) FROM " + table_name + ";") #queries to the sql database to get the average score
	Score.average_score = int(db.query_result[0]["AVG(SCORE)"]) #sets the first value of the "avg(score)" to the gobal of Score.average_score

	$HIGHESTSCORE/COUNTER.set_text(str(Score.high_score)) # set the high_score text to the global variable value
	$AVERAGESCORE/COUNTER.set_text(str(Score.average_score)) # set the average_score text to the global variable value

func _on_Backbutton_pressed(): # change scenes back to main menu
	get_tree().change_scene("res://Scenes/MainMenu.tscn")

```

Settings.gd:

```gdscript
extends Control

func _ready():
	# load data from the global script of "Crosshairs" to see if there are already on
	if Crosshairs.c_1:
		$CheckButton.pressed = true
	else:
		$CheckButton.pressed = false
	if Crosshairs.c_2:
		$CheckButton2.pressed = true
	else:
		$CheckButton2.pressed = false
	if Crosshairs.c_3:
		$CheckButton3.pressed = true
	else:
		$CheckButton3.pressed = false
	if Crosshairs.c_4:
		$CheckButton4.pressed = true
	else:
		$CheckButton4.pressed = false
	if Crosshairs.c_5:
		$CheckButton5.pressed = true
	else:
		$CheckButton5.pressed = false
	if Crosshairs.c_6:
		$CheckButton6.pressed = true
	else:
		$CheckButton6.pressed = false

func _on_Backbutton_button_up(): # change the scene back to main menu
	get_tree().change_scene("res://Scenes/MainMenu.tscn")


```

```gdscript
func _on_CheckButton_toggled(button_pressed): # press the button to turn on and again to turn it back off
    if button_pressed:
        Crosshairs.c_1 = true
    else:
        Crosshairs.c_1 = false

func _on_CheckButton2_toggled(button_pressed): # press the button to turn on and again to turn it back off
    if button_pressed:
        Crosshairs.c_2 = true
    else:
        Crosshairs.c_2 = false


func _on_CheckButton3_toggled(button_pressed): # press the button to turn on and again to turn it back off
    if button_pressed:
        Crosshairs.c_3 = true
    else:
        Crosshairs.c_3 = false


func _on_CheckButton4_toggled(button_pressed): # press the button to turn on and again to turn it back off
    if button_pressed:
        Crosshairs.c_4 = true
    else:
        Crosshairs.c_4 = false


func _on_CheckButton5_toggled(button_pressed): # press the button to turn on and again to turn it back off
    if button_pressed:
        Crosshairs.c_5 = true
    else:
        Crosshairs.c_5 = false

```

```gdscript
func _on_CheckButton6_toggled(button_pressed): # press the button to turn on and again to turn it back off
    if button_pressed:
        Crosshairs.c_6 = true
    else:
        Crosshairs.c_6 = false
```

Stats.gd:

```gdscript
extends Control
# Before the game starts, preload before anything happens
const SQLite = preload("res://addons/godot-sqlite/bin/gdsqlite.gdns")

var db # database
var db_name = "res://Data Storage/SCOREDATA" # Path to database
var tableName = str(Score.ID) # table name of the user

func _on_Backbutton_button_up(): # change scene back to the main menu
    get_tree().change_scene("res://Scenes/MainMenu.tscn")

func _ready():
    Input.set_mouse_mode(Input.MOUSE_MODE_VISIBLE) # set the cursor back to visible

    db = SQLite.new() # creating a new instance of the database to use
    db.path = db_name # putting a path towards the file location
    db.open_db() # open db

    $YOURSCORE/COUNTER.set_text(str(Score.current_score)) # set the current_score text to the global variable value
    $HIGHESTSCORE/COUNTER.set_text(str(Score.high_score)) # set the high_score text to the global variable value
    $AVERAGESCORE/COUNTER.set_text(str(Score.average_score)) # set the average_score text to the global variable value

```

Evaluation

Final Test Plan (with stakeholders)

In this final test plan, I aim to record my stakeholders playthrough of my game starting from the login screen. This will help provide more evidence for how the game would be run and its robustness if a real user tested my game, and I will be able to see how usable it is with someone who does not know all the features beforehand.

The videos of my stakeholders POV are shown here.

Test Description	Test Data	Expected Outcome	Link	Evidence
Testing the login screen with incorrect login information.	Incorrect username Incorrect password	Expect error message to show up	Evidence for robustness, testing each situation will provide a different error message that will let the user understand the issue.	AdamPOV.mp4 Timestamp: 0:00 to 0:25
Testing all the buttons work as intended	User left click	change_scene ()	Evidence for robustness, each scene change unloads the old scene and loads the new scene, showing efficient and healthy management of memory.	Shown throughout the each clip: AdamPOV.mp4 CavanPOV.mp4
Test to see if the settings have kept the data the user inputted.	Left click on and off buttons for each of the crosshairs	Saved data and load data to new scene	Evidence for robustness, the global variables are kept in the script and are loaded when the scene is opened. Shows that data is only in memory when called.	AdamPOV.mp4 Timestamp: 0:30 - 0.35 (shows that the crosshair is saved to the game mode scene)

Test to see the average score, high score, and current score is updated from the database.	Get score from left clicking the cubes. Data is written to the database and queried.	Current score updated, Average score updated, high score updated.	Evidence for robustness, the calculations from each query statement show how fast it calculates the result.	AdamPOV.mp4 Timestamp: 1.36 - 1:43
Test to see the profile is updated when the scene is loaded.	Left clicking the profile button in main menu	Average score updated, high score updated.	Evidence for robustness, the data is retrieved from the database and updated. The updated data is accurate and handles any rigorous testing.	CavanPOV.mp4 Timestamp: 3:39 - 3:41

Evaluation of robustness

All these tests are linked to the success criteria and these tests test the robustness of the program by my stakeholders playing multiple rounds, and the algorithm storing all the data correctly with the stat screen scene loading the new updated value, which was different for each of my stakeholder's profiles. This proves that my program has good robustness because each calculation and score counter was seen to be accurate within the video, without any additional buffering or loading time between each scene. (The lag shown in the videos are because of the powerpoint recording, rather than the program)

Post Development Testing

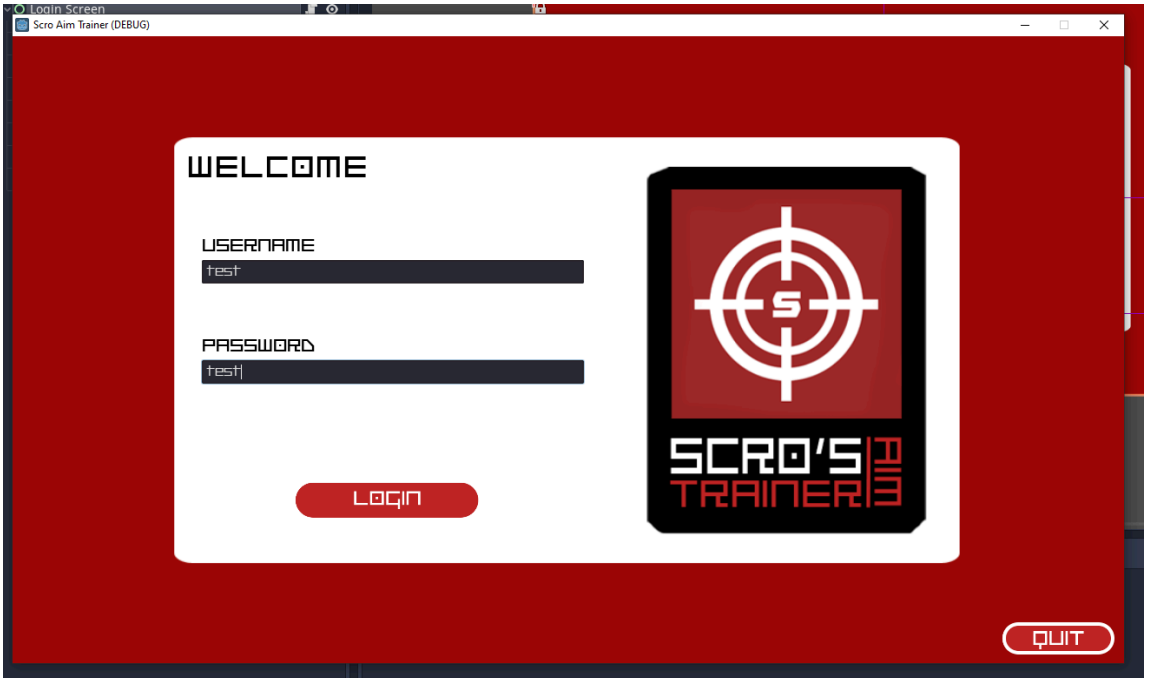
In this final section, I will be doing final tests to make sure everything is working together and linking to each other, with reference to the success criteria.

ID (T)	Description	Code/Input
1	Test to see if the user can type and enter their account credentials for verification.	Data needed = Username, Password When data is correct, Data entered = Username, Password. When data is incorrect, Data entered = IncorrectUsername, IncorrectPassword

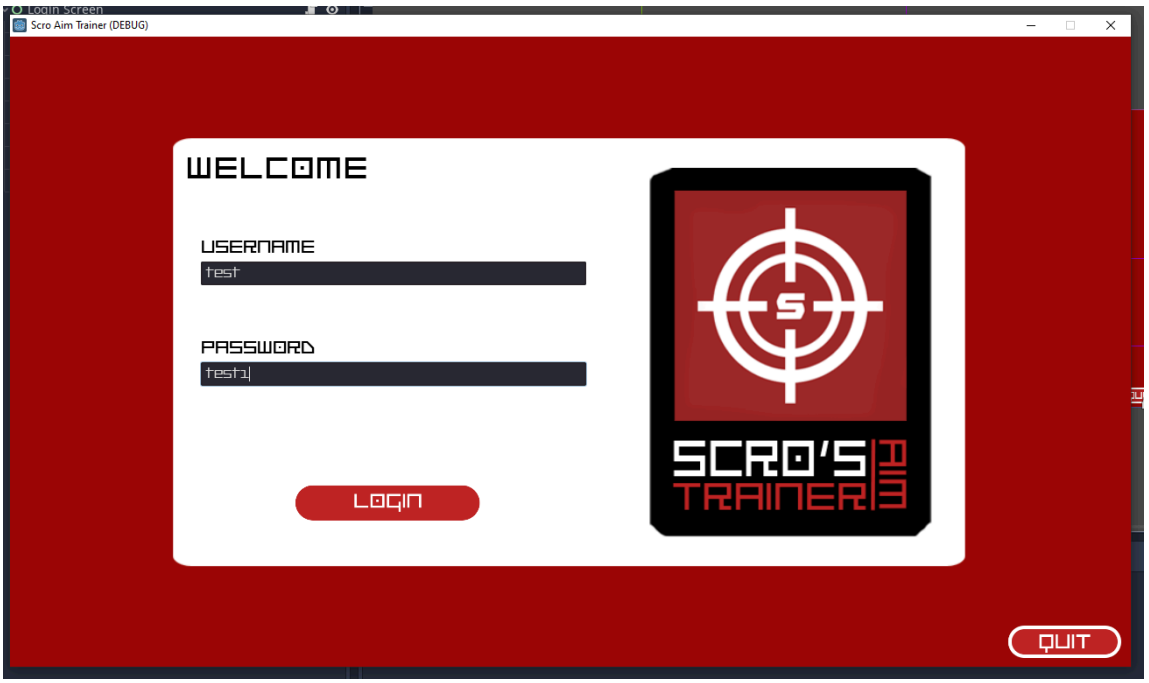
	Result	

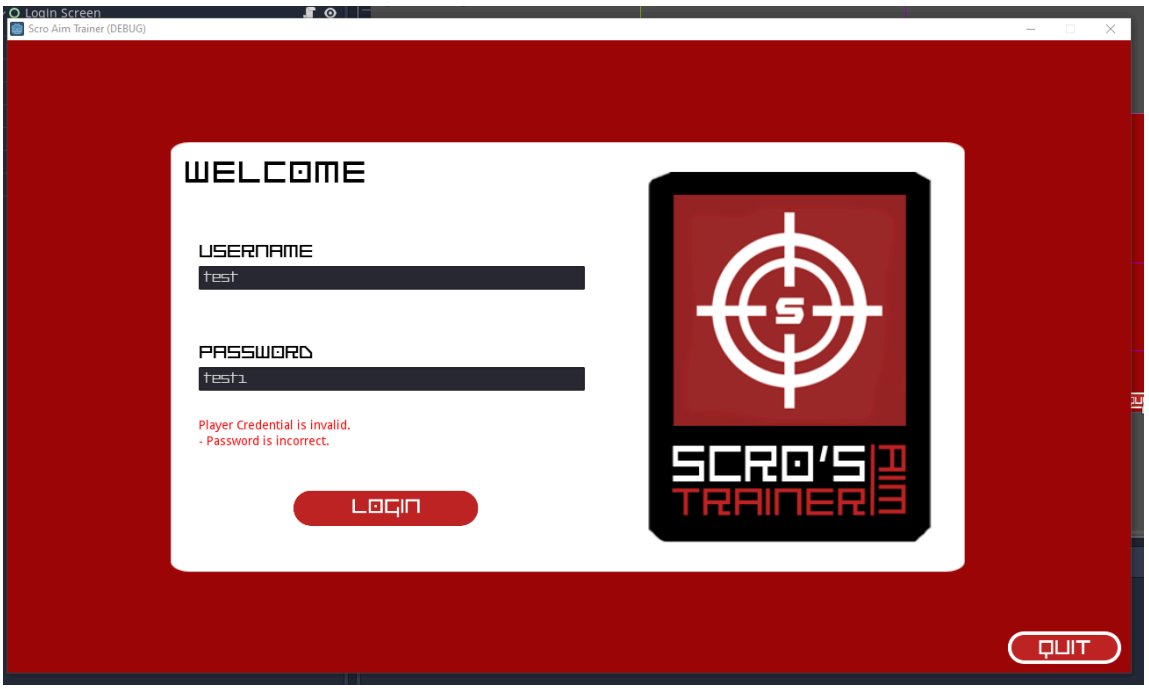
When the user signed in with data that is correct, it should send them to the main menu. When the user signed in with data that is incorrect, error messages should appear depending on the condition.

When data is correct,

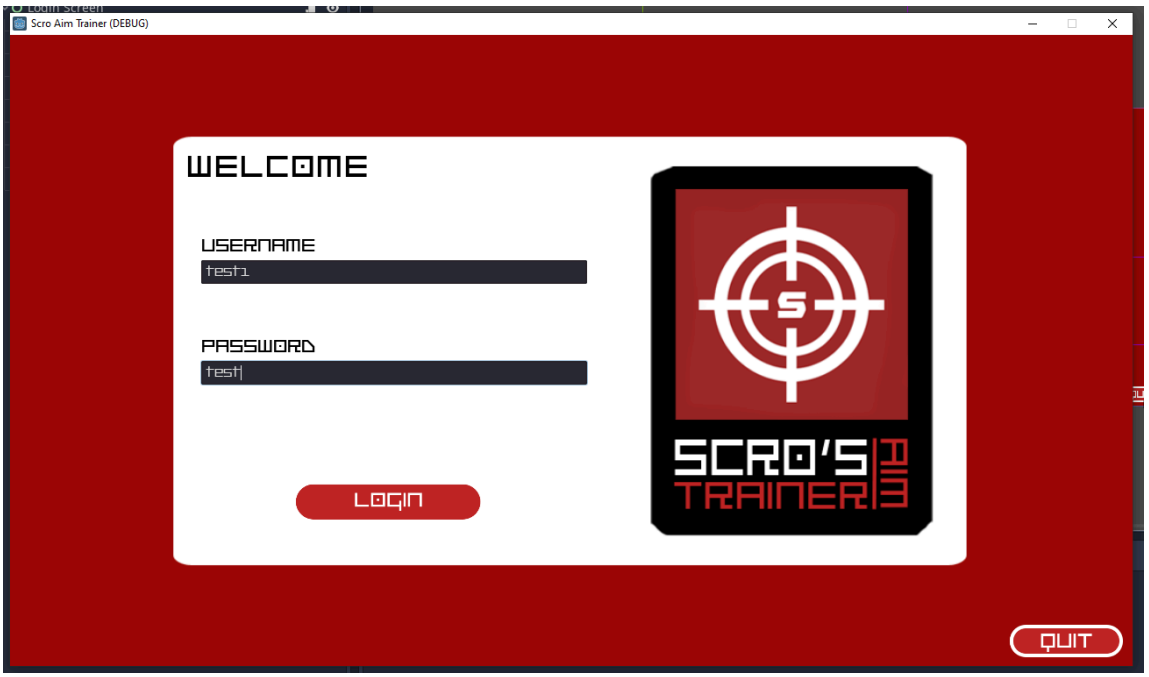


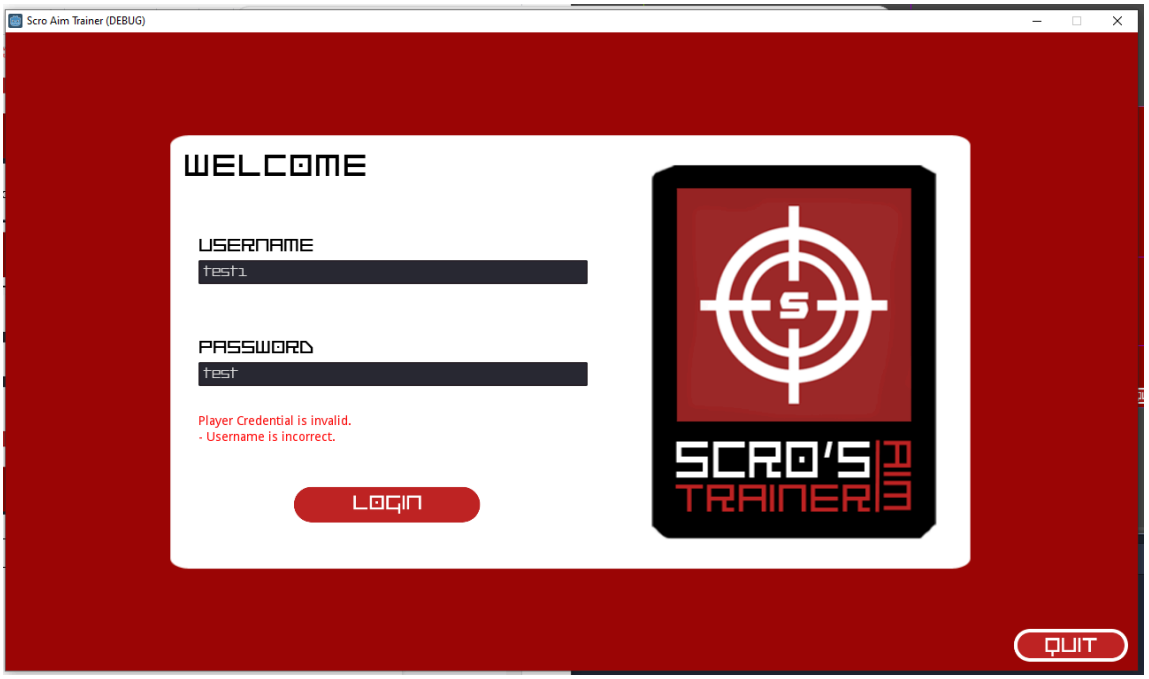
When the data is incorrect due to incorrect password,





When the data is incorrect due to incorrect username,



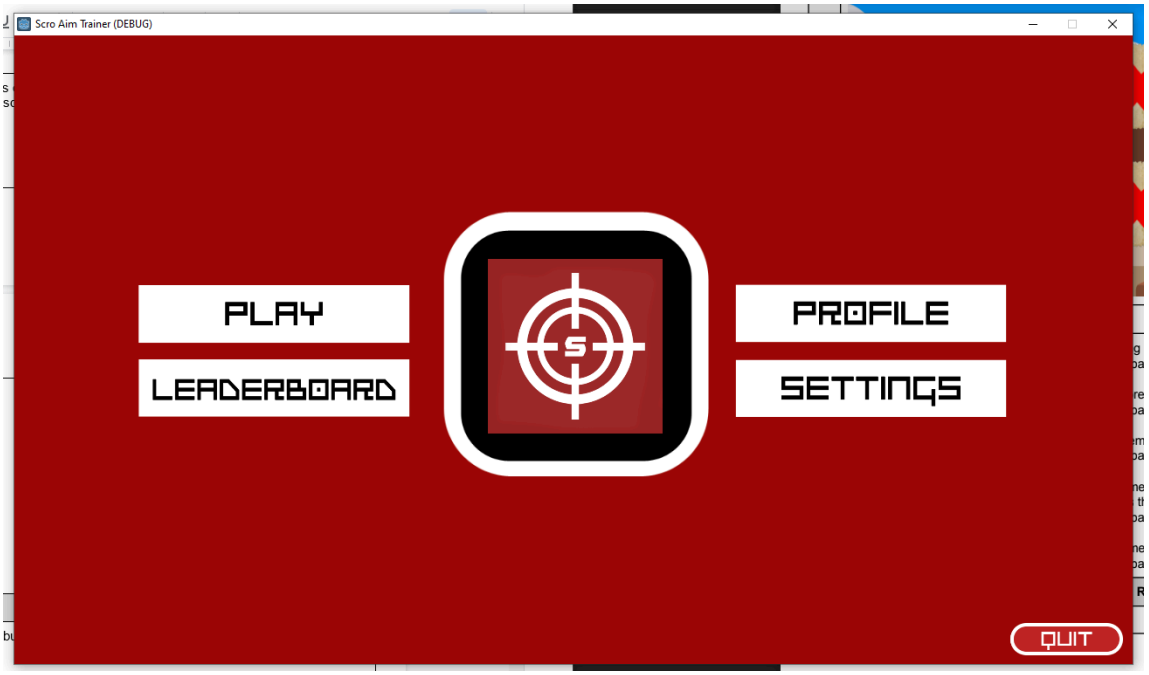


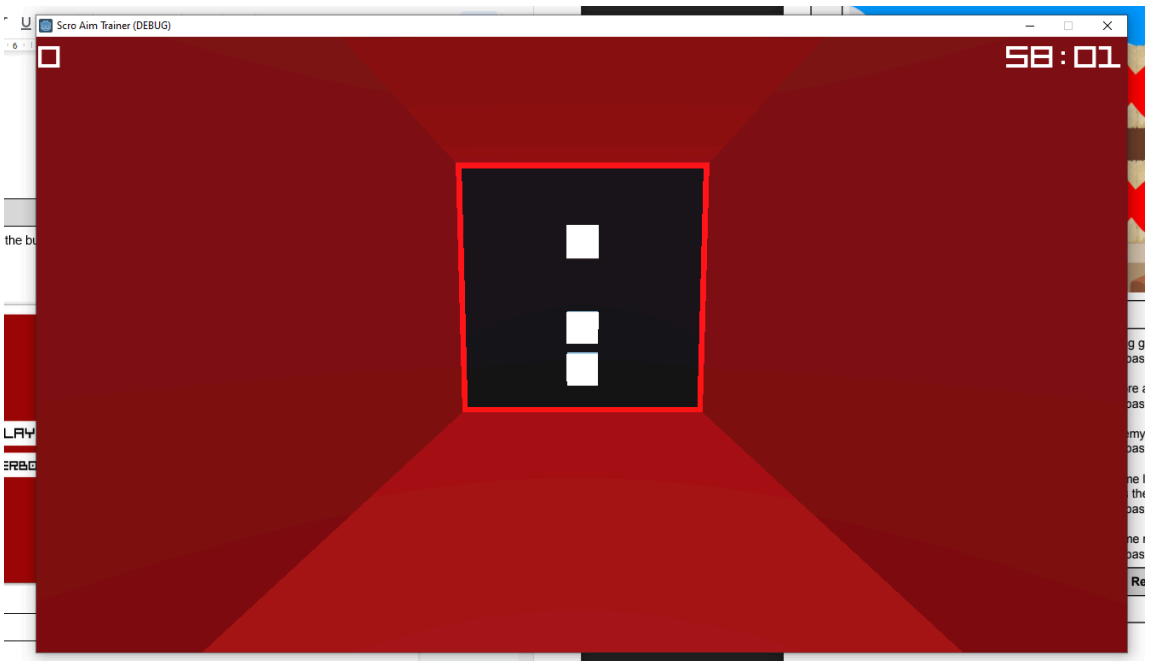
When no data is inputted,

<table>
<tr><td></td><td> 

This meets success criteria 1 and 3 because the login screen displays exactly what is wrong with the login and the database is up to date with the correct login information.</td><td></td></tr>
<tr><td></td><td>Fix?</td><td>N/A</td></tr>
<tr><td>2</td><td>Test to see if the buttons on the menu go to the correct scene.</td><td>When the play button is pressed, the game mode 1 scene should start.

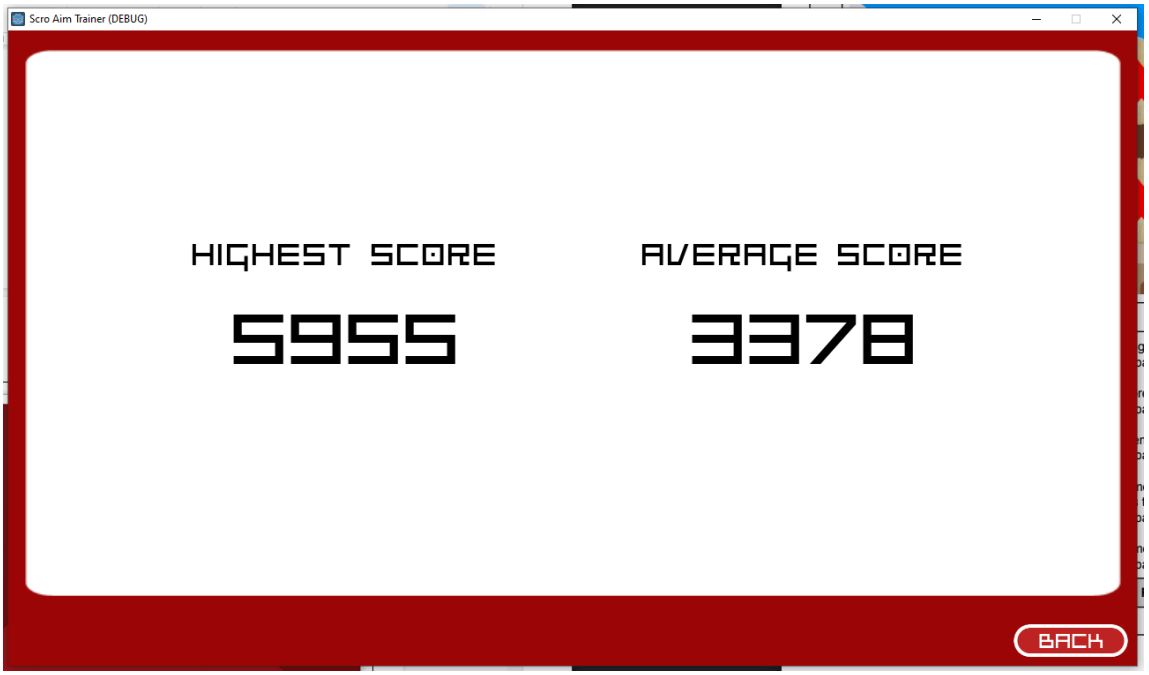
When the profile button is pressed, up to date statistics should be shown</td></tr>
</table>

		When the leaderboard button is pressed, it should not do anything. When the settings button is pressed, the scene should be changed. When the back button is pressed, the scene should be sent back to the main menu. When the quit button is pressed, the program should be closed.
	Result	
	In all the situations, the button pressed will either change or close down the scene. Play button: 	



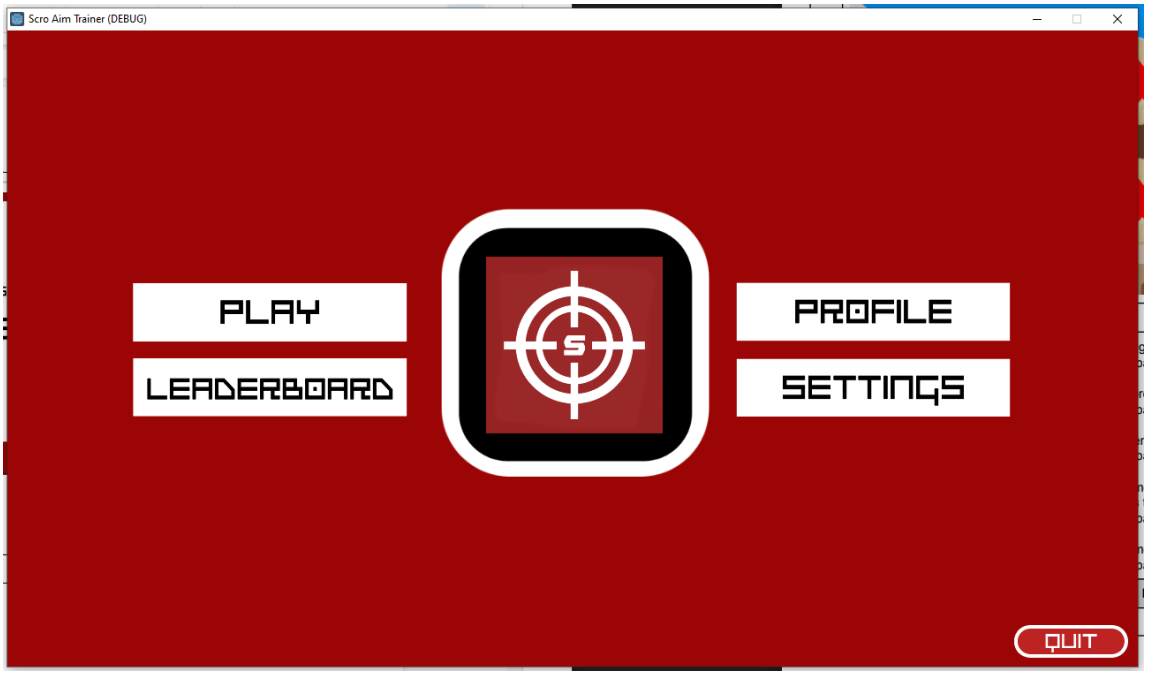
PASS

Profile button:



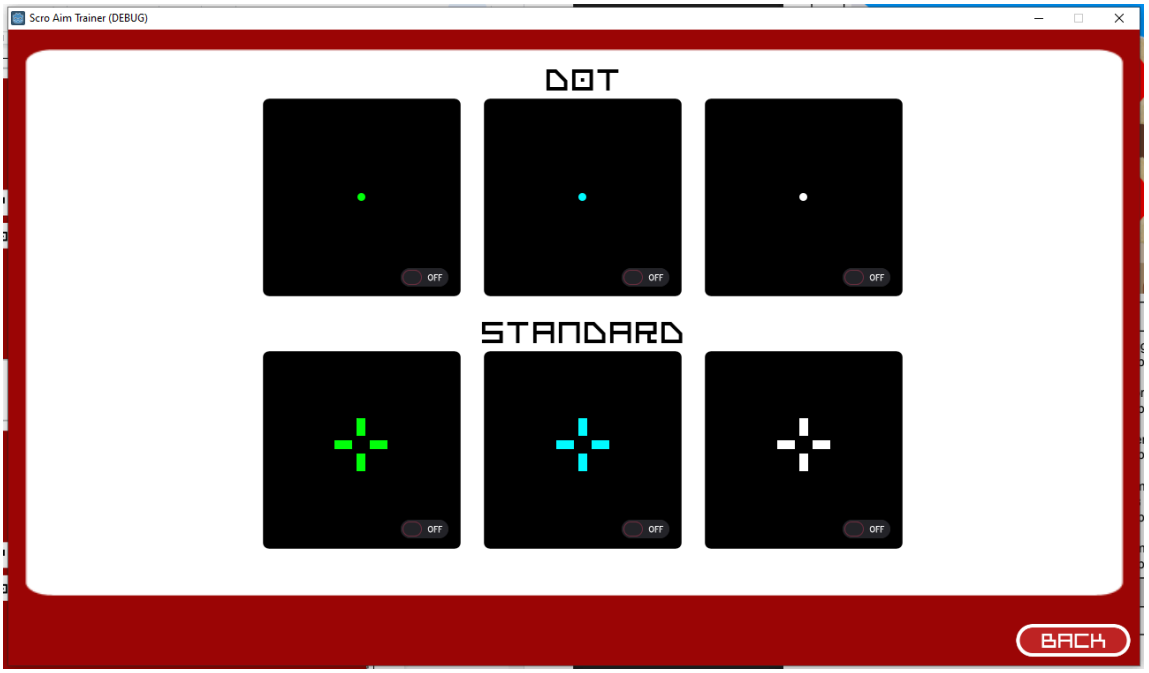
PASS

Leaderboard:



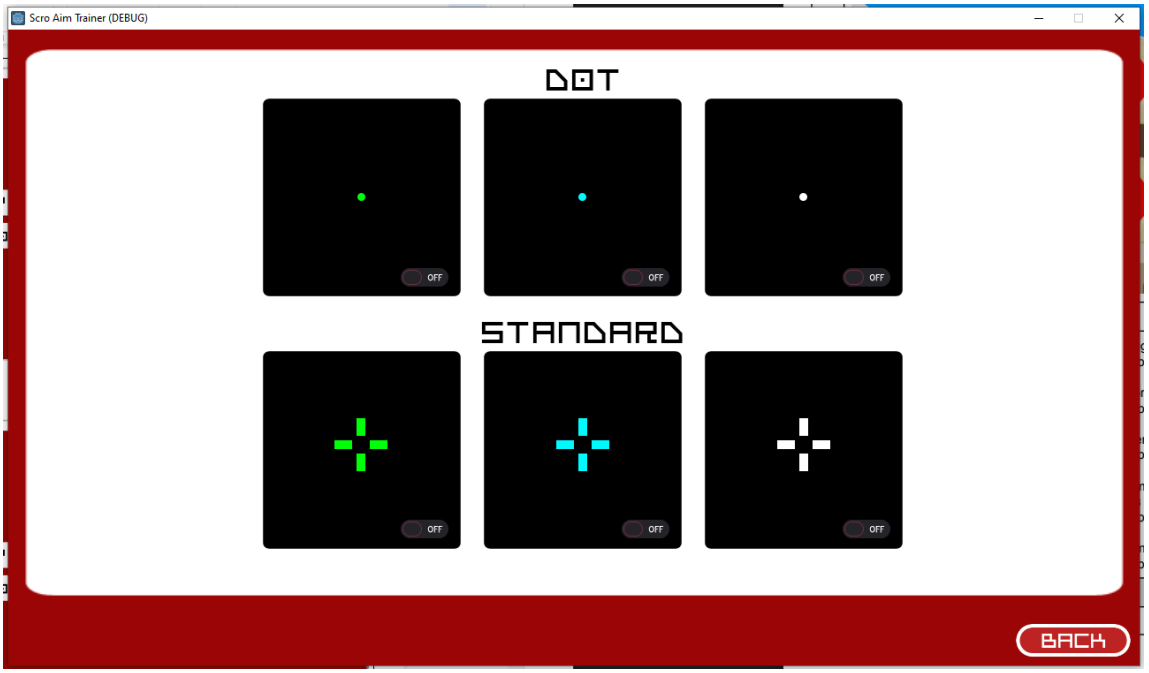
PASS

Settings:



PASS

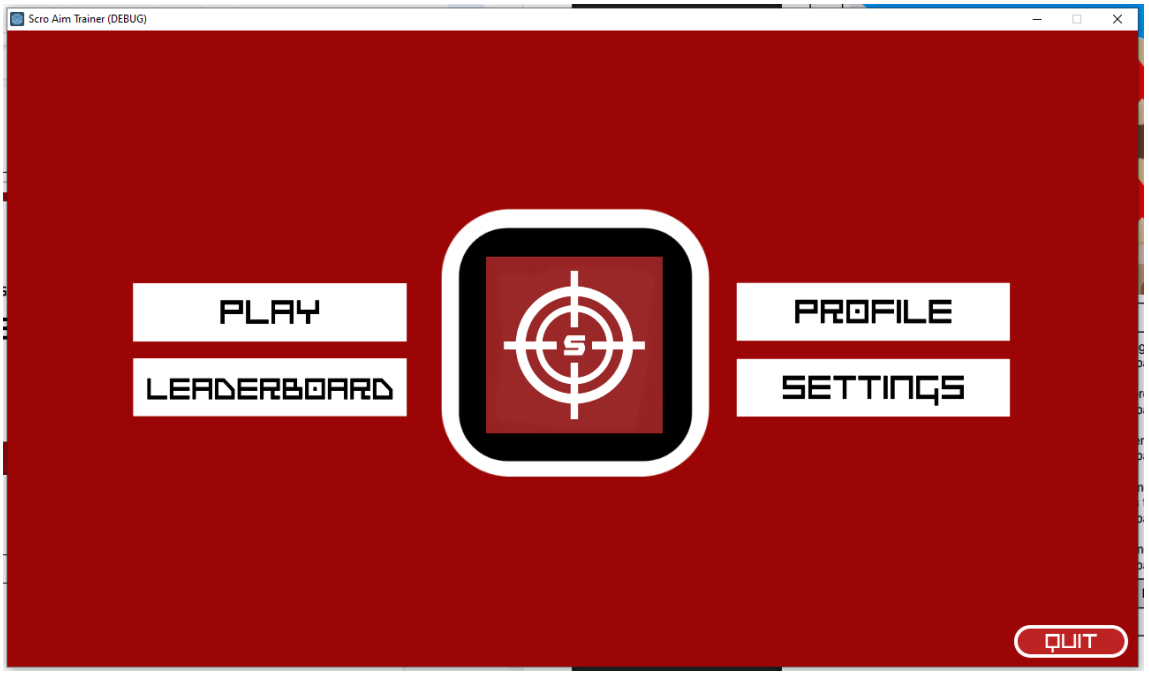
Back button:



(back button is pressed)

PASS

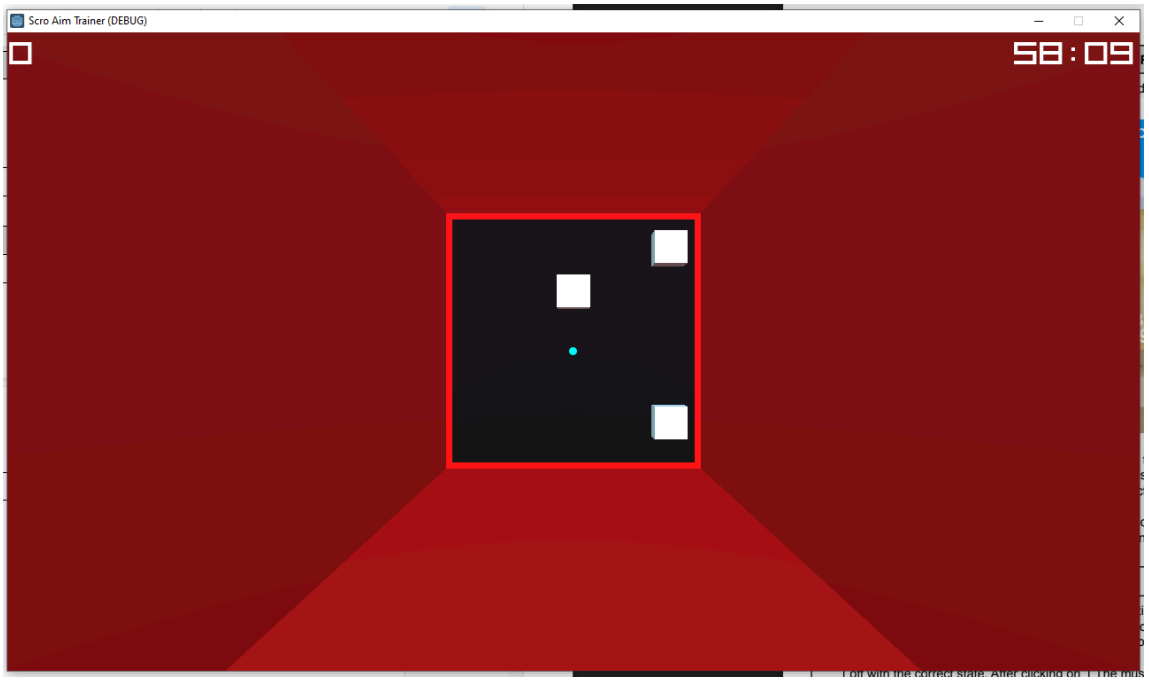
Quit button:

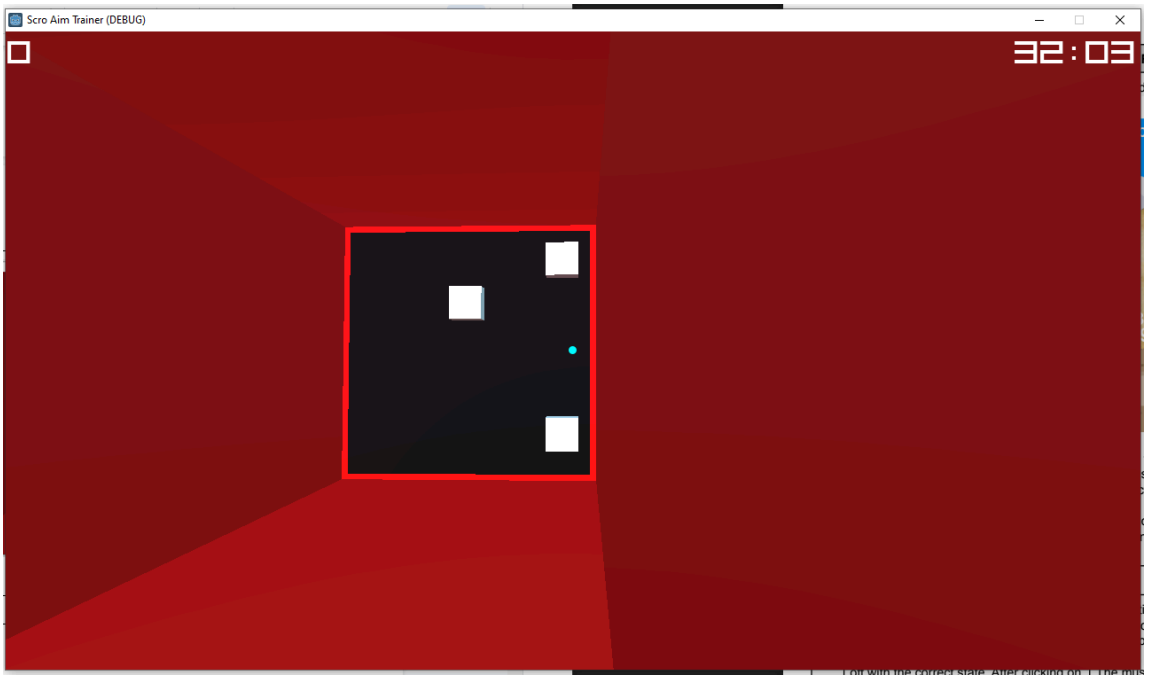


(program is closed)
PASS

This test shows that I have succeeded in the success criteria 9, where the usability of the program is quite simple with the back button instantly changing it back to the main menu. Additionally, this fits in with success criteria 8 because of my modular code, as this would make the game easier to run as there isn't redundant code that might take up the space in the RAM.

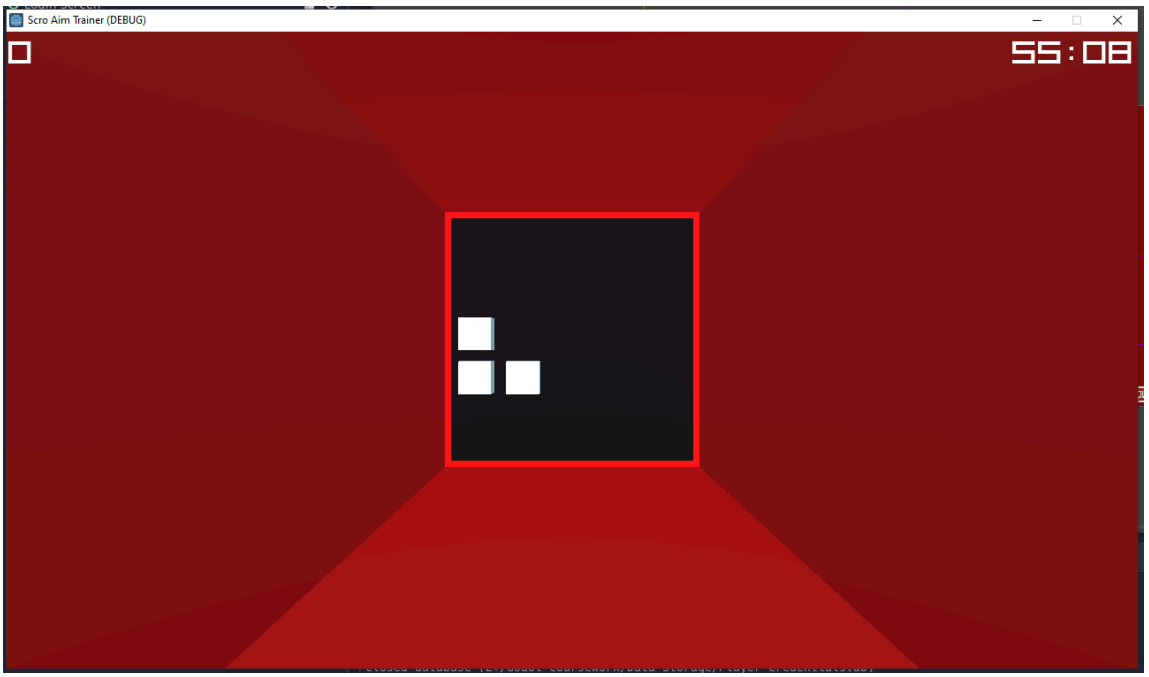
	Test to see if the game mode 1 game scene is working properly with the full functionality that I intended for.	
	Fix?	N/A
3	Test to see if the game mode 1 game scene is working properly with the full functionality that I intended for, excluding the score algorithm for this test.	When the play button is pressed, The user should be able to move using WASD or arrow keys. The user should be able to move the camera around with their mouse. The user should not be able to click on the cubes within the countdown timer. The user should be able to click esc to return to the main menu. The user should not be able to move outside the red container. The user should be able to click a cube when the countdown timer ends. A cube should appear after another cube has been clicked. There should always be only 3 cubes on the screen. The timer should go from 60 to 0. It should change the screen to stats when the timer has hit 0.
	Result	
	User Movement:	

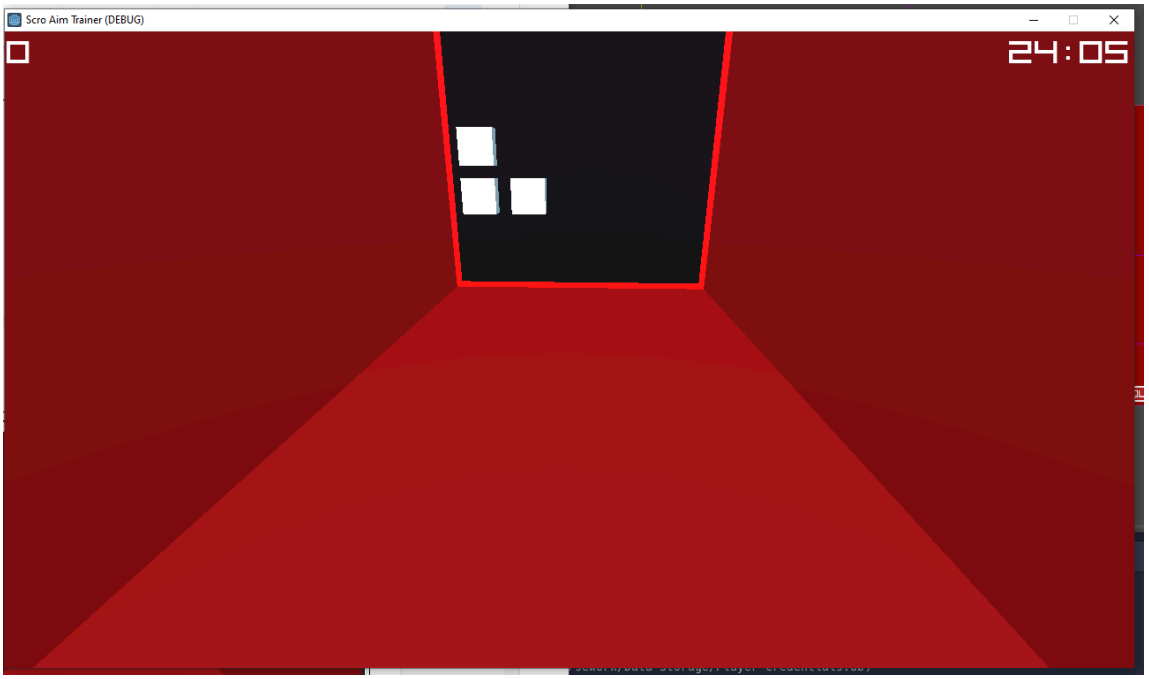




PASS

Camera Movement:



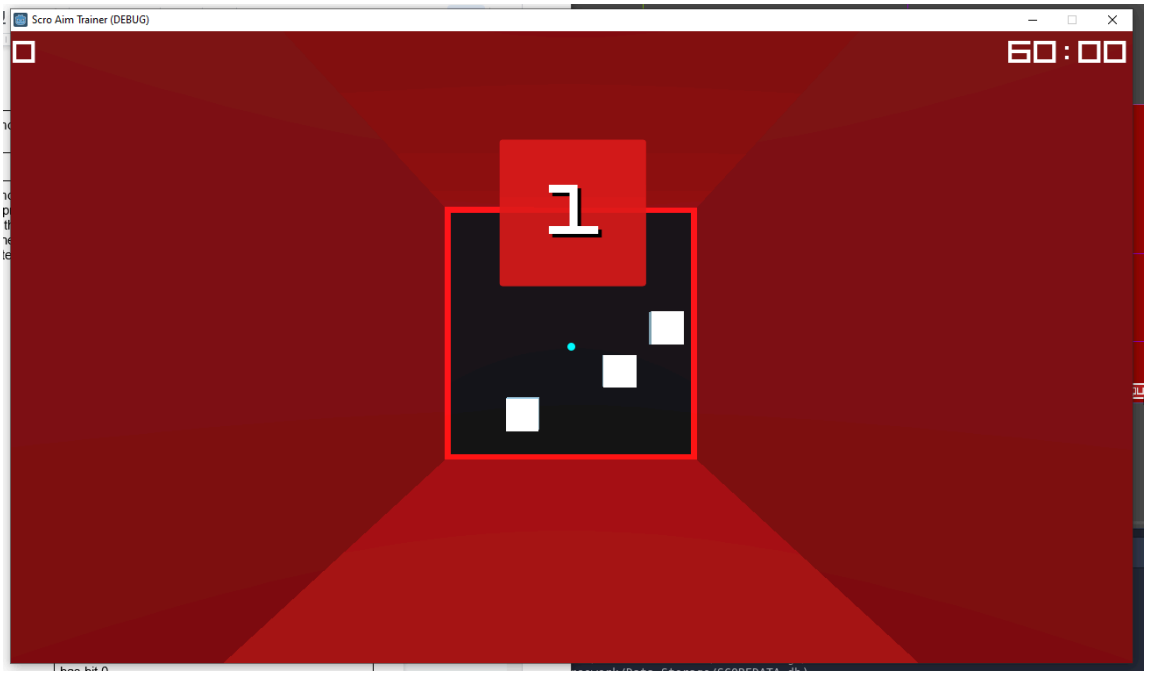


PASS

Clicking on cube during countdown timer:

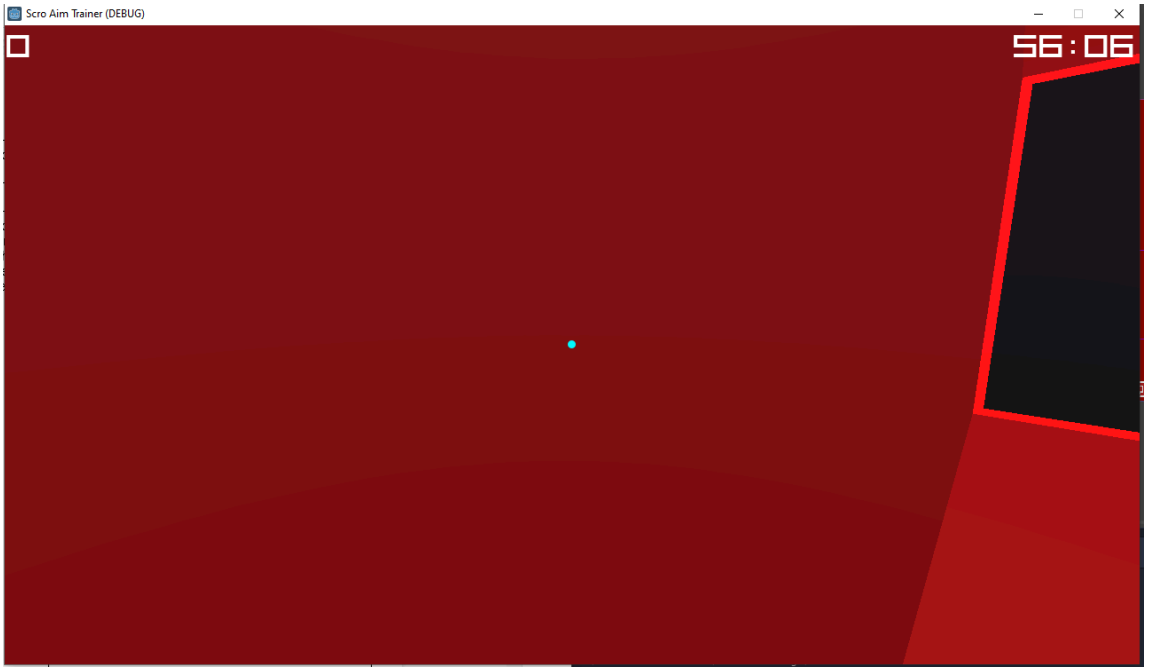
PASS

Esc test:



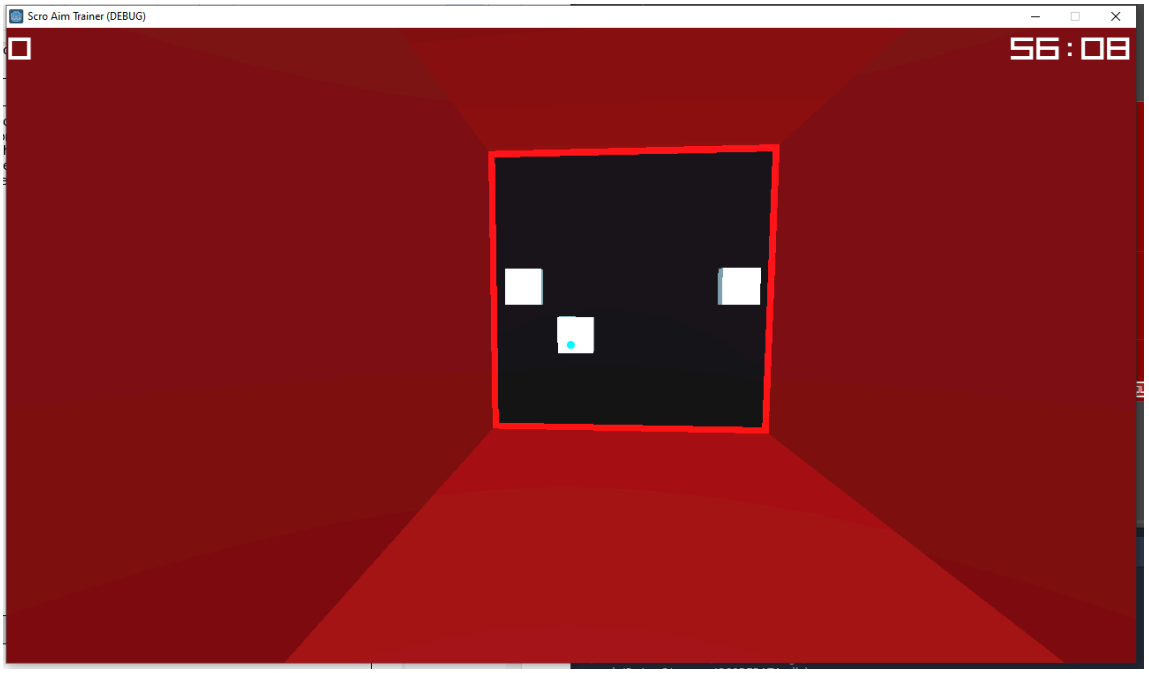
PASS

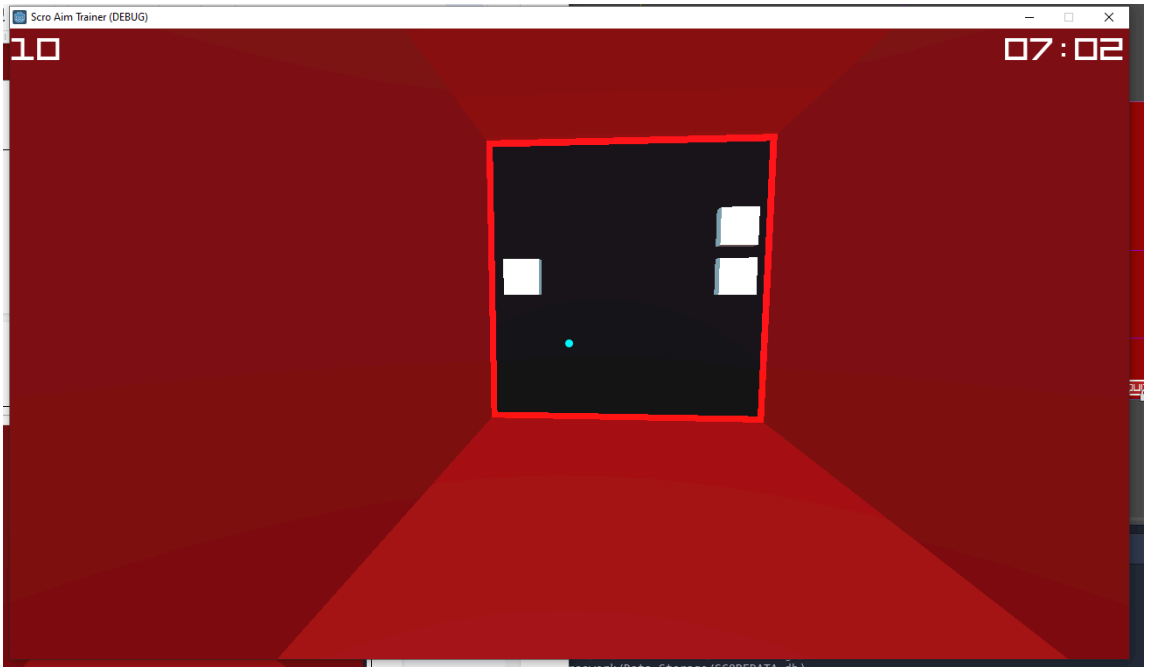
Preventing walking through red walls:



PASS

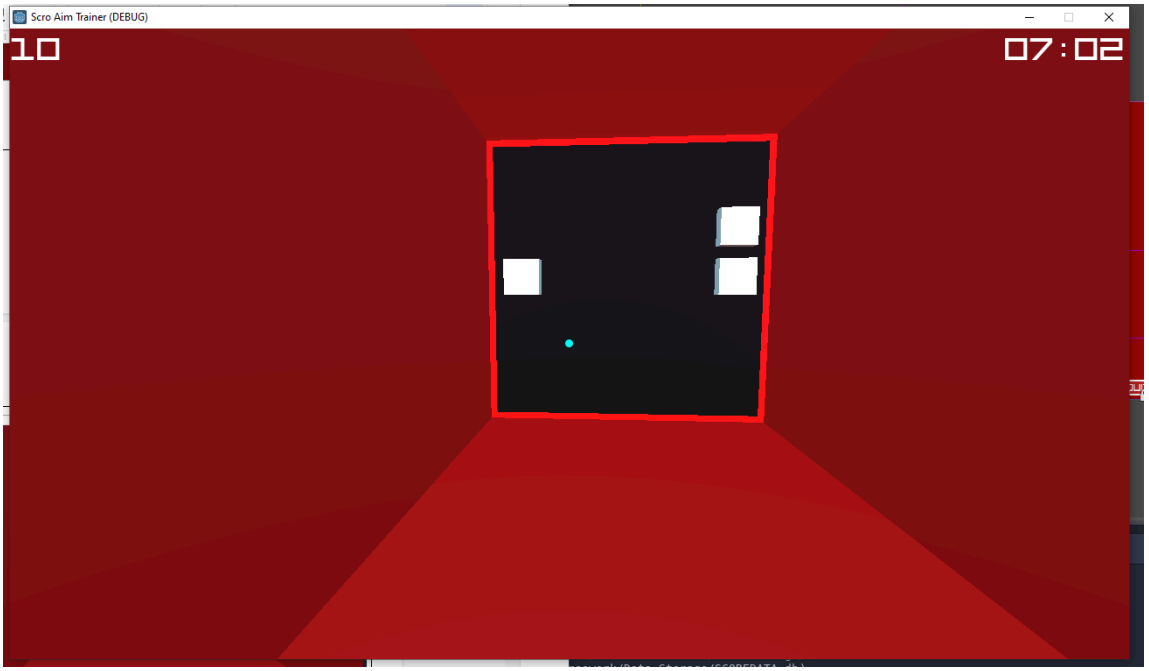
Left click disappear and another should reappear:





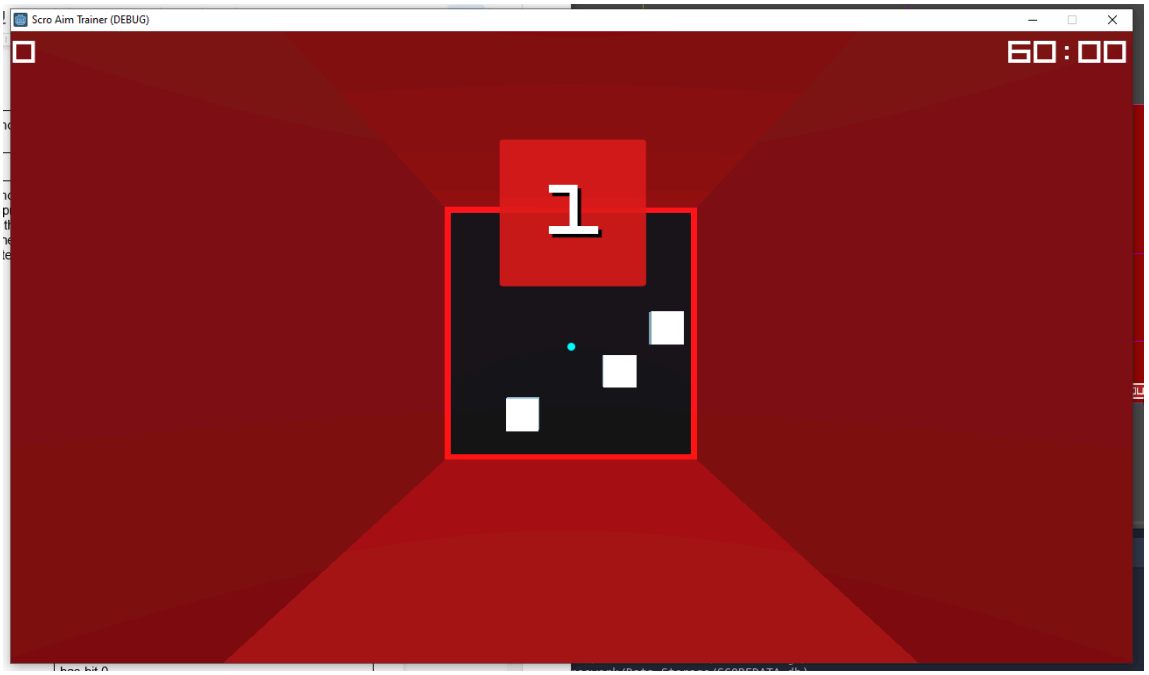
PASS

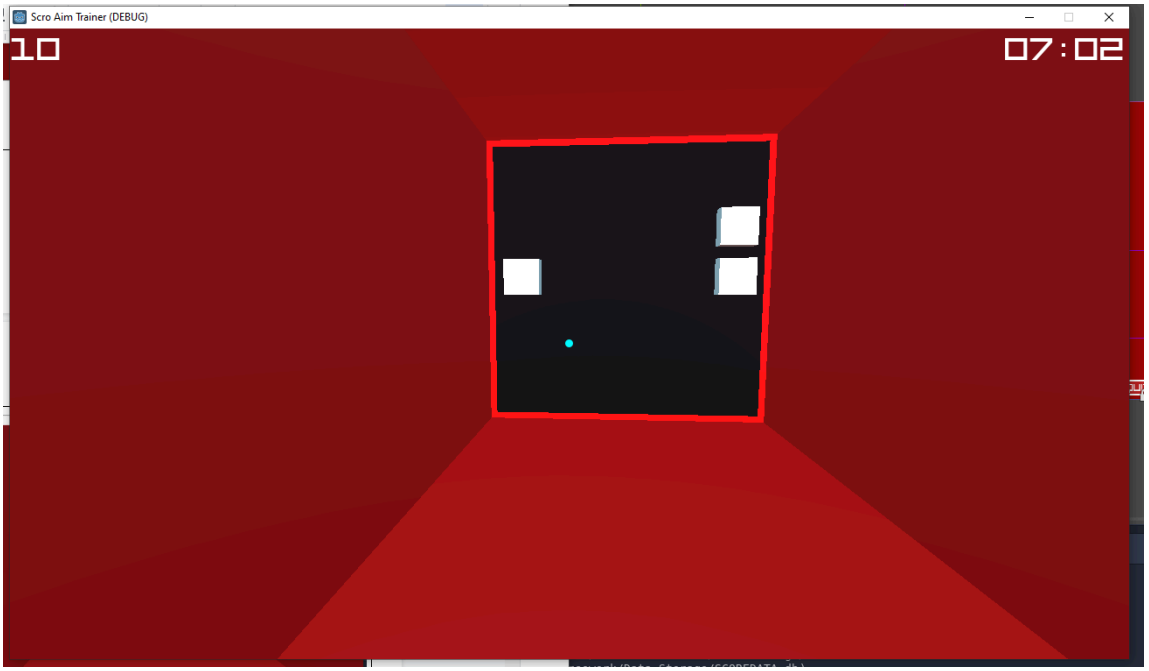
There should only be 3 cubes on screen:



PASS

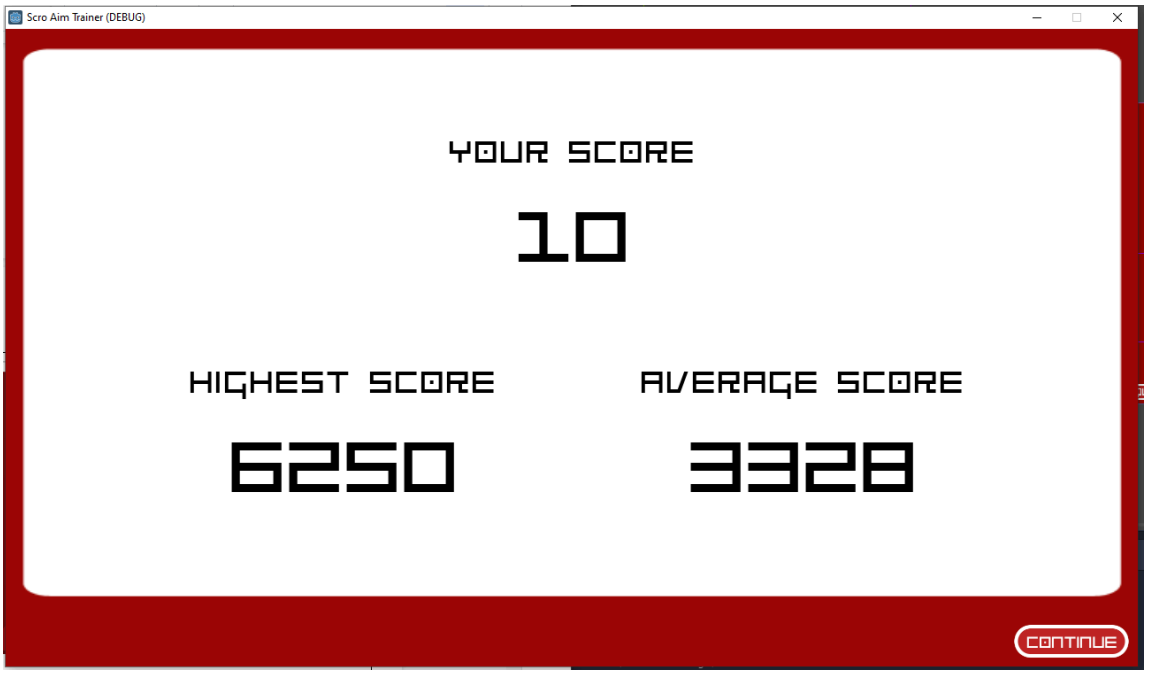
Timer should go from 60 to 0:





PASS

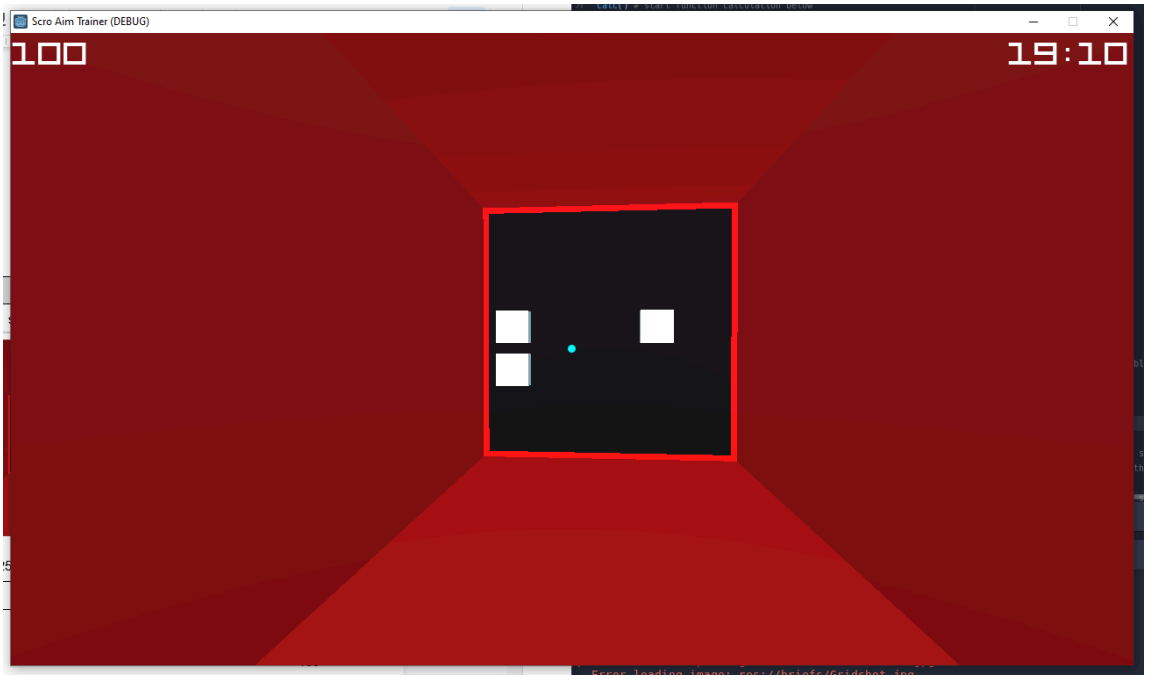
Changing scene:



PASS

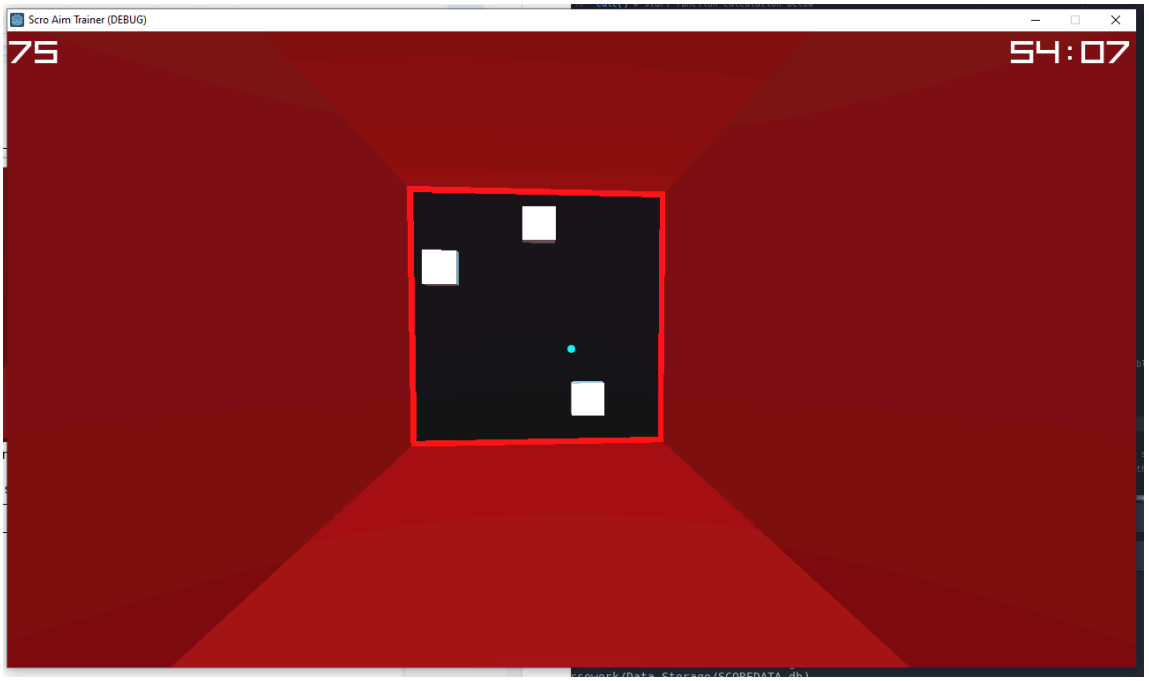
In all of these tests, it all confirms the success of success criteria 5, 8, and 9. With success criteria 4, I have successfully created a fully functionable game mode that works with everything that I wanted it to have, albeit partially because I haven't fully completed all 3 game modes. However, by focussing my attention into finishing this game mode so it has no issues, the user will have a flawless and fun experience. This

	ties in with success criteria 9 partially, where the user only needs to use left click to get score, plus I added features that made the user more comfortable such as a customisable point of view, user movement, and simple cube targets. For success criteria 8, I have made the game scene very compact with the red walls, so the game will be easy to run on any computer.	
	Fix?	N/A
4	Testing the score and data saving algorithm within the game scene.	In the scene, If the user takes <= 0.01 seconds between each cube hit, the score is incremented by 100. If the user takes <= 0.025 seconds between each cube hit, the score is incremented by 75. If the user takes <= 0.05 seconds between each cube hit, the score is incremented by 50. If the user takes > 0.05 seconds between each cube hit, the score is incremented by 10. If the time = 0, the score counter should be written and saved in the database, with the high score updated.
	Result	
	If the user takes <= 0.01 seconds:	



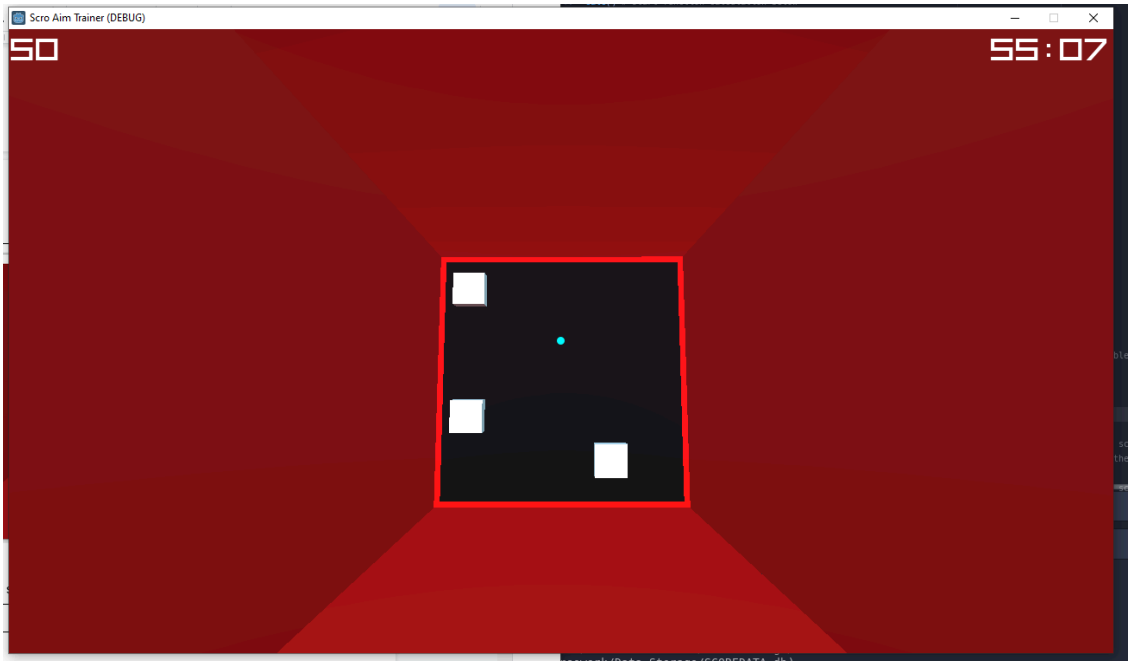
PASS

If the user takes <= 0.025 seconds:



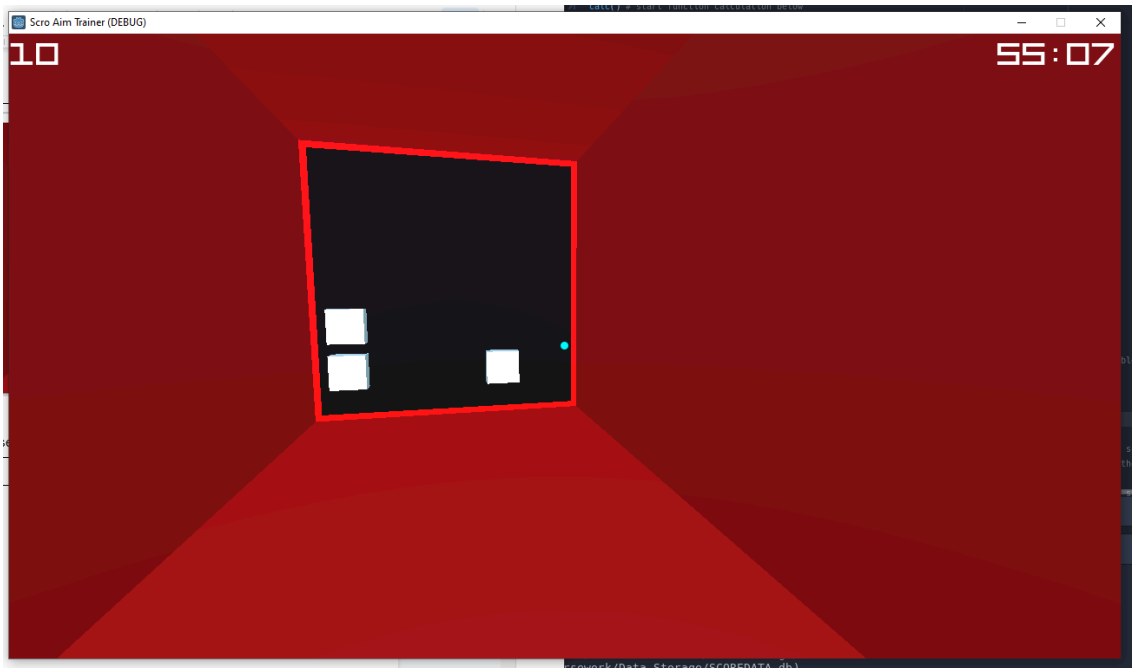
PASS

If the user takes <= 0.05 seconds:



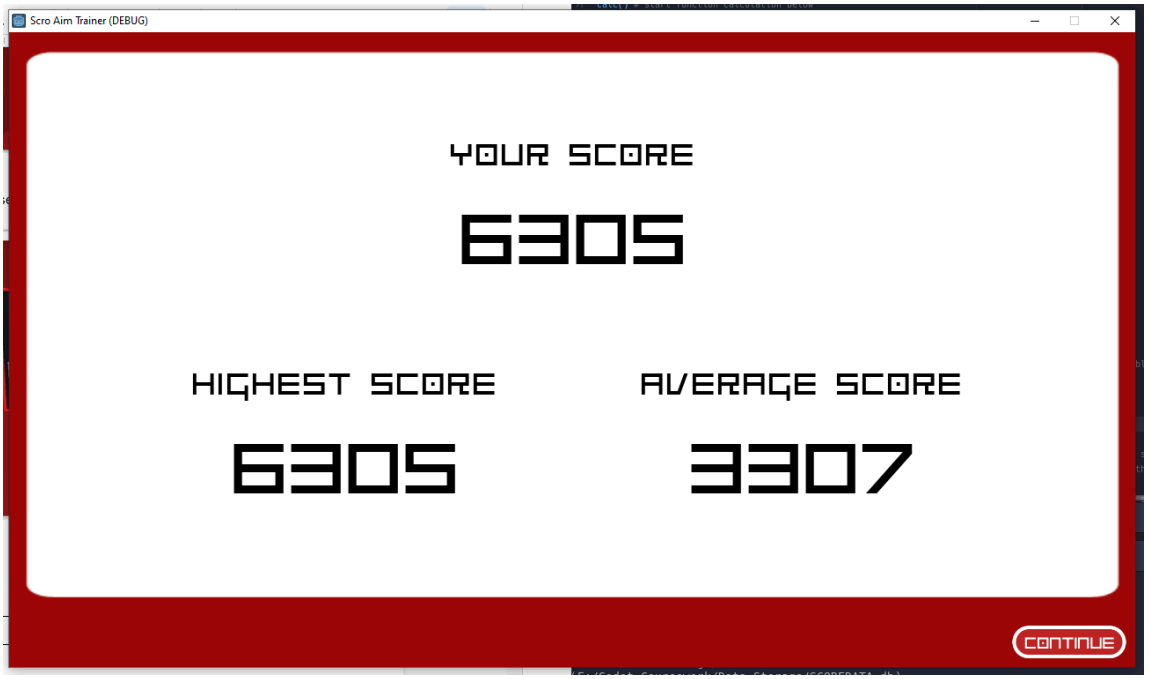
PASS

If the user takes > 0.05 seconds:



PASS

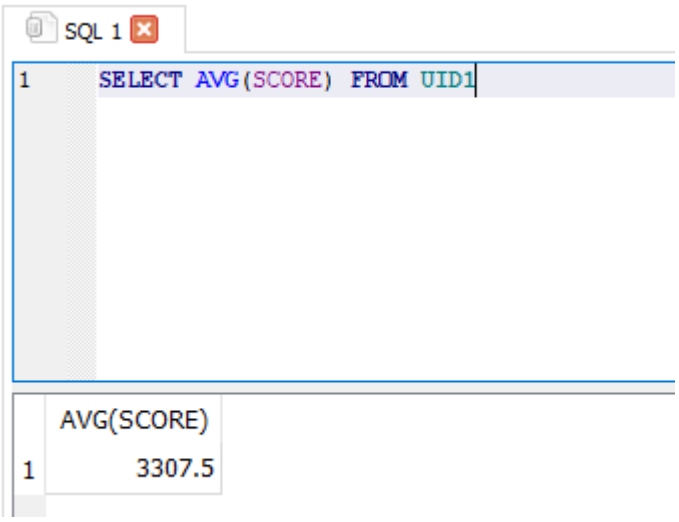
If time is 0:

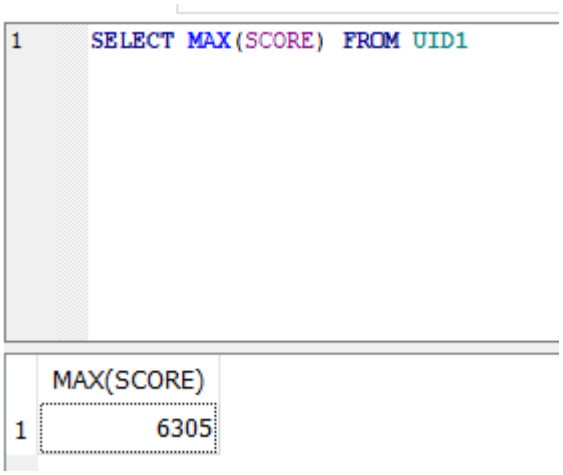
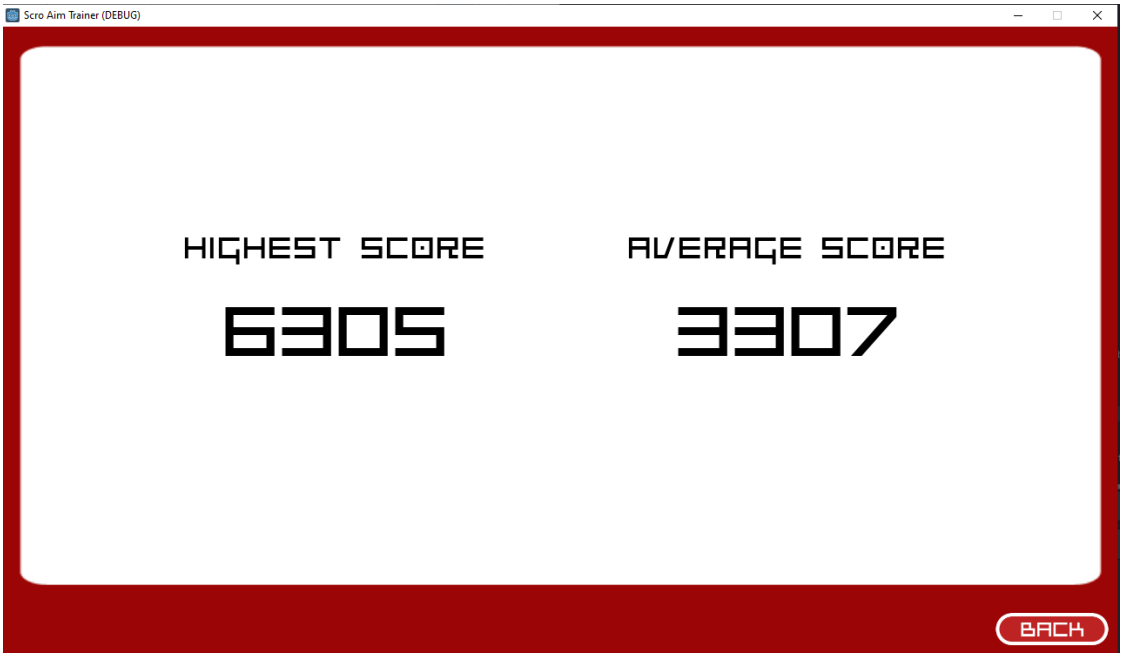


PASS

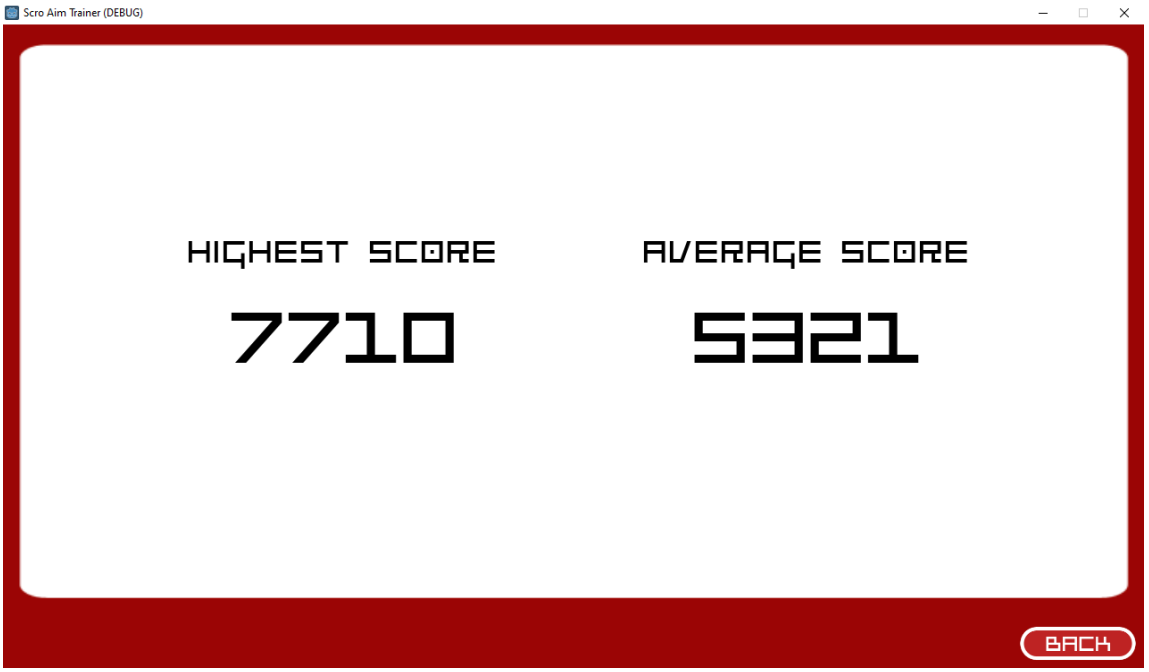
In every one of these tests, it completes either success criteria 2 or 3. For success criteria 2, the score algorithm is implemented, with the multiplier being in place to reward those users with high reaction time and accuracy. This not only completes this success criteria, it also links back to my analysis section, as this is what I intended to do all along from advice with my stakeholders. For success criteria 3, the database is being kept up-to-date as seen in the last screenshot, where the highest score updated and the average score changed. However, I do have to say that graphs weren't made as I initially wanted, but the feature is a less important one as it doesn't massively impact the project, so I can add it if I wanted to later.

(Proof:)



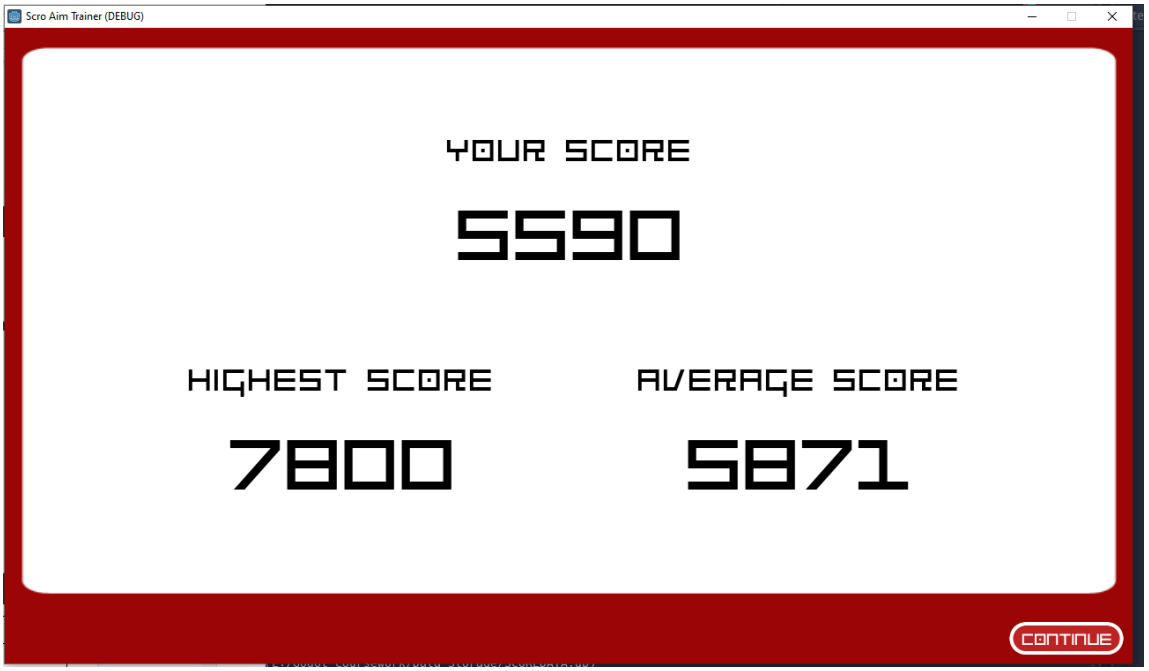
		
	Fix?	N/A
5	Profile	For this scene to be successful, It must be the most up-to-date data for the respective user when it is loaded.. It must change depending on any new results.
	Result	
	Consistent/Newest data depending on user: User 1: 	

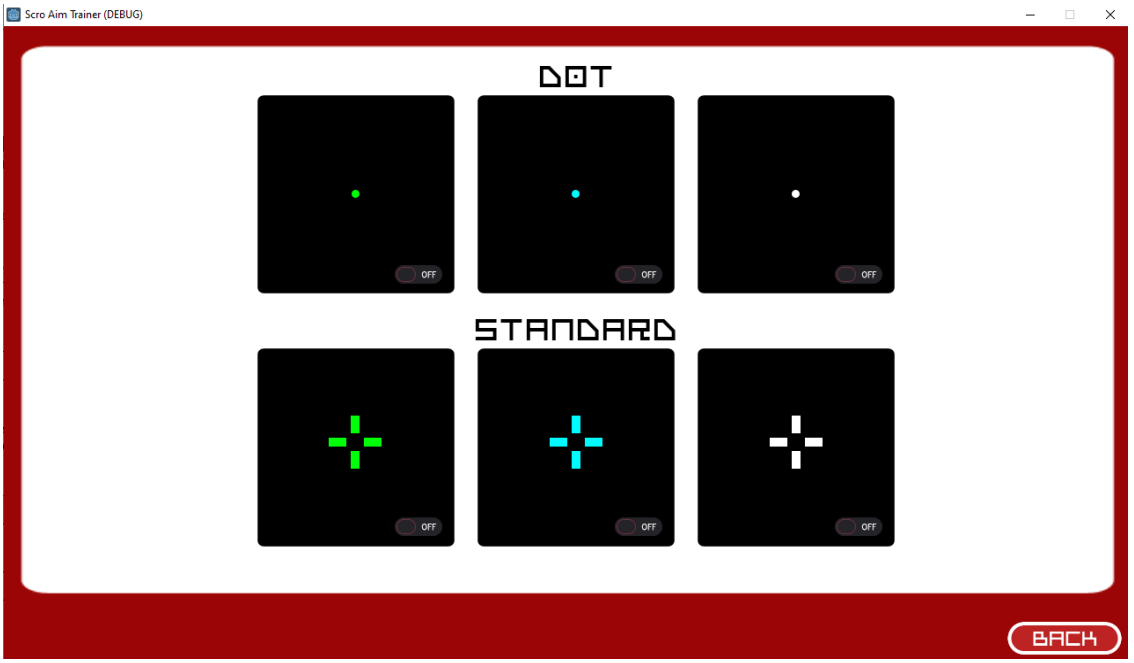
User 2:

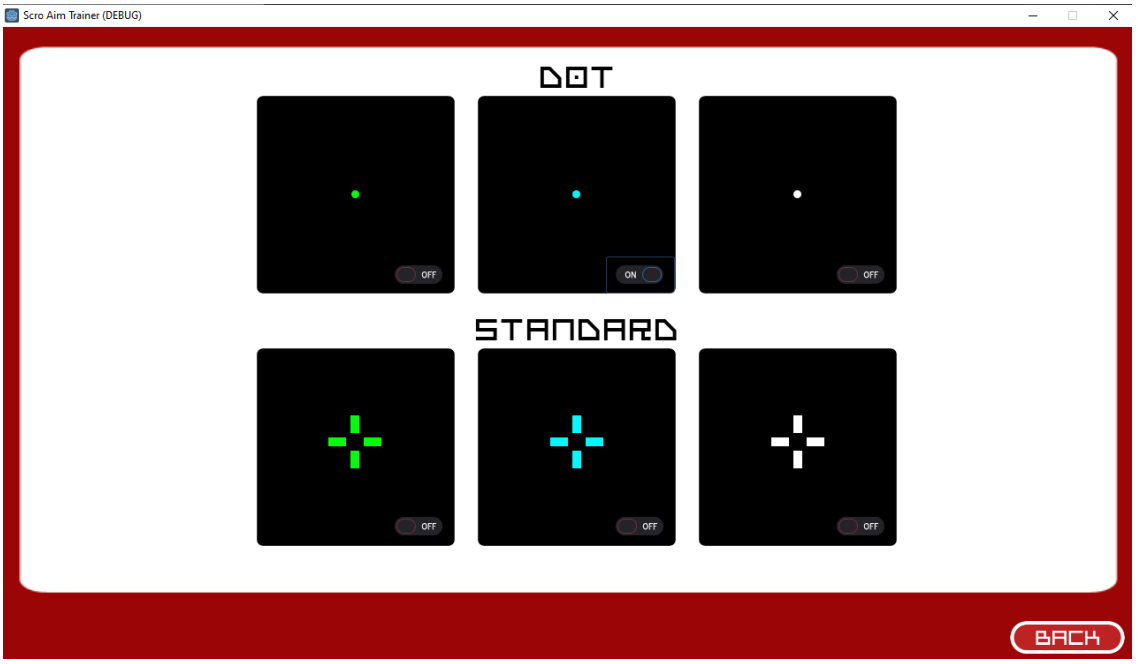


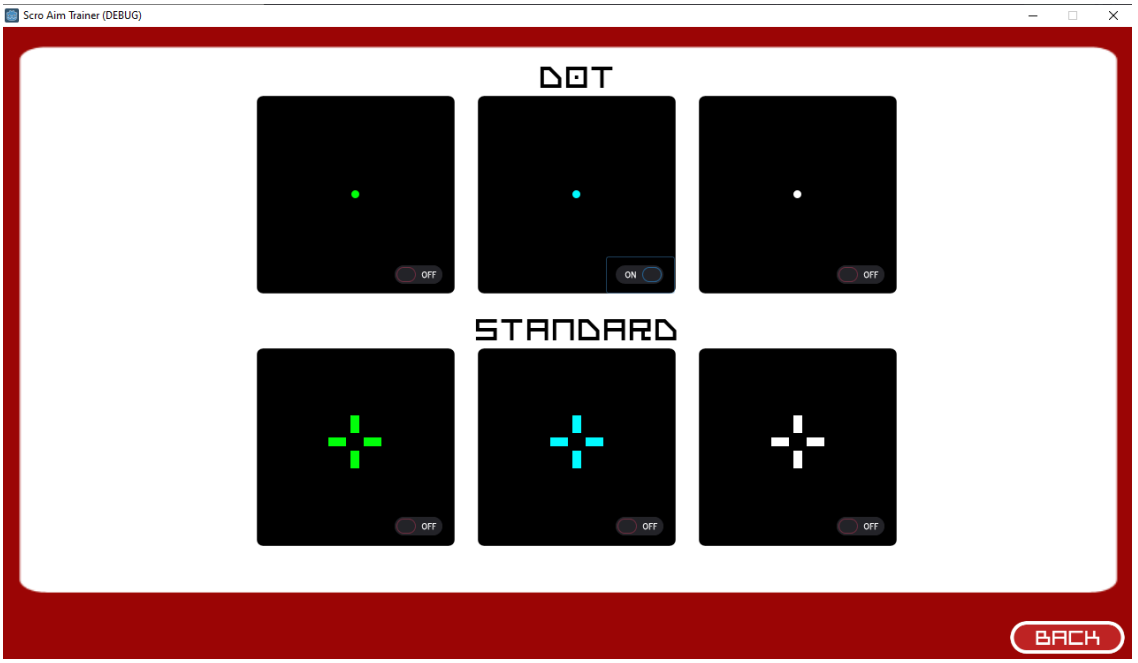
PASS

User2 stats changing depending on new results:

	 PASS With these 2 tests, it perfectly showcases success criteria 3 and how the data of each user are different and how each new result is updated in the profile and the stats screen after the round is played. This means that the user gets as accurate information as they will get, and it will save even if I close down the program as this is calculated each time the user goes to one of these scenes. Additionally, we can see that User 2's profile has a higher average score and high score, meaning that we can conclude that their aim is more consistent and more accurate/faster than User 1.	
	Fix?	N/A
6	Settings	In the settings menu, We simply need to test if the crosshairs will save the data that was inputted even if we leave the scene page and go back to it.
	Result	
	Settings load data:	


Scro Aim Trainer (DEBUG)
DOT
OFF
OFF
OFF
STANDARD
OFF
OFF
OFF
BACK


Scro Aim Trainer (DEBUG)
DOT
OFF
ON
OFF
STANDARD
OFF
OFF
OFF
BACK



PASS

In this test, it is simple but very important for the user. This is for success criteria 9, where the settings mode is saved even when the scene is changed. This is because godot unloads each scene if it is not in use, meaning that my crosshair would not have saved unless I saved it to a global variable and loaded it each time the scene is changed to it. This allows the user to turn off and on each crosshair whenever they like, without having to reload the program each time they want to try a new crosshair. Furthermore, this also ties in with success criteria 8, because each scene is unloaded when it's not in use, so the computer does not need to load more than the user sees (excluding global variables).

	Fix?	N/A

Success Criteria

1. Creating a Login Screen

Having a login screen will allow the user to login to their own specific profile, which would be the first requirement to gain access to the game. As such, it links with requirement 4, mentioned above. This would be vital as it then sends them to the actual interactive part of the game of the home page, which will consist of buttons sending to other pages such as settings, leaderboard and training area.

I have successfully met this criteria, as I've created a login screen UI that checks the login credentials of the user is correct. If they are correct, it leads them to the main menu. If the information is incorrect, it displays an error message. This is evident in test 1 in post development testing.

2. Score and Multiplier

Creating the algorithm to track the multiplier will need to be completed successfully, as this is the only indicator in the game to where their skill level is at. This requirement links with 2, where the user can see an increase during their round of training.

I have successfully met this criteria, as I have designed and created an algorithm that gives more score depending on how fast the user reacted between each of the cube hits. Additionally, this is then sent off to be written into the database, so the user will be able to see their average score and try to increase it. This is evident in test 4 in post development testing.

3. Keeping database accurate and up to date

The final score after each of the individual rounds the user plays will be noted down and summarised to the user using charts and graphs to show weaknesses and strengths. This links to requirement 3 and 8, where it is vitally important for the user to see their improvement in a specific category. Therefore, charts and graphs show round data and average data to compare with so the user knows if they performed over-average or under-average.

I have mostly met this criteria, as database integrity is kept up to date and easily transferable within each scene. The data is saved into tables depending on which user is playing the round, and the statistics change every time the user loads up either the profile or stats screen. This is evident in test 4, 5, and 6 in post development testing. However, I did not

manage to create the graphs that I intended from the analysis, but now I think these would not be as vital as other features in the game such as the settings scene.

4. Successful make the 3 gamemodes

The 2 different gamemodes will consist of differently unique training that consists of "Snipershot","Motionshot" and "Gridshot", which relates to requirement 1. I have chosen these from doing further research into why each of them are effective, and these are the best and unique options that I have seen that helps train the basic skills that a user would need to use, so these are quite important to succeed in.

I have partially met this criteria, as I have created a fully functioning game mode that the user can repetitively play to try and increase their general skill in reaction time and accuracy. I have decided, alongside my stakeholders, that it would be beneficial to create a game mode with tons of features and a highly-complex algorithm than to make multiple game modes that only provide the basic features and possibly redundant code. This is evident in test 3, and 4 in post development testing.

On account of this, I still have not completely met this criteria. To address this, I could have developed solely on each of the game modes first instead of doing the login screen and main menu first because these were the main areas that I spent the most time in. In the specific success criterias behind both Motionshot and Snipershot, I have articulated why I have not done this and how I would've achieved it with further development.

5. The Algorithm behind Gridshot

The algorithm behind gridshot should be a smooth experience for the user by adding a good collision between the crosshair and the cube. This is especially important in this game mode, as the user will be flicking rapidly, so the hit detection needs to register correctly so the algorithm can pick it up. This relates back to requirement 2.

I have successfully met this criteria, as this was the one game mode that I had focussed on intensively. Making the game mode within 3D allowed me to develop to what I have done in the design phase, plus adding additional features such as collision with the walls, a countdown timer, and score multipliers. This is evident in test 3, and 4, in post development testing.

6. The Algorithm behind Motionshot

Motionshot is a tricky concept because it requires the user to hover over a cube and track it to wherever the cube goes. This means that I will need to change the collision detection on this specific cube so any part of the cube, including the edges, are detected as collision. Additionally, to add more difficulty to this game mode, the cube can accelerate in the direction it intended to go towards which in return would grant the user with more score. This relates back to requirement 3.

I have not met this criteria. This is because I had spent all of my time creating all the new features within the first game mode, and it resulted in me not having enough time to develop the rest. However, the main input and aim register algorithm have already been created once, and since the code is modularly designed with functions, I can re-use these sections of code that I have written. The motion would be the only problem that I would have to face, but this would have been done using a random number generator to generate the random direction it would head in, and if the acceleration was high, then the score would have a high multiplier.

In addition to all the back-end algorithms, I would have needed to create the front-end game scenes that would have the same style and functionality as the first game mode, but with their own specific design. This would not have been met successfully because of this due to the fact that another design would need to be made so it can be developed.

7. The Algorithm behind Snipershot

The last game mode is a unique iteration of the gridshot cubes. More elements can be re-used in this application, with the difference of the range of random sizes for the cubes and the area of play being a lot larger than the previous one. Similar to Motionshot, the collision detection needs to be very accurate, as the collision of the smaller cubes will be really hard to hit fast, or I can make the collision of the smaller cubes slightly bigger to prevent any of the users struggling heavily on one of them. This relates back to requirement 4.

I have not met this criteria. This is because this specific game mode would have been easily made from using the same exact algorithm in the first game mode and only change the size of the cubes. I didn't think that this would have been worthwhile to do, since the game would only be harder to run and process, for a game mode that lacked demonstration of my programming skills. Whilst it would have been good for the users to have another game mode, I think prioritising the database and global variables was the better choice, as this will hone their general skills that all users need to be proficient in.

What I would have done to create this would be to have a much larger playing field compared to gridshot, so people would need to spend more time clicking each one as accurately as possible. This game mode would probably achieve the least score because of this, but the cubes could also be rotated as well for a multiplier.

8. Keeping the game easy to run

This would be especially useful as anyone should be able to run the game, despite their system components, as it aims to improve someone's aim rather than the game sense which would only be attainable in that particular game which will definitely have higher system requirements. This means that my game will have more functionality, as it will be able to be played on a worser machine, school as library or school computers that will help them improve when they can't run the game they want to play. Seen in requirement 9.

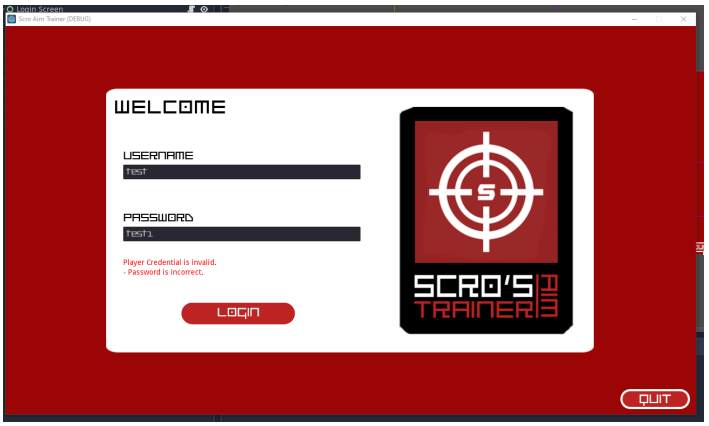
I have successfully met this criteria because even though I decided to create it within a 3D game space, godot is very space efficient with unloading each scene when it is not in use, so it does not use any more memory than what it is needed at that moment. Additionally, I made the game scene compact in order for the user to focus on the aim of the game, rather than looking at the pretty graphics. This is evident in test 3,4, and 5 in post development testing.

9. Usability and customisation

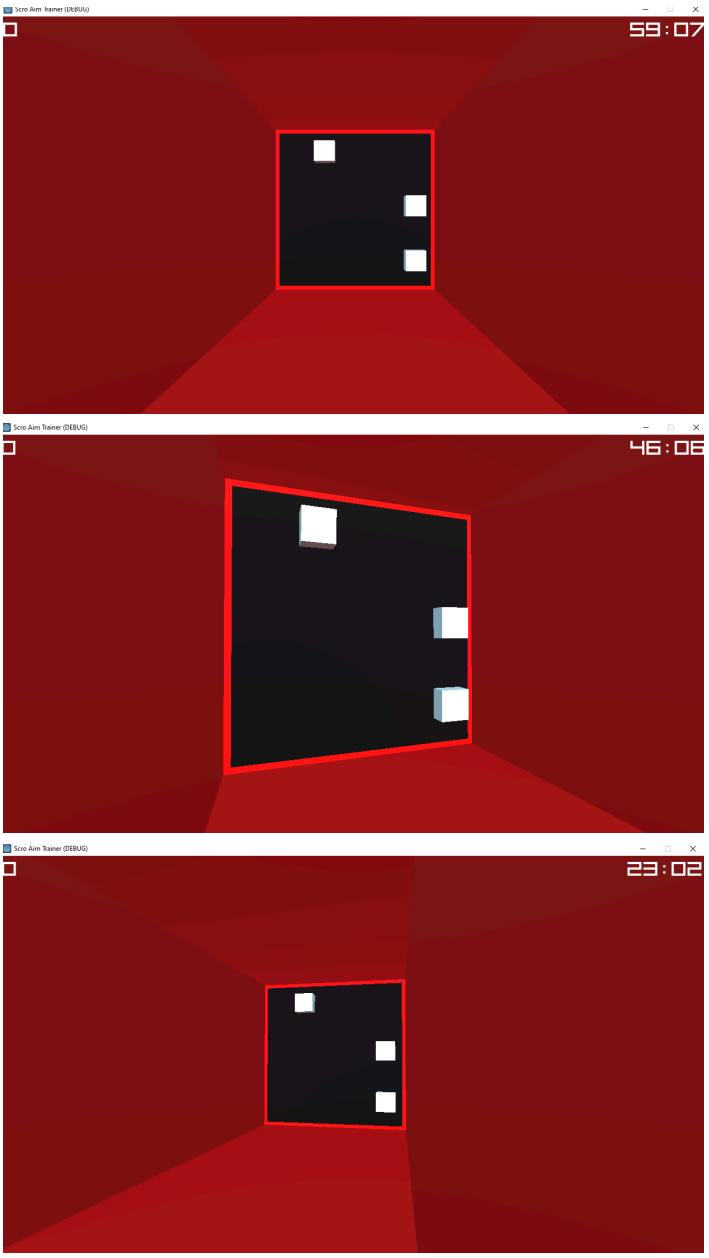
This needs to be easy to navigate for the user as mentioned in requirement 10, everyone needs to have an easy experience when going through the program. This would also help the user to know where everything is for future use, and prevent them from clicking on something that they didn't want to do. Their settings also need to be easy to customise and useful for the user as mentioned in requirement 8, everyone has different crosshairs that they like to play with and having more than one style would give each user a preference and fairness.

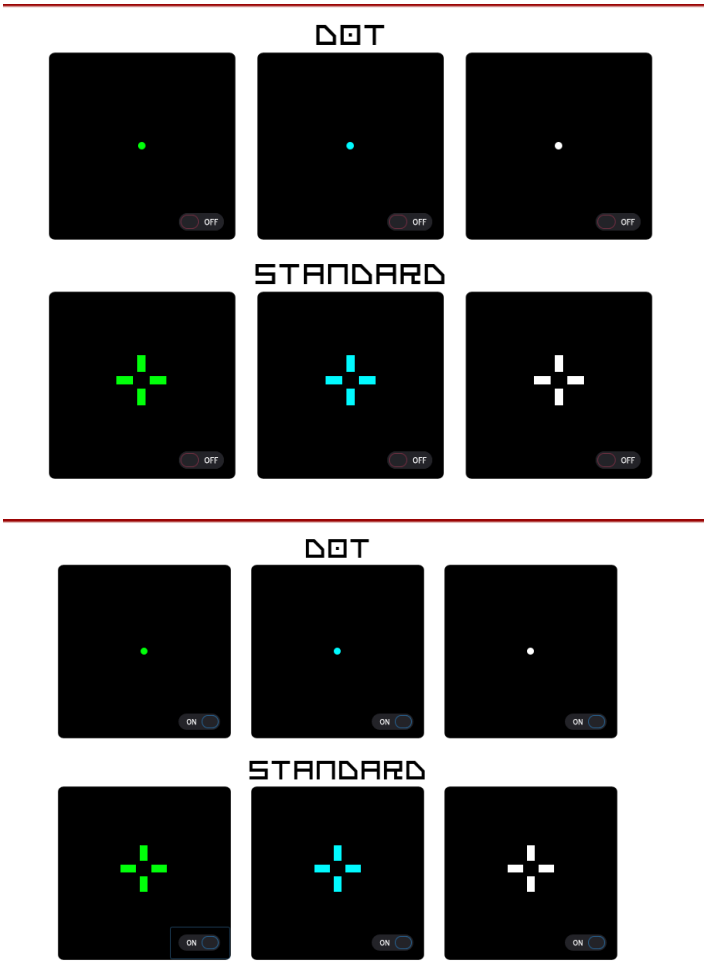
I have successfully met this criteria, as I have designed the UI with the intention of keeping it user-friendly from adding a consistent layout for the buttons, making the text nice and big, error message is in red to represent "problem" to name a few. Furthermore, the settings are fully customisable from the user, and it saves the mode of the button, so it can be loaded to its correct mode every time the scene is loaded. This is evident in test 3,4, and 5 in post development testing.

Usability

ID (T)	Description	Test Evidence
1	Test to see if the error messages on the login screen are easy to read.	  
	Explanation	
	In this login screen, I think the usability is very precise and clear. Whenever the user has made a mistake, a message pops up on the screen which shows what happened and the exact mistake the user has made in red below it. The text is an average size as well so this is not a problem.	

	Fix?	N/A
2	Test to see if all the buttons are easy to use and read.	
	Explanation	
	With the buttons on the main menu, they are large and clear, with the text font that is consistently used throughout the program. They are the same size to represent that each of them can be clicked on at any time (no order), and the text colour is complementary to the background of the button to make the text easy to read. With the quit and back buttons, they are smaller and only consist of red and white as these buttons have different purposes to the main menu buttons. Additionally, these buttons are always in the bottom right hand of the screen to remain a consistent layout, so the user knows there is a button whenever they change to a new scene.	

	Fix?	N/A
3	Test to see the functionality within the actual game mode.	
	Explanation	
	This is the best representation of the usability within this program. The features of movement and changing camera POV allows the user to personally move to a location that they feel more comfortable aiming at, whether closer, further, or to the left etc. Additionally, the black screen in the back is complementary to the 3D cubes, so the user will be at their optimum skill output whenever they play a round. In addition to this, the	

	score and timer is out of the way of the middle where the user is focussing, so this will not distract them while they are playing.	
	Fix?	N/A
4	Test to see the buttons and arrangement of my settings scene.	
	Explanation	
	In this scene, I am very proud of the usability. They are arranged in the style of a grid and this relates back to my game mode, which is also called “Gridshot”, thus this layout is suitable for this program. Additionally, the off mode is coloured red and the on mode is coloured blue, so even if the user cannot read the text clearly, they will be able to see the mode using the colour.	
	Fix?	N/A

My first feature of usability that is accessible to the user would be the error messages that will occur if the user had put login credentials that the database could not recognise. This allowed the user to know the exact problem in which they were wrong, and the red text was different from the rest to represent the importance. On the other hand, I could have improved the usability if I were to add an extra feature to give the user a way to fix their problem of either forgetting or remembering their details incorrectly. (Evident in Test 1)

An additional feature that I have implemented for usability is that if a button's style is the same to another one, it will have the same purpose. This is evident within Test 2, where each of the main menu buttons in the centre have the same style, so it means that they don't have a priority over the others. The purpose in doing this, is to allow the user choose what they want to click without making them do what I want them to. Another case would be the buttons in the bottom right corner, where their purpose is to assist the user in navigating through the program in one flawless playthrough. What I would want to improve in them would be to either have more buttons around the edge of the main menu or make the centre buttons bigger so the user can click on them easier for navigation.

In Test 3, I have added many features for the aid of improving my usability that help the user in customising their position within the 3D space, so it gives the user the option to freely change their strategy while they are playing each round, trying to find out what spot in the box they get a higher average score in. I have made the text small for the reason of not redirecting their attention, as this would make them less focussed on the aim of the game. However, the only thing that I want to improve/add would be that the user should be able to crouch, as this is present in most newer games. This would shift the POV of the user down in the y-direction, which might change the outcome of their skill level that cannot be tested as of now.

My final evidence of my good usability would be with my settings scene. The layout is placed in a style which is relevant to my game with it being displayed within a grid, and the buttons shown (in Test 4) are easy to understand from the colours changing off and on. But, in this case, this would be the scene to work on the most out of all the ones I have shown, as I think the user should have the ability to create their own unique crosshair, alongside these preset ones. This is because the crosshair is universally different from user to user, and many hardcore FPS players would create their own as they have customised it to their own strengths.

Maintenance

For the future plans of this project, I have included many examples of thinking about maintenance within this entire documentation.

My first example would be my modular-approach to all of my code, as I have included many functions and I have the ability to add more code into each section without breaking any of the pre-existing code.This is evident in the entire Development section, where there are clear script names that dictate what future code goes into what function of that script.

Additionally, I have tried to use global variables the least, so then the individual scenes can run at its maximum RAM capacity on any computer the user has.

My second example would be my naming conventions and detailed comments, evident in my Naming Conventions and Final Project Code. This would give any other programmer that works on this project the required information to know what is happening with every line of code and why it is essential Furthermore, since I have said what each script/section of my code does, it can be reusable in the future if I or the other programmer requires the same code.

My third example is the lack of maintainability for the new users. Since the sign-up feature has not been added due to lack of time, the developer would need to add a new profile each time a new individual user wants access. This would prevent the long-term sustainability of the program, but this could simply be developed by rearranging the data storage and making it retrieve data more efficiently. This is explained further in the limitations section.

My last example would be the documentation of this project. This is especially important for any new programmer that might want to continue this project because it provides the motive behind the creation of the program and where I intend for the project to go in the future. This would help them know what other features I would've liked to add, and if their ideas were similar to mine. This would be mostly evident in the analysis and the reviews I have done.

Limitations

While I did include a modular-approach to the code in order to reduce the development time, it still did not help provide time to complete all of the additional features that I intended to add beforehand.

The most important example of this would be the inability to complete all 3 game modes that included a complex algorithm. When I was documenting all of my code, it took a lot more time to figure out what the problems and potential solutions were, as I was completing this project in a new game engine, for which I have never used before. This required me to learn all of the basics and features of the game engine first, before actually being able to code which cost me even more time. However, if I had to do this project again, I would have still done it within godot because of the learning experience and how more developer-friendly the environment was to code in. Besides, the game only having one game mode actually helped create a focus for the user, as this game mode trains the user on all of their basic skills that need to be high before training on the other game modes.

Another example would be the inefficient use of databases. This is something that I discovered while commenting on my code afterwards, where I could have made the score database within a table in the player credentials database. Other than storing each profile as a table within the second database, I can store it as a record in the main login credentials database instead. Furthermore, this would make it easier to add and remove a user's record as all of their data would be located within one database. This would have improved the efficiency of the code and reduced the time complexity by the program only needing to access one database for all the data. Below is what I have adjusted in the main database to

visualise how the data would be written into the database by using the SQlite library commands.

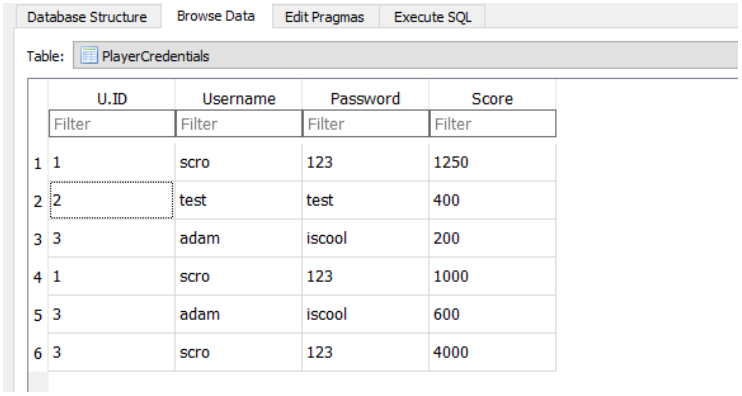

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: PlayerCredentials

	U.ID	Username	Password	Score
	Filter	Filter	Filter	Filter
1	1	scro	123	1250
2	2	test	test	400
3	3	adam	iscool	200
4	1	scro	123	1000
5	3	adam	iscool	600
6	3	scro	123	4000

An additional example would be the missing sign up feature that I thought I would have time for at the end of my project. This is partially successful since the code does work as intended by writing a userID to the database. However, because the development took a lot longer than I expected it to take, it was not finished in the time that I had available.

```
#SIGN UP FUNC - DO AT END
func commitDataToDB(): # Function to call the database
    db = SQLite.new() # Creates a new command
    db.path = db_name # Links the SQLite database to godot for use
    db.open_db() # Opening the database
    var tableName = "PlayerCredentials" # Declares variable to the table in the pre-existing db
    var dict : Dictionary = Dictionary() # Declares variable for godot to call the fields of the db
    dict["Username"] = "Scro" # Username Example
    dict["Password"] = "ScroThePro" # Password Example

    db.insert_row(tableName,dict) # Inserts the new data into the corresponding fields

"""

"""
#Getting text from the user
func _on_Usernametype_text_changed(NameofUser):
    db.open_db()
    db.query("SELECT Username FROM " + tableName + " WHERE Username=" + NameofUser + ";")
    if db.query_result[0]["Username"] == true:
        print("True")
    else:
        print("False")
"""
```

Shown above is the code that I had written before I left my first phase, but the code is commented , so if I were given more time, I would be able to simply complete this first by implementing it into the login scene. In the end, I did not finish this but if the users that want to play the game already have a userID and they haven't forgotten their logins, then it won't be a problem.

My last example would be the leaderboard in which I failed to complete. This feature would recognise all of the users in the database and rank the 1st, 2nd, 3rd, 4th, and 5th users on their highest score achieved within the game. However, I was unable to complete this because the database wasn't built efficiently, so it would be very difficult to try and query the userID field in one database to the Score field in the other. If I were to do this project again, this is what I need to make sure I do correctly, as this prevents me from implementing the leaderboard into my game.

Bibliography

https://www.youtube.com/watch?v=ya5BaDzcwkk
https://github.com/0xspig/3dCharacterController
https://www.youtube.com/watch?v=kE9q9yfpr24
https://www.youtube.com/watch?v=v4IEPi1c0eE&t=180s
https://www.youtube.com/watch?v=jf_Hz0dil8Y
https://www.youtube.com/watch?v=jN_f9i_bAyU
https://www.youtube.com/watch?v=PqRilWo8hSE
https://www.youtube.com/watch?v=jK6a0RqmmOY
https://www.youtube.com/watch?v=gYNgW3Z9zhU
https://www.youtube.com/watch?v=aRrsHQIB6Kg&t=130s

I have also used the official godot discord to ask questions that I was having trouble with. I did not get the exact answer, but it made me identify the essential logic features such as loading the global variables and calculating the new updated values for the scores.

Appendix

Final Project Code
Interview