Cosmic Quest!

Computer Science Project

Henry Walker

Candidate Number: 5204

Centre Number: 16341

The Sandon School

Contents

Analysis

The Problem I Will Solve

The issue that my project is seeking to address is a common problem in schools. Most of the current revision methods used by students are often not engaging enough, competitive or rewarding, which can lead to disinterest in the subject and revision all together. This may result in students using suboptimal revision methods or not revising at all, which ultimately affects their academic performance.

Therefore, my project aims to create a revision method that addresses this problem by combining revision with competitiveness and rewards. By incorporating elements of competition and reward into the revision process, students will be more motivated to engage with revision and do more than they currently do while still having fun. This can have a positive impact on their academic performance and help them to achieve better results.

By developing a program based around revision that is interactive and fun, students can benefit from a more engaging and effective revision experience. This approach can help to combat the disinterest and lack of motivation that most traditional revision methods cause and encourage students to become more active and enthusiastic towards revision.

Overall, my project aims to provide a solution to the problem of disengagement and suboptimal revision methods among students by creating a fun, competitive and rewarding revision method.

Project Overview

To tackle the issue mentioned above, my plan is to create a desktop app that offers an engaging revision experience for physics students. The app will offer a competitive platform for students to revise GCSE Physics. The rocket-flying game is the key feature that will attract students and cause them to revise more effectively.

During the game, students will fly their rocket between different planets while answering a question at each planet. They will have to answer each question correctly before moving on to the next planet, which adds an element of competitiveness to the game. The questions in the game will be brief and straightforward, consisting of multiple-choice answers. This feature will enable students to review a broad range of topics within a short period of time.

In addition, the program will save the time it takes for students to complete each planet and answer the questions, which will be stored in a database on a server and linked to their account. The program will feature a public leader board, which will enable students to compare their performance with other students in their class. This element of competitiveness will keep the students engaged, and it can provide a fun and exciting way to revise physics. By gamifying the revision process, students will be motivated to study more and achieve better results.

Third parties and end users

Although my project is to help lots of students, I will be specifically designing it for “my client”. This will be Mr Mismar who is a GCSE and A level physics teacher at my school. I have chosen him as my client as I was originally planning on the program being some sort of space simulation and he got his master’s degree in astrophysics.

As well as Mr Mismar being my client, the project will be specifically to help students. Therefore, I will have a second client who is a friend of mine studying Computer Science with Physics at St Andrews University who has agreed to help with the project but asks to remain anonymous. By having him as a second client, I can make sure the project meets the requirements from both end users of the program – teachers and students.

Throughout the project I will mainly be discussing the program with my second client as they

Why this project?

The Original Idea

Originally, I wanted to make a program that can be used in physics lessons to help teach certain topics. Since I am studying A-level physics, I asked Mr. Mismar, a teacher who teaches physics to students studying GCSE and A-level, if there are any topics that a simulation would be very helpful for or if there are any topics that students find difficult to understand.

Figure 1 - Email from Mr Mismar

Morning Henry,

I hope my suggestion does not cause you hours of work so I will start with the caveat that the following is just a suggestion as I am unsure how such a simulation would be coded.

A simulation I have always wanted to see and demonstrate would be the life cycle of a star.

Some simulation where you could have a user interface to begin with a high mass or low mass star and then show its various transitions to becoming a red giant or a red super giant and eventually a black hole. Approximate a sphere with outward forces and inward gravitational forces and how one force exceeds the other and the effect on size and temperature etc.

Again, this might be a mine field of problems to code.

What do you think?

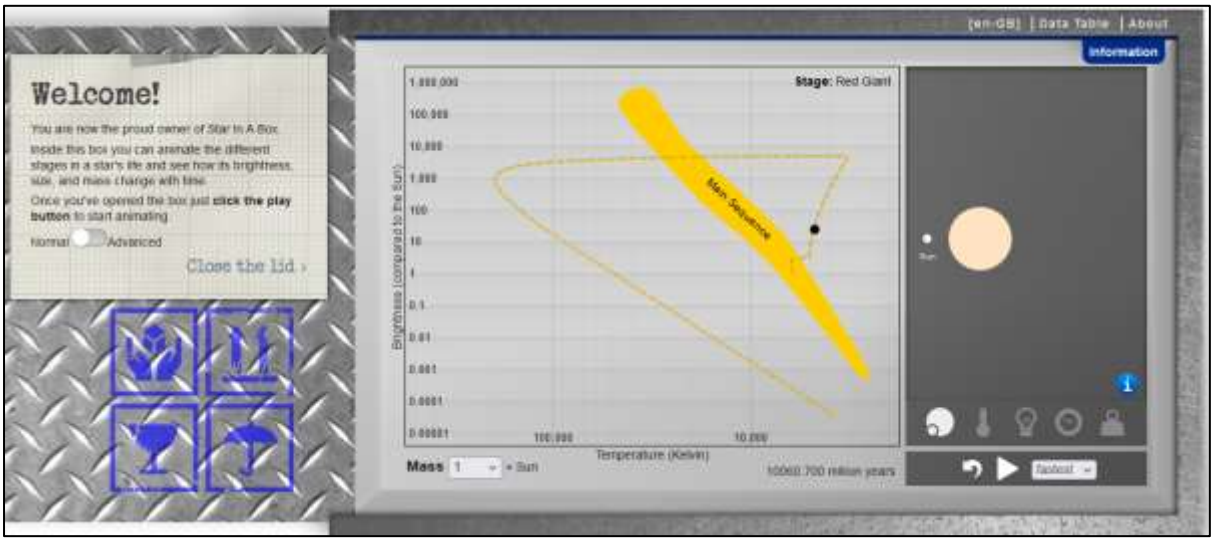
Regards

Mr Mismar

He said he has always wanted a simulation of the life cycle of a star to help explain how stars evolve but that he has never been able to find one online. I liked this idea a lot as I am fascinated by how the solar system and universe was created. My initial thoughts were to simulate many particles and show how they clump together to form a star using a graphical interface.

Not quite knowing if this would be the best approach, I decided to research how other star simulators simulate stars and I found “Star in a Box”[1].

Figure 2 - Star in a Box online star simulator



This simulator focuses on showing you how the temperature and brightness of a star changes over time, this differs from what I was thinking as it focuses more on the mathematics instead of the graphical simulation.

After more thought, I decided that for the simulation to be realistic I would have to simulate each particle using formulae I would learn from taking A-level physics. Researching into this more, I found a Wikipedia article about “N-body Simulations”[2].

Figure 3 - Wikipedia article about simulating particles

Nature of the particles [edit]

The 'particles' treated by the simulation may or may not correspond to physical objects which are particulate in nature. For example, an N-body simulation of a star cluster might have a particle per star, so each particle has some physical significance. On the other hand, a simulation of a gas cloud cannot afford to have a particle for each atom or molecule of gas as this would require on the order of 10^{23} particles for each mole of material (see Avogadro constant), so a single 'particle' would represent some much larger quantity of gas (often implemented using Smoothed Particle Hydrodynamics). This quantity need not have any physical significance, but must be chosen as a compromise between accuracy and manageable computer requirements.

As seen above, it explains that it isn't possible to simulate every particle as there is simply too many and it wouldn't be feasible. It then goes on to explain that you'd have to simulate many particles as one which compromises accuracy. I wasn't happy with this as I was wanting my simulator to be as accurate as possible. From reading this article and finding out that the formulae needed are much more advanced than A-level physics, I determined that it would be too complex to create a realistic simulation that I'd be satisfied with.

Having concluded that simulating a star would be too challenging, I researched about simulating the solar system. I believed the solar system would be easier to simulate because, as I read in the Wikipedia article, a star cluster could be simulated by having a single particle represent each star. So similarly, I had the idea of simulating each planet in the solar system as a single particle. This would be much more feasible as it would be far fewer 'particles' to simulate than the star, and the formulae needed to simulate gravity between 2 bodies of mass are covered in A-level physics. Wanting to get some ideas as to what to include in the simulation, I looked at some readily available simulators. The first I looked at was "Space Engine"[3] which describes itself as:

> "A realistic virtual Universe you can explore on your computer. You can travel from star to star, from galaxy to galaxy, landing on any planet, moon, or asteroid with the ability to explore its alien landscape."

Figure 4 - Space Engine, showing planets.

The simulator is very realistic and is scientifically accurate. You can move about and visit stars or planets. The main features I liked were:

- All the planets are scientiffically accurate allowing the user to understand true distance and size of the planets.
- The statistics for each star/planet and navigation menu is simple and efficient.

Although a program like this could be beneficial for helping teachers explain the solar system, it is unlikely that a student would have any need to use this program. Moreover, if they had a desire to use this program, I found that after a short amount of time, it lost its spark, and I didn't see any point in using it again.

After researching “Space Engine”, I realised that I enjoy playing games and more importantly, competitive games. This led me to the idea of creating a solar system simulation where the user can explore by flying a rocket around the solar system.

The Final Idea

Although the idea above is more interactive (much like a game), it’s not competitive, and I believe it’ll be the competitive aspect of my program that’ll make it enjoyable. Therefore, I expanded it by adding multiple choice questions which the user will have to answer as quickly as possible. These questions will appear at each planet, and once all planets have been answered correctly, their time will be stored. This time is what will make it competitive as it will be a combination of luck and skill. This is because the questions will be random, and the location of the planets will be different each time, but the user will still have to answer the questions correctly.

“Minecraft” is very different from my intended program, but I believe the competitiveness of it is very much like what I’m trying to replicate. Below is a table that shows how my idea links very closely to Minecraft and how it will help keep users engaged.

Minecraft	My Program
Minecraft generates its worlds using a random seed meaning the player will have no idea where they’ll be or where they’ll need to go.	My program randomises the location of the planets so the player will not know where they’ll be flying too next.
Items needed to beat the game are obtained randomly meaning there’s luck to beating the game.	The questions the user will get will be randomly selected meaning they could get an easy question or a hard one.
The time taken to beat the game is what is competitive**,** and people take many attempts to get a good time.	The time to answer all the questions is what will be competitive for my program, and if the user wants a better time, they’ll have to have another go, therefore revising more.
Although there’s luck to it, players learn ‘shortcuts’ and how to beat the game more efficiently.	If students want to get a quicker time, they will study more outside of the game, so that they can hopefully get a better time in the game.
There’s an official leader board showing the quickest times which gives something for the player to aim for.	My program will have a leader board built into it so that everyone will be able to access it and easily. There will be an incentive to get on the leader board if I only show the top 10 quickest times instead of everyone’s times.

Target Audience

As my game is going to be designed for students, the intended target audience will be GCSE physics students. I am not going to design my program for A-level students because not all the knowledge needed for A-level physics is easily testable through quick, multiple-choice questions. Below is a table which shows how many students took GCSE physics each year and as shown, the number of students taking physics is only increasing.

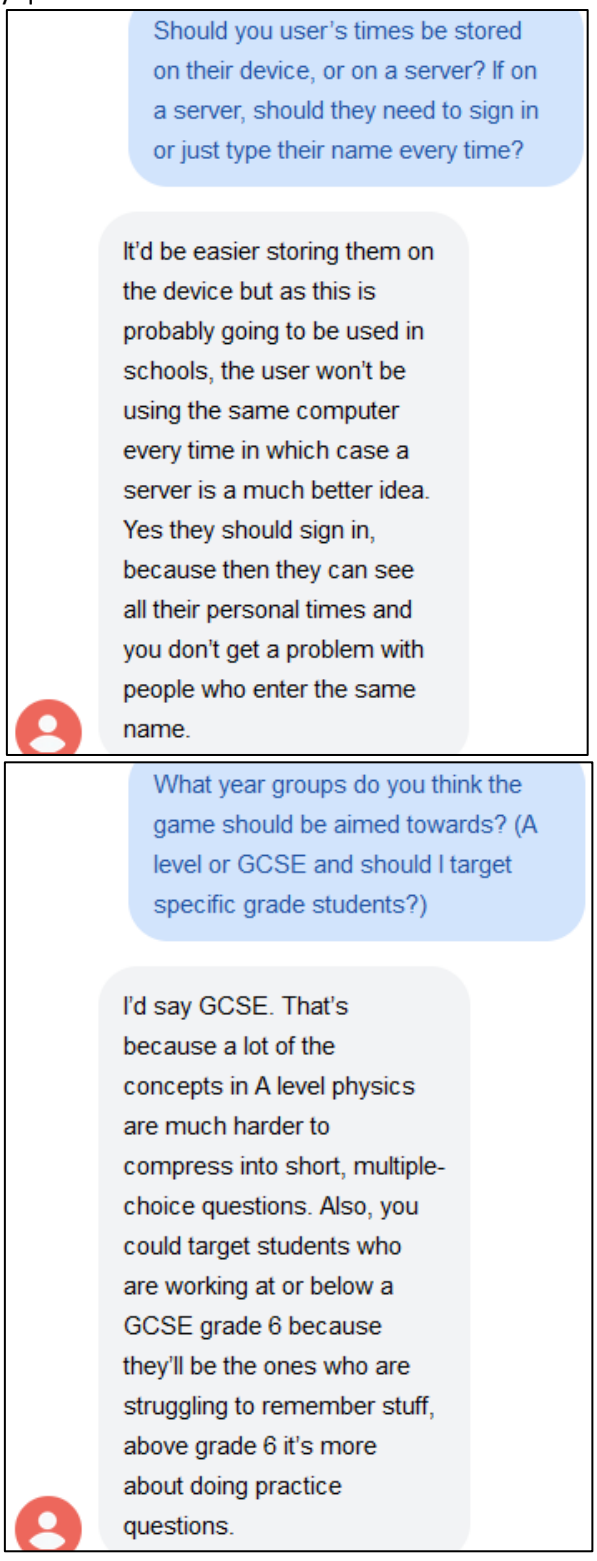
Year	2015	2016	2017	2018	2019
Num. of entries	124,986	130,830	132,159	155,994	157,819
Change from prev. year	-	▲5,844	▲1,329	▲23,835	▲1,825

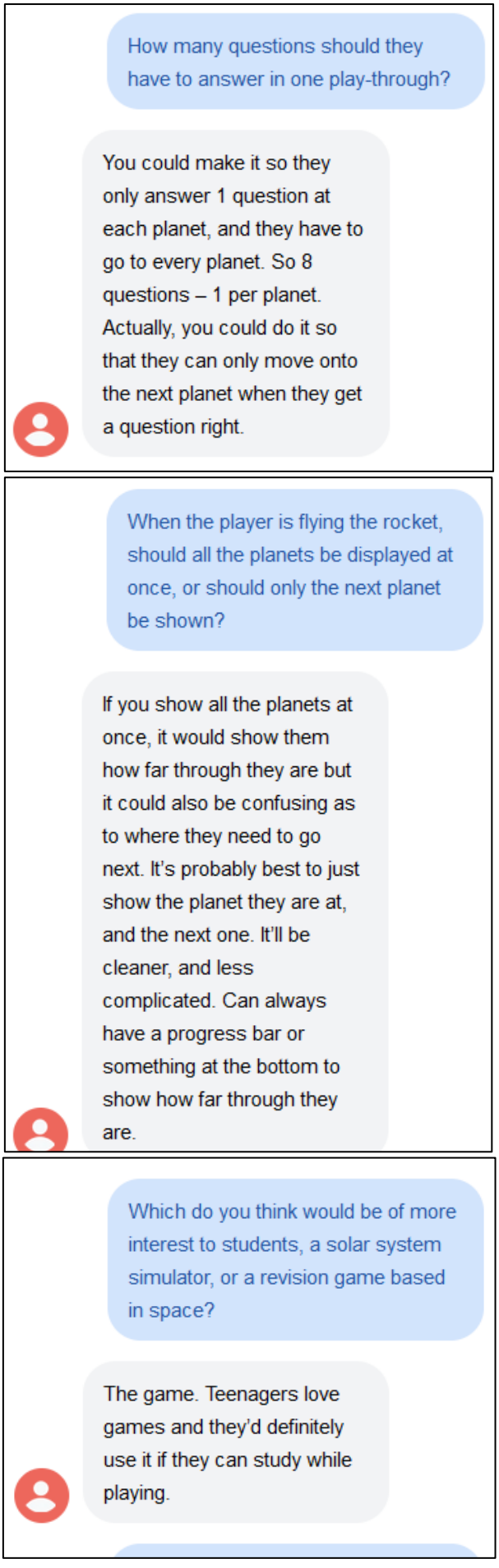
Although I will target GCSE students, I will specifically target those that are working at or below a GCSE grade 6. This is because to achieve above a grade 6, you need to know more than just the facts – you need to understand the concepts, and my program is going to be teaching the facts, not the concepts.

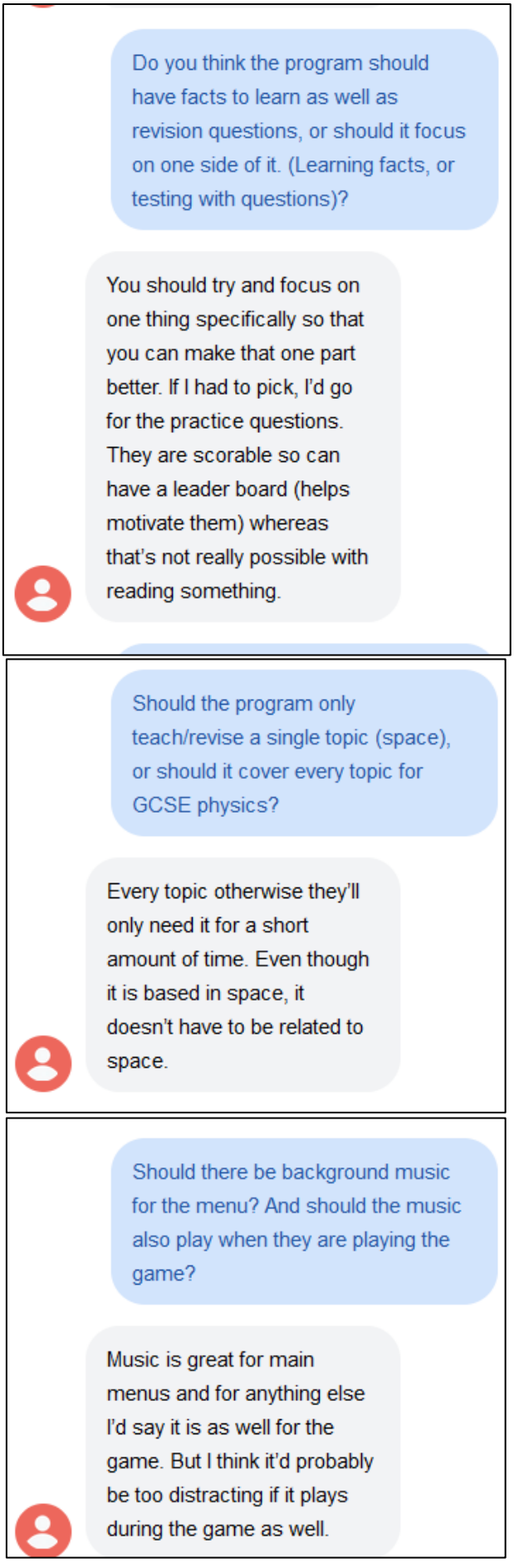
These students who the game will be targeted towards will be in year 10 or 11 (14- to 16-year-olds), of any gender and specifically studying AQA GCSE Physics.

Research to Derive Objectives

Although Mr Mismar was very helpful with the original idea, the program will mostly be used by students not teachers. Therefore, I decided to get some help from a friend of mine who is currently studying towards a BSc degree in Computer Science with Physics at University of St Andrews. I asked them several questions to help guide my design stage. The following are my questions and their answers:




How many questions should they have to answer in one play-through?
You could make it so they only answer 1 question at each planet, and they have to go to every planet. So 8 questions – 1 per planet. Actually, you could do it so that they can only move onto the next planet when they get a question right.
When the player is flying the rocket, should all the planets be displayed at once, or should only the next planet be shown?
If you show all the planets at once, it would show them how far through they are but it could also be confusing as to where they need to go next. It's probably best to just show the planet they are at, and the next one. It'll be cleaner, and less complicated. Can always have a progress bar or something at the bottom to show how far through they are.
Which do you think would be of more interest to students, a solar system simulator, or a revision game based in space?
The game. Teenagers love games and they'd definitely use it if they can study while playing.



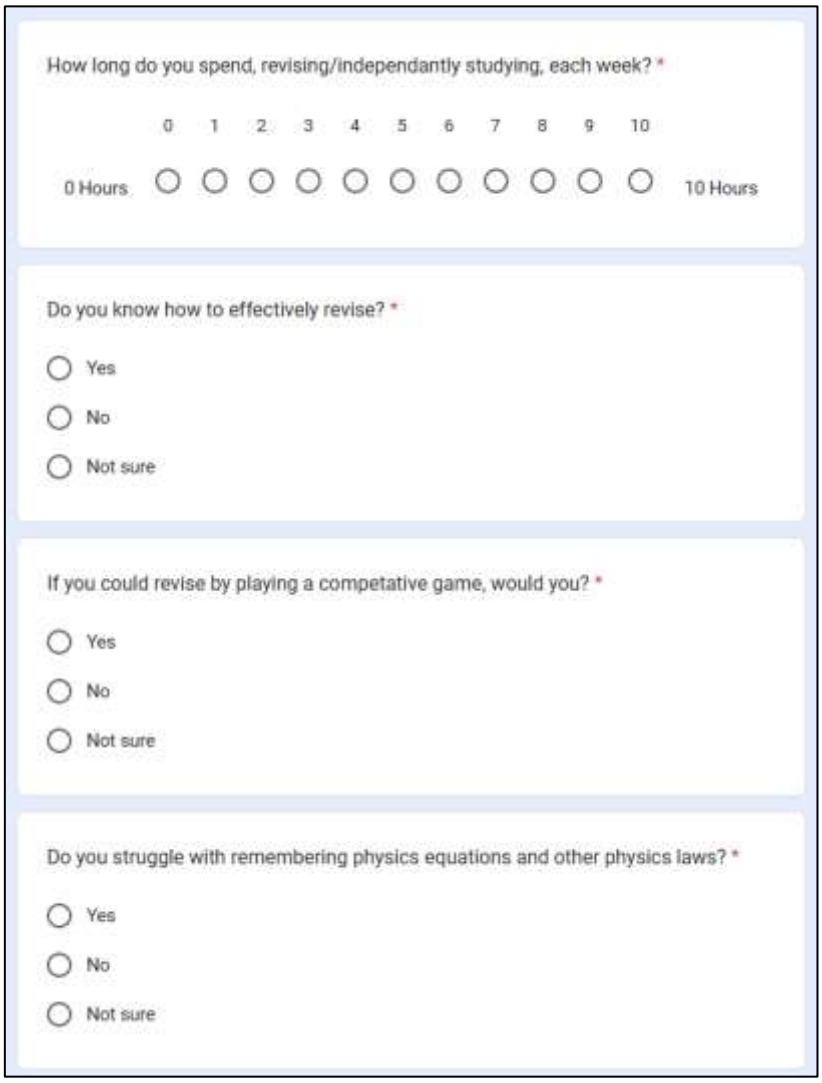
These answers helped to shape some of my objectives, those of which I have written below:

- The user will be able to answer multiple choice questions to help revise for exams.
- The program will show questions covering every topic within GCSE Physics.

- Background music will not play during the game as it will distract the user while they are trying to concentrate on the questions.
- The user will have to successfully answer a question at all 8 planets to complete a run-through.
- When the user is flying between planets, only 2 planets will be displayed – the current planet, and the next planet. This is so that the user does not become confused about where to fly too next.
- All users' time's will be stored on a server so that they can see their results from different devices.

Online Survey

To understand if there is a need for a physics revision game, I made an online survey[4] asking students the following questions:



The online survey was filled in by 170 students across different year groups. Below is a sample of the responses:

	A	B	C	D	E	F
1	Timestamp	What year group are you in?	How long do you spend, revising/independantly studying, each week?	Do you know how to effectively revise?	If you could revise by playing a competative game, would you?	Do you struggle with remembering physics equations and other physics laws?
2	29 March 2022	Year 12	1	No	Yes	Yes
3	29 March 2022	Year 12	3	Not sure	Yes	Yes
4	29 March 2022	Year 13	2	No	Yes	Not sure
5	29 March 2022	Year 11	1	No	Yes	Yes
6	29 March 2022	Year 12	5	Not sure	No	Yes
7	29 March 2022	Year 11	1	No	Not sure	Not sure
8	29 March 2022	Year 11	1	Not sure	Yes	Yes
9	29 March 2022	Year 13	2	Yes	Yes	No
10	29 March 2022	Year 12	7	Yes	Yes	Yes
11	29 March 2022	Year 11	2	No	Yes	Yes
12	29 March 2022	Year 11	10	Yes	Not sure	No
13	29 March 2022	Year 10	4	Not sure	Yes	Yes
14	29 March 2022	Year 11	0	No	Yes	Yes

Below is an analysis of the responses as well how this has influenced my project.

Analysis	Influence
Years 11 struggle most (compared to years 12 and 13) with revising and aren't sure how to effectively revise. For every 5 students only 2 know how to properly revise.	As many students struggle revising, a game could be beneficial to them as it could seem more like a leisure activity than an academic activity.
Across all the year groups studied, 71% of students would prefer to revise using a competitive game than traditional revision methods.	This has shown that the competitive aspect of the game is what will influence people to play it which is why this will be a large part of my focus.
Over 75% of responses from years 10 and 11 agree that they struggle to remember all the different physics equations.	As equations aren't given for GCSE exams, I will make sure to include all the equations as part of my program.
Years 10 and 11 spend on average, 3.7 hours revising per week whereas years 12 and 13 spend an average of 5.5 hours revising per week.	With years 10 and 11 revising for a shorter amount of time, the time they do spend revising needs to be more efficient, and I believe an enjoyable game will do so.

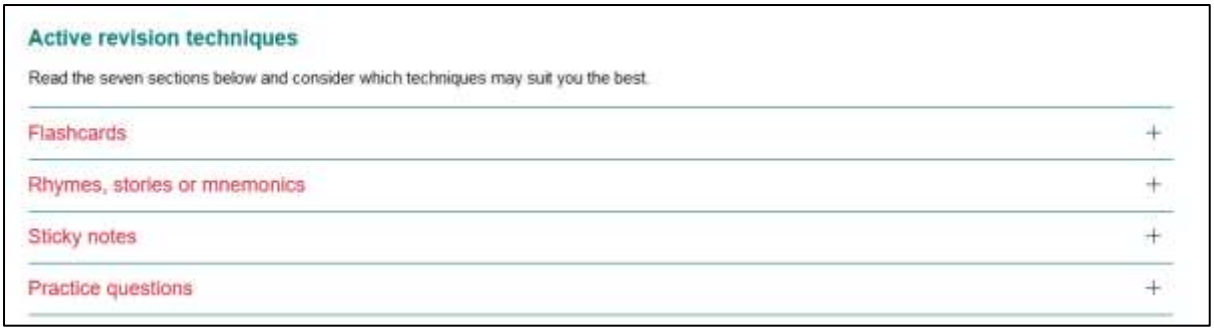
From analysing the results from my survey, I have come up with the following objectives:

- The program will be designed for GCSE Physics students, as I believe it will have the most benefit to them.
- There will be questions that cover all the GCSE Physics equations because from my own research I found that 75% of students struggle to remember them.

How Students Currently Revise

To understand how my program could be used for revision, I first looked at how students currently revise. Looking on the website of The University of Sussex[5], I found a list of active revision techniques (shown below) which are recommended to be better than passive techniques (such as reading notes).

Figure 5 - List of recommended revision techniques



Technique	Pros	Cons
Flashcards	Good for covering a wide range of content, quickly. They make the student recall something, helping to build the connections in their brain.	Doesn't teach exam technique or trickier concepts.
Rhymes, stories, or mnemonics	Helps to memorise specific facts (e.g., the order and names of the planets)	They take time to come up with and can be hard to remember if they aren't your own.
Sticky notes	Uses spaced repetition to teach small bits of information and they're quick to make.	Like flashcards but they don't have any recall so not as effective.
Practice questions	The best option for students who know the content, but struggle applying it.	Not good for students who struggle to remember the content.

How students will revise using my game

Although digital revision has become quite popular with examples such as Seneca or Quizlet, these are only digital representations of traditional revision methods. My program will differ from current digital platforms as it will be more like a game than a revision method.

With the common revision techniques, they require time and effort from the student (such as making flashcards), whereas with my program they won't need to create anything. Students will be able to play the game whenever they want, and they won't have to spend long playing either.

Scientific Paper on Spaced Learning

From reading the paper called "Spacing Learning Over Time"[6], I now understand in more detail how my game should work. The paper, written by Will Thalheimer, discusses that by spacing out your repetitions, you are likely to have better long-term retention of it than if the repetitions are not spaced. This is called 'Spaced Learning' and I shall include this by recommending the user to come back tomorrow. The paper states that:

> *Immediate repetitions are much less effective than repetitions given after short or long delays.*

the paper also discusses the different forms of learning stating that the repetitions do not need to be word for word the same, instead they can, and they should, be in other forms each time. Therefore, I will try and have each concept covered in multiple different questions.

Later in the paper it discusses that group learning is also beneficial to increasing long-term retention. Although this is another great way to learn, there are times when it isn't possible for students to work together, for example, if they are studying at home. My program is designed for these cases where group studying isn't possible or practical.

Second Interview with Mr Mismar

Now that I have a better understanding of what I want my game to achieve, I went back to Mr Mismar and spoke to him in person where I asked him what features he'd like to see in the game.

From our discussion, I've summarised his key features he'd like to see in the game below:

- There should be a leader board to show the top scores. Thus, making students compete to be the best.
- Although the game aspect of it will make it more engaging, the focus should still be on the questions. So, the game should be quite easy to play.
- The students will be using different devices each time they play as they won't always use the same computers in classrooms. So there should be some way that they can login and use the same username to save their results.

My Objectives

I have learnt a lot from my research about what I want my program to do. Below are all the objectives I will aim for my project to follow.

#	Objective	Why?	How?
1	The code will be closed source.	This is so that users cannot cheat or break the program by modifying the code.	The final program will be contained within an .exe file so that it can't be modified.
2	The program runs on the Windows operating system.	This is because Windows is the most common operating system within schools and is what my clients school uses.	I will test the program on various Windows devices to make sure it is fully functional.
3	The program properly runs on the most common sizes of screens.	People will likely have different monitor sizes as well as different aspect ratios.	The program will not use absolute positions, instead it will use relative positions based off of the size of the screen.
4	The questions will cover all the GCSE Physics content.	This is so that users will be able to revise everything for their GCSE Physics, not just a specific topic.	I will add questions to the database from every topic in GCSE Physics.
5	All questions should be multiple choice and require no calculations to get the answer.	This is so that the questions can be answered quickly, allowing the user to answer more questions in the same amount of time.	I will write the questions to have worded answers instead of requiring a calculation.
6	Nothing will need to be installed (even Python) to run the program (they'll only need the files).	This will make it easier for the program to be distributed to many computers as well as being easier to maintain as other required programs won't need updating.	The final program that will be distributed will be an EXE file which will run natively on Windows devices.
7	Users need to create an account or login before accessing the program.	This is so that when the user completes the game their scores can be stored and linked to their account, so their name shows up in the leader board.	The main program will only be shown once the user has created an account or logged into an existing account.
8	User accounts will be stored on a remote database.	This is so that the users can play the game using their account but don't need to use the same computer.	The program will connect to a server which will add the user account to the database.
9	All stored passwords are hashed before being stored.	This is an industry-standard which means if the database	When the user creates an account, their password will be hashed and

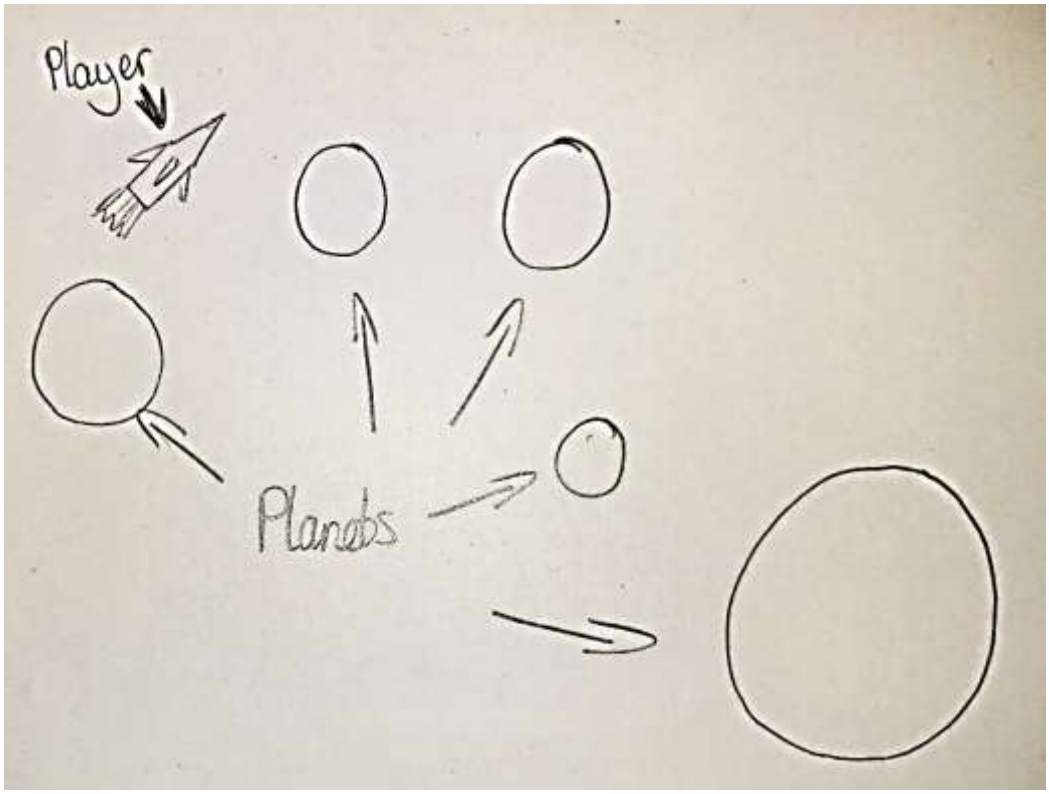
		gets compromised, the passwords stored within it are useless as they cannot be un-hashed.	then stored in the database. When they come to login, the password they enter will be hashed and then compared with the stored hash.
10	The program will have a main menu which will allow the user to choose what they want to do.	Instead of forcing the user to play the game, the main menu allows them to choose, creating a feeling of freedom and comfort for the user. This also allows the teacher to display the leader board without needing to complete the game.	Once the user has got past the login stage, the main menu will have buttons allowing the user to choose what they want to do.
11	Music will only play while the user is on the main menu, and not anywhere else in the program.	This is to reduce distractions while the user is playing the game. There is no music on the leader board so that the teacher can show the leader board without it producing lots of noise.	The music will stop once the user leaves the main menu and will start again when they return.
12	Before playing the game, clear instructions will explain how to play the game and what the aim is. This can be skipped if the user desires.	This is for people who haven't played the game before to find out how to play it and what they need to do.	The program will show instructions which the user can either read or skip.
13	The timer will only start once the user starts moving the rocket.	This allows the user to see what is happening and get ready before playing the game.	As soon as the program detects a key being pressed the timer will start.
14	The questions and answers will be stored in a remote database.	This is so that more questions can be added, existing questions can be removed or edited, and so that users cannot change the questions.	The program will connect to a server which will get random questions from the database before sending it back to the program.
15	Only 2 planets will be shown on the screen while playing the game.	This is so that the user knows where they are aiming for and isn't confused by having lots of planets on the screen. The main aim of the game is to revise, not to fly a rocket.	The planets will be in random positions and the rocket will start at one of them, so the user knows they need to fly to the other planet.
16	The user will have to correctly answer 8 questions for the timer to stop and the game to end.	This is to represent the 8 planets in the solar system. They need 8 correct answers as this makes it competitive as the user will want to get the questions correct to get a better time.	The user won't be able to move onto the next planet until they answer a question correctly.
17	If a question is answered incorrectly, the user is given another question and they get a time penalty.	This is to make the program more competitive as it incentivises getting the questions correct.	The program will get a large number of questions from the database and will show them a different question each time. When they get a question wrong their time will increase by a certain amount (It increases by 3 seconds).
18	When the user completes the game, their score will be stored within a remote database.	This is so that others will be able to see the user's score in the leader board and also so that the user doesn't need to use the same device to see their scores.	The program will connect to a server which will then be able to add the user's score to the database.

19	The program will have a leader board where users can see everyone's scores. The scores will be accessed from a remote database.	So that users can see how they compare to others and to make the game more competitive as users will want to get high scores.	The program will connect to a server using a network connection where it can then access the database and get the scores for the leader board.
20	A server will be used to receive requests, process them, and form a response for the client.	This is so that the user doesn't need to use the same device every time. It will allow them to access their account which stores their scores from any device.	The server will run on a specific device which will also have the database on it. The program will be able to connect to the server where it can then send requests and receive responses.
21	The way the user interacts with the program should be simple, minimal, and quick and the rocket should be controlled using W, A, S, and D.	This is so that users who don't use computers as much or play games can easily play the game without any prior experience.	Controls will be explained before the game and there will as few buttons as possible.
22	At least 50% of GCSE Physics students say they would use the game to revise.	This will be used to judge the success of the project and program.	I will run another online survey once the program is complete and ask for students' opinions on the game.

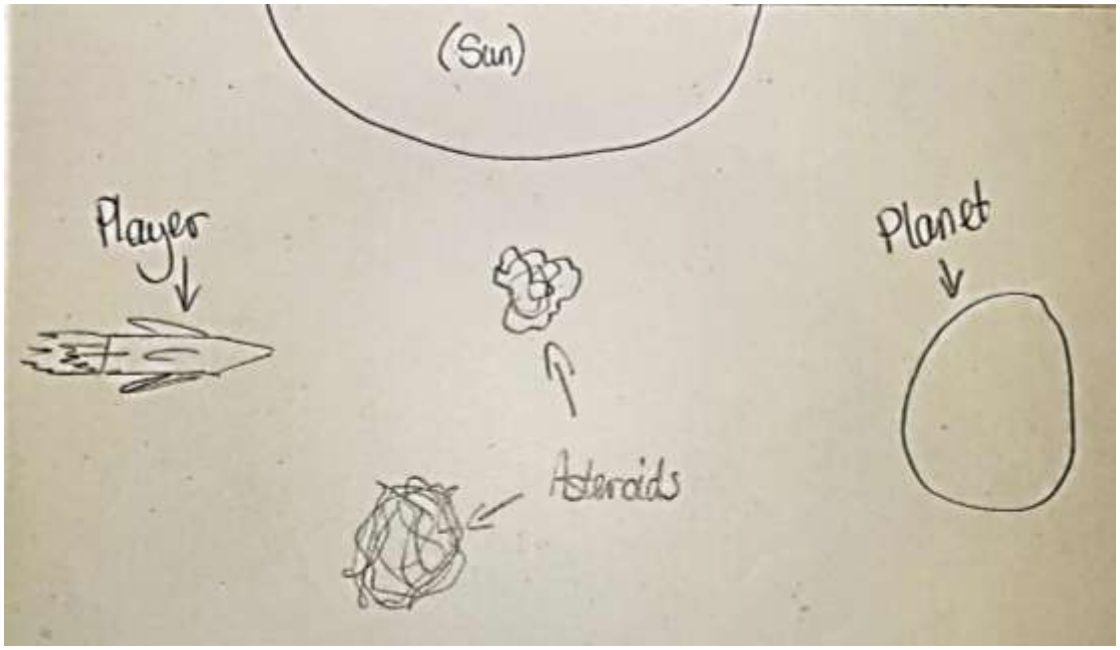
Initial Models

Initial Designs

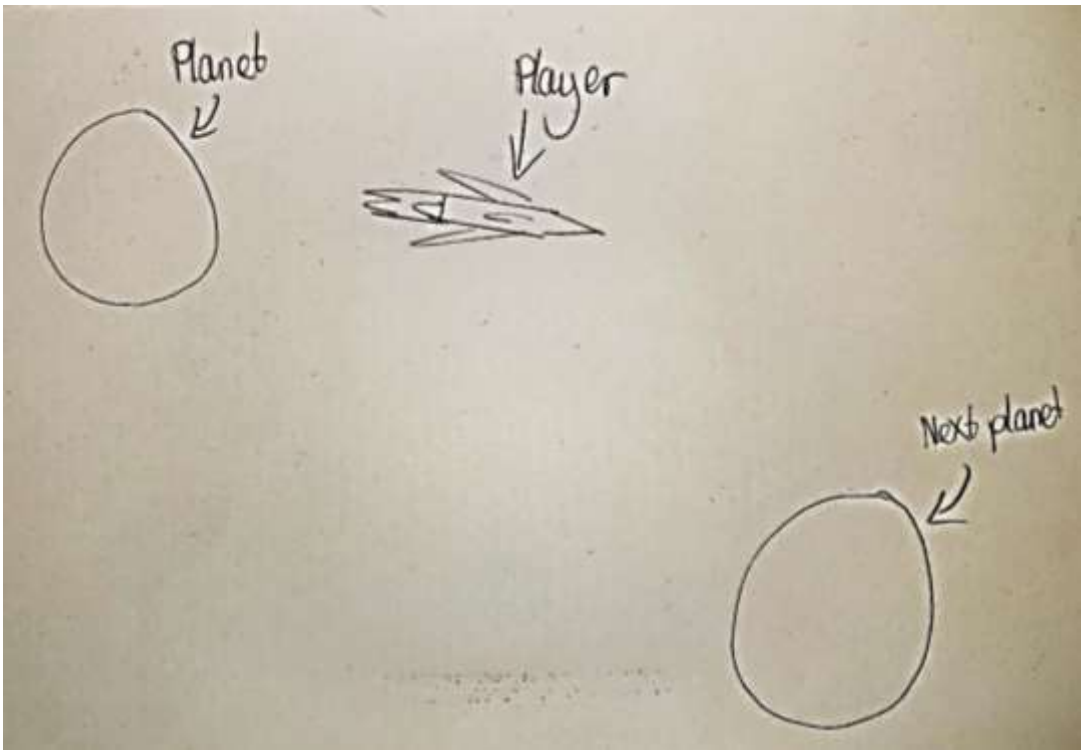
Now that I know what the project will roughly be like, I came up with some initial models as to what the program would look like.



The first design shows many planets on the screen and the user must answer a question at each planet. Their score is how many questions they get correct in total. To answer the questions the user must fly the rocket to each planet.



The second design shows only 1 planet, and the user must again fly the rocket to the planet to answer questions. However, this time, they must also avoid asteroids while flying. If they get hit by an asteroid, their score decreases.



The third design shows 2 planets and the rocket. The rocket starts at one of the planets and the user must fly the rocket to the second planet where they will then answer questions until they get a question correctly.

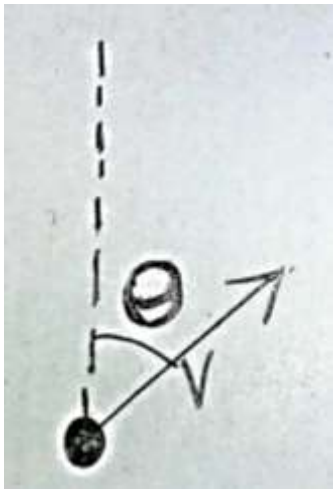
After creating these 3 models, I asked my client, Mr Mismar, which of these initial models he preferred. He said that out of all of them, the 3rd model seemed like it would be the best as it isn't too complicated, and the focus will still be on answering the questions instead of the focus being on playing the game.

Modelling Algorithms

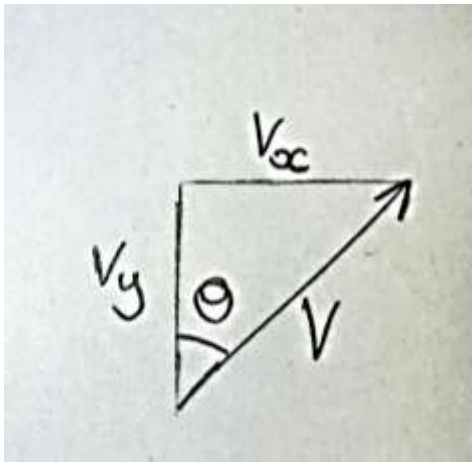
Calculating the rockets new position

Now that I had an initial model of the game, I needed to model the algorithms that I would use. For the game, the main algorithms will be mathematical based algorithms for calculating the positions and angles of the rocket.

To calculate the new position of the rocket, I will have to use trigonometry.



Using the above drawing, you can see that the rocket (the dot) will have a velocity, v, and an angle to the vertical axis which let's say is θ. To calculate the new position, we need to work out how much the x position and the y position need to change.



To do this, we will split the velocity into a triangle as shown above. So now using trigonometry we can calculate the change in x and the change in y. These are:

$$x = V * \cos(\theta)$$

$$y = V * \sin(\theta)$$

Now that we have the change in x and y, we just add these on to the current position of the rocket and we now have the new position of the rocket.

Modelling Data Structures

As well as algorithms, data structures will be an important part of the program. Within the program I will use each of these data structures:

- List
- Dictionary
- Queue
- Vector

Lists will be vital to my project as they will be used to store buttons and textboxes. This is because they don't need to be accessed in a certain order, they just need to be contained.

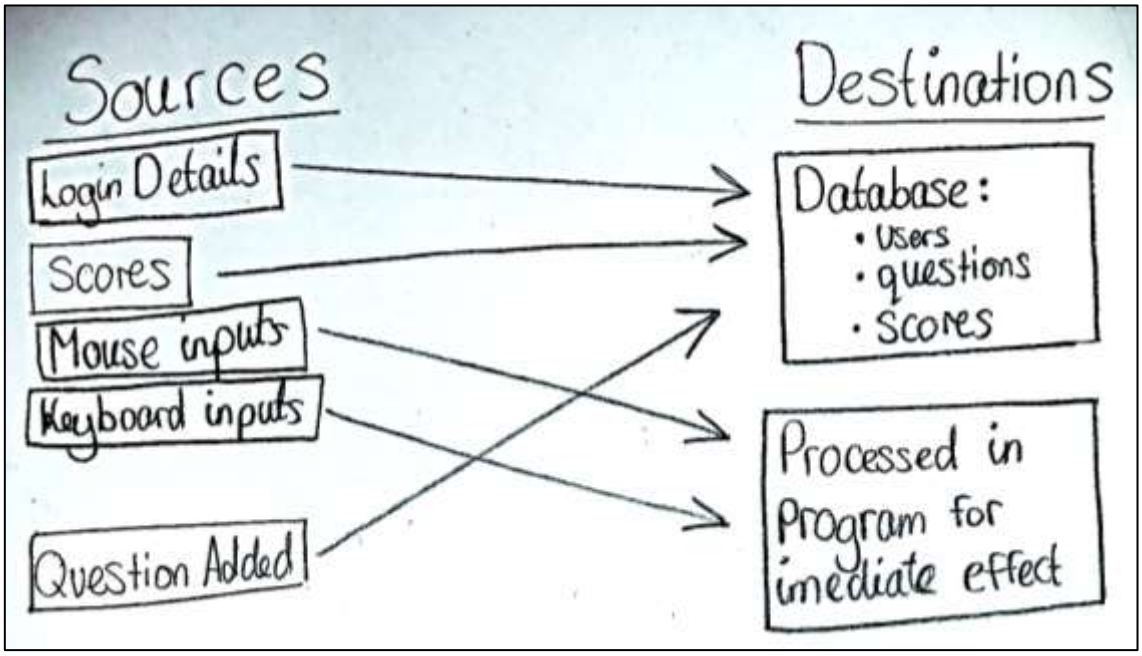
A dictionary will be used in my program, I will use it for storing the constants. This is so that they only need changing in one place and also so that they can be referenced using a specific string.

A queue will be used to store the questions as they will be in a specific order, and it will take the next question in the queue and display it to the user.

Vectors will be used within the program as well to store the positions of rocket and planets as well as storing the speed and direction of the rocket.

Data Sources and Destinations

Within the program there will be lots of data coming in, and lots of data going out. So that I account for everything in the design and technical stages, I needed to identify all the sources of data. Once I had worked out all the different sources of data, I then worked out where they would be stored. Some data would be stored for long-term use, this would be stored in a database. Other data such as keyboard and mouse inputs only need to be stored temporarily until they have been processed.

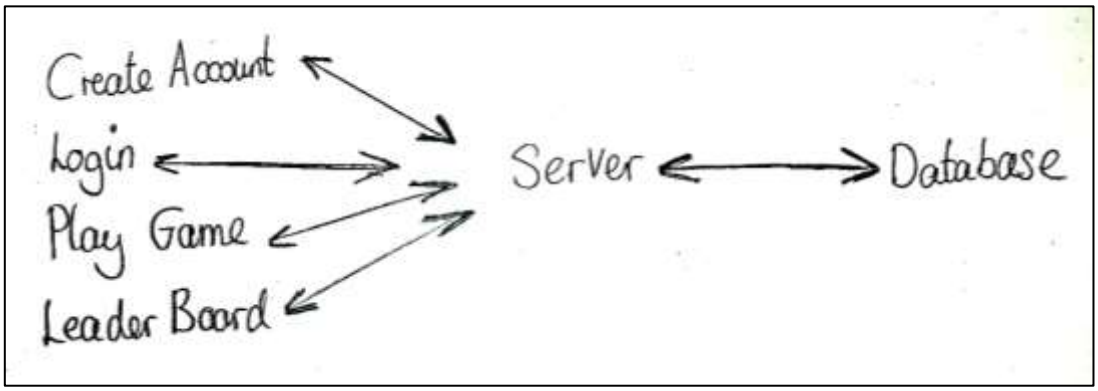


The diagram above identifies the different sources of data and then shows where they will stored/processed. As you can see, the user's login details and scores will be stored in a database as well as the questions they will be answering. This data will be stored long-term as it will need to be accessed once the program has been closed.

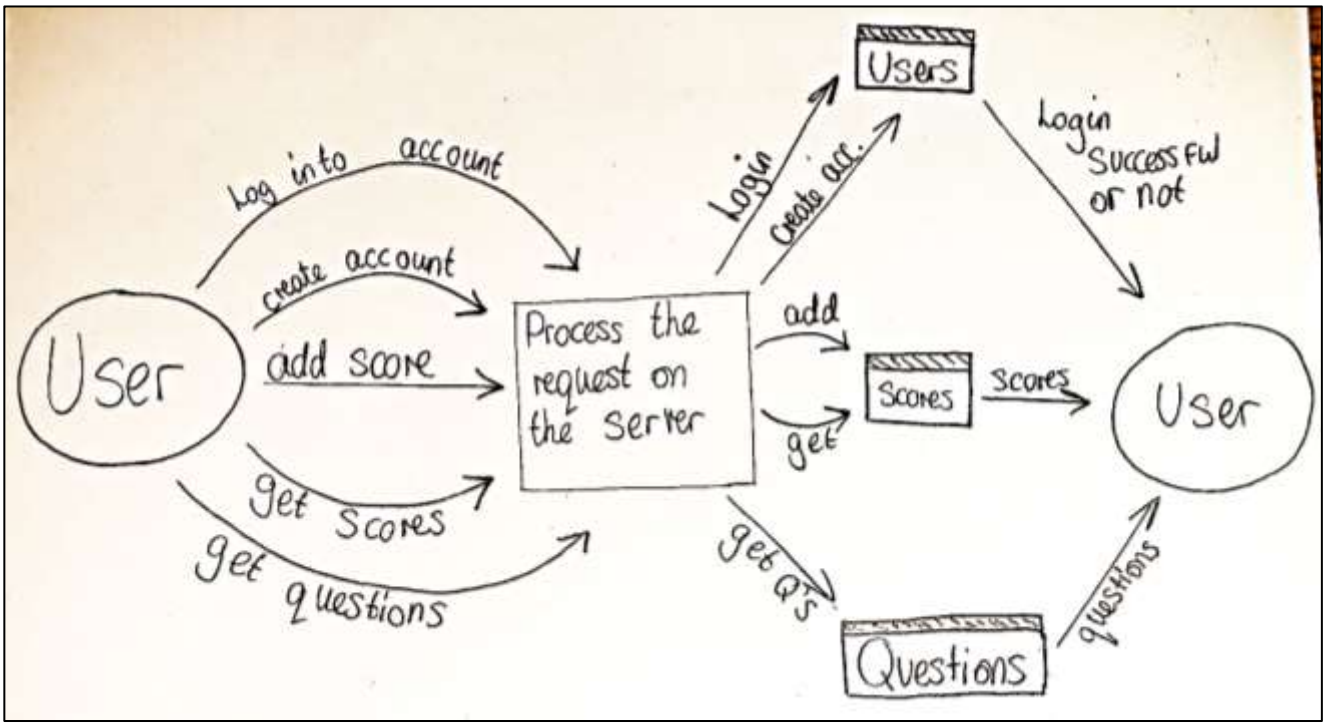
On the other hand, there will also be short-term data which will be mouse and keyboard inputs. These will be to interact with buttons or control the rocket and these will be processed immediately as they need to change something straight away. Once it has been processed, the data will be deleted as it will no longer be needed.

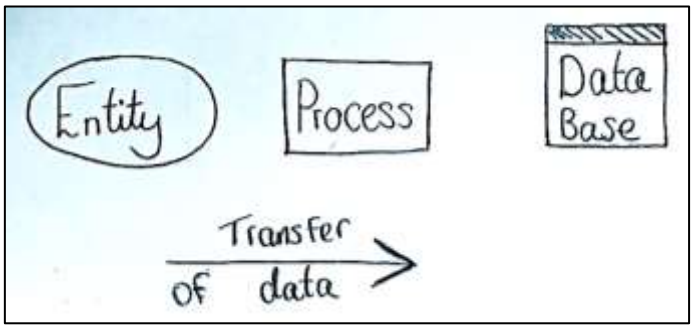
Data Flow Diagrams

Now that I have identified the sources and destinations of data within my program, it is important to model how this data will "flow" through the different parts of the program.



The diagram above shows the main data inputs and that they will all go through the server to connect to the database. The database will then be able to store / access the data which can then be returned back to the user. This diagram is useful however it is too simple as it doesn't show the specific data inputs or how they will be returned.

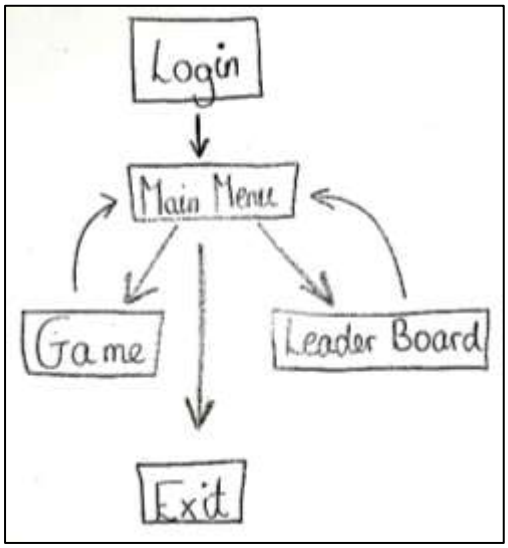




The above data flow diagram is a more detailed version of the diagram above this one. A key is also provided directly above to show what different shapes represent. This more detailed version is better as it shows the separate data sources as well as where they go more specifically and how they are returned. As you can see from the diagram, all data requests go to the server where they are then processed. When being processed the server will work out what it is requesting and then decide what it needs to do. The server will then either insert the data into a database or access the data from the database and then return it to the user.

Program Stages Diagram

Now that I have modelled how the data will "flow" through the program, I will also model how the user will "flow" through the program. When the user is using the program, there will be distinct different sections of the program and they will follow a certain order.



This order is visualised in the above diagram. As you can see, the user will always start with the login, and they will then go to the main menu. Here they will have the option between the game, the leader board, and exiting the game.

It is important to model this part of the program as it will help split up the programming of the project into smaller, more manageable sections.

The Chosen Solution

Having modelled and prototyped various different solutions to the problem as well as getting ideas from Mr Mismar and my second client, I have now chosen and decided on my final solution. To give you the most useful explanation of the chosen solution, I have explained every part of it as simply as possible in the below bullet pointed list.

- The user will first create an account or login. This account will be used to store their scores for the leader board. The program will connect to a server so it can create the account or check the login details.
- Once logged in, the user will choose between playing the game or seeing the leader board.
- When they choose to play the game:
    - They will first be given instructions on how to play game as well as what they need to do.
    - The program will then get a large number of questions from the server while the program loads the game.
    - They will then be shown 2 planets and a rocket on the screen. The rocket will start at one of the planets.
    - Once they start moving, a timer will start (counting up).
    - The user's aim is to get to the other planet.
    - Once at the other planet, they will be shown a multiple-choice question.
        - If they answer the question incorrectly, they will be shown another question and they will get 3 seconds added to their time.
        - If they get the question correct, they will be shown 2 new planets and need to fly to the other planet again.
    - Once the user answers 8 questions correctly, the timer will stop.
    - The program will then send the user's score (their time) to the server.
    - They will then have the options to play again, see the leader board, or exit the game.
- When they choose to see the leader board from the main menu or the end of the game:
    - The program will send a request to the server to get the top 10 quickest times.
    - It will then display the times in order showing their position, username, name, and time.
    - The user will be able to move up and down pages to see all the results and they will also be able to refresh the results.
    - The user will have the option to return to the main menu as well.

Design

Before I started making the program, it was important to create a plan that laid out all the details of how the program would work and what it would look like. This was really important because it made sure that the program would meet all the requirements and objectives. The design guided the development of the program and made sure that everything was on track.

IPSO Chart

The chart below details the inputs that the program will receive, the processes that will be performed on those inputs, the outputs that will be generated as a result, and the storage of any necessary data. Having an IPOS chart allows for a clear understanding of the program's intended functionality and helps identify any potential issues before the coding the program.

Inputs	**Processes**
Create Account **Login** **Mouse inputs** **Keyboard inputs**	**Create account.** **Check login details.** **Detect button presses.** **Detect user's answers to questions.** **Update rocket's position depending on keyboard inputs.**
Storage	**Outputs**
Database tables: - **Accounts** - **Questions** - **Answers** **Constants file**	**Program changing state.** **Visual changes on-screen.** **Rocket being moved on-screen.** **Questions appearing on-screen.** **Leader board being displayed.**

The user's primary mode of interaction with the program is through the use of the keyboard or mouse. Once they have successfully logged in, they will be able to mouse to navigate the menu and for the game they will use the familiar W, A, S, and D controls, which are commonly used in games. This design decision was made to provide a more intuitive and user-friendly experience for new users, as the controls will feel familiar. By using these controls, it will make the program more accessible and help users to quickly become familiar with the interface, making it more natural to use.

Much of the program will be based around the processing of the user's inputs. These processes include checking the user's login details, getting questions from the database, checking the user's answers, and processing the rocket when the user presses certain keys.

Data about the user as well as the questions and answers will all be stored within a database. The reason for using a database is because it allows for efficient and secure storage and retrieval of user information. Databases provide a way to organize and structure data in a way that can be easily queried and manipulated, for example making it simple to check if a user's login details match the entered details. The database will be located on a server which the user connects to when starting the program. This means the user does not have direct access to the database making it much more secure.

Modular Design

Although as a program it will function as one, the program will be designed in multiple sections. The sections the program will be split into are:

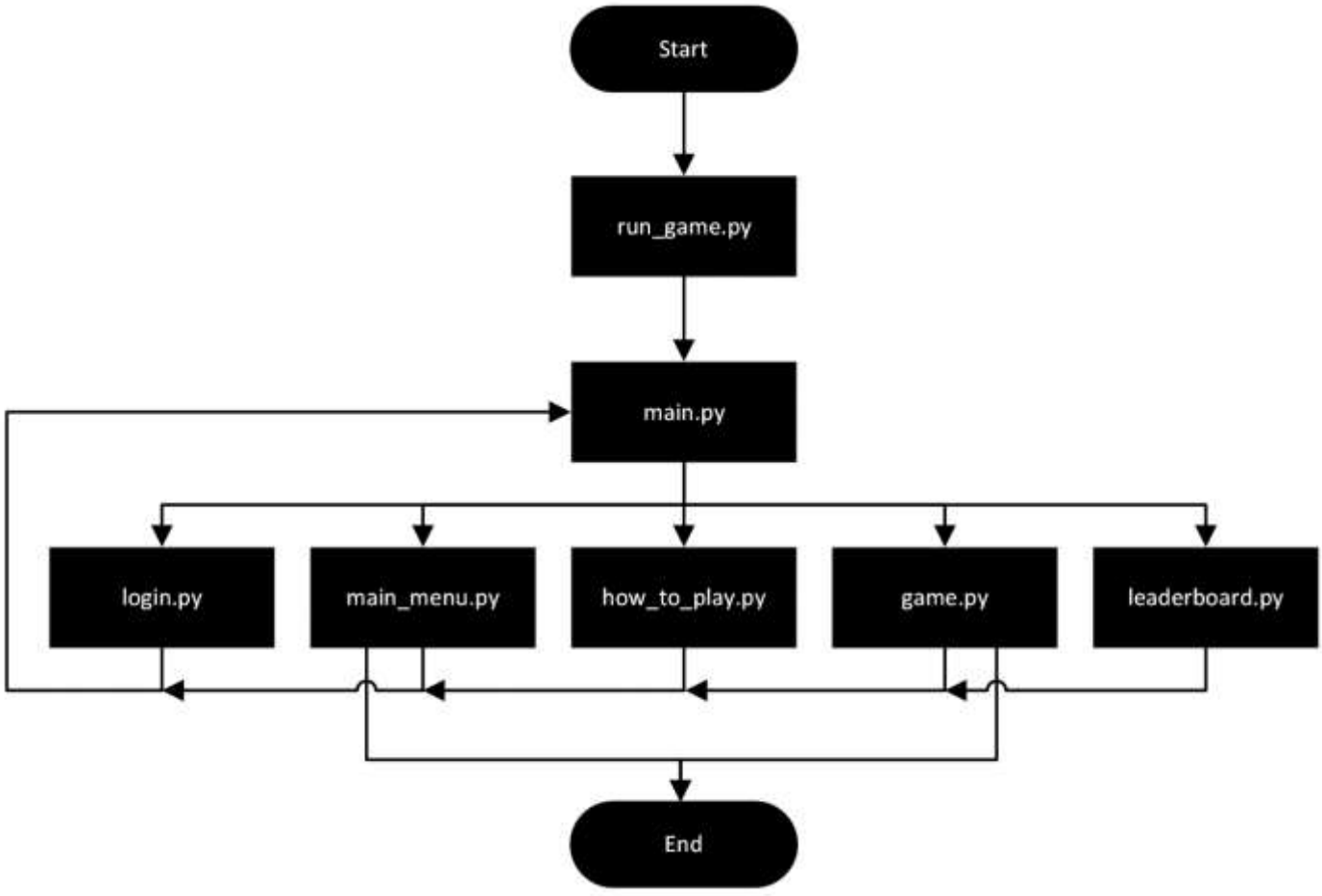
Name	Description
Main	This is what will manage the switching between sections.
Login	This will allow the user to create an account or login.
Main menu	This will let the user choose between playing the game or seeing the leader board.
How to play	This will teach the user how to play the game and what they will need to do.
Game	This is where the user will play the game and answer the questions.
Leader board	This is where the user can see the global leader board.
Server	This will run separately to the client's program and will manage requests from clients.

By splitting the program into sections, it means that when it comes to programming it, I will be able to work on a single section at a time before moving onto another section. This will hopefully allow me to be much more productive with my time as I won't be having to handle as much at once.

Program Control Flowchart

Due to the design of the program being modular where it's divided into multiple files, each with a specific function, I need to design how each section will connect and interact with each other. Each file needs to be able to communicate with every other file and also needs to be able to share control when needed.

To make sure that everything runs correctly, I created a flowchart to show how control will be passed between the files. This chart shows the different steps that the program will go through and how each file will be involved. By having this flowchart, it makes it easier to understand how the program works and where any issues might arise. It also helps to make sure that everything is working as intended and that the program is functioning correctly.



In the flowchart, you can see that the main.py file is responsible for controlling the overall functioning of the program. It makes sures that each file can be accessed at the correct time and can switch to other files when needed. When the program is started, main.py will take control and direct the user to creating an account or logging in before they then enter the program.

Once the user has completed a section of the program, the file that was being used will hand over control back to main.py. The file will also let main.py know which section the user wants to go to next. Main.py will then take this information and start running the desired file.

This process continues throughout the program, with main.py managing the different files and making sure that everything runs as intended.

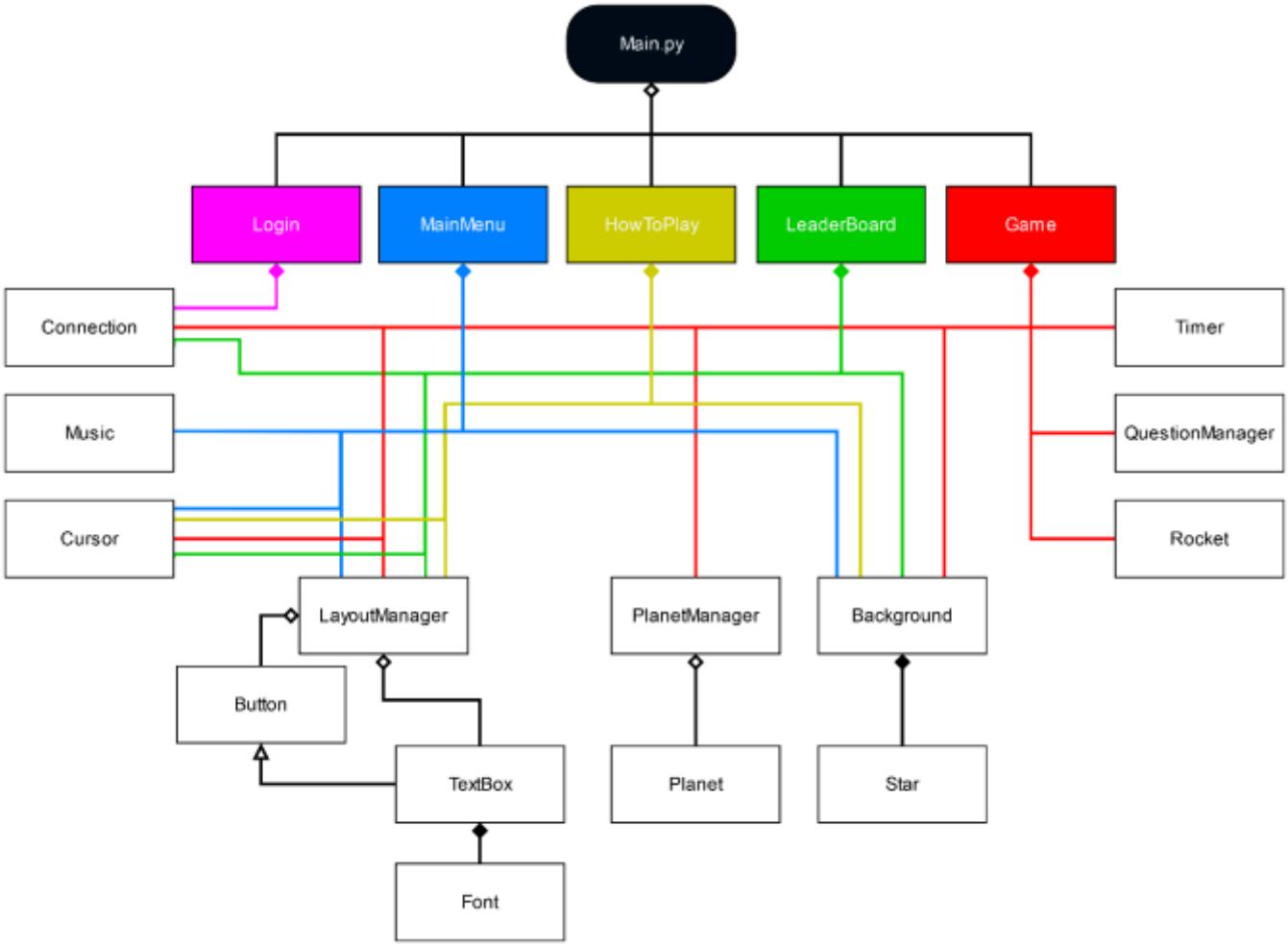
Inheritance Diagram

When creating the program, I wanted to make sure that it was easy to use and maintain. One way I did this was by designing the program with a structure that allowed for code to be reused. This means that I can use the same code in different parts of the program, which makes it easier to test and debug the program. It also ensures that the program has a consistent design throughout.

To help visualize this structure, I created the inheritance diagram below. This diagram shows how the different parts of the program are related to each other and where the different classes will be used. On the edges and bottom of the diagram, you can see the classes that I created. These classes are used repeatedly throughout the program, which helps to simplify the code and make it more organized.

Across the top of the diagram, you can see the main sections of the program. Each section is connected to the classes that is uses. The diagram also shows the different types of relationships that are used throughout the program, such as inheritance, aggregation, and composition.

By using this inheritance diagram and designing the program in this way, I was able to create a program that is easy to use, maintain and understand. It also means that if I need to make changes or add new features to the program, it will be easier to do so.



To access the above image in colour, then visit the website below or scan the QR code. On the website there are 3 testing videos as well as the above image in colour.

https://henryjwalker.github.io/

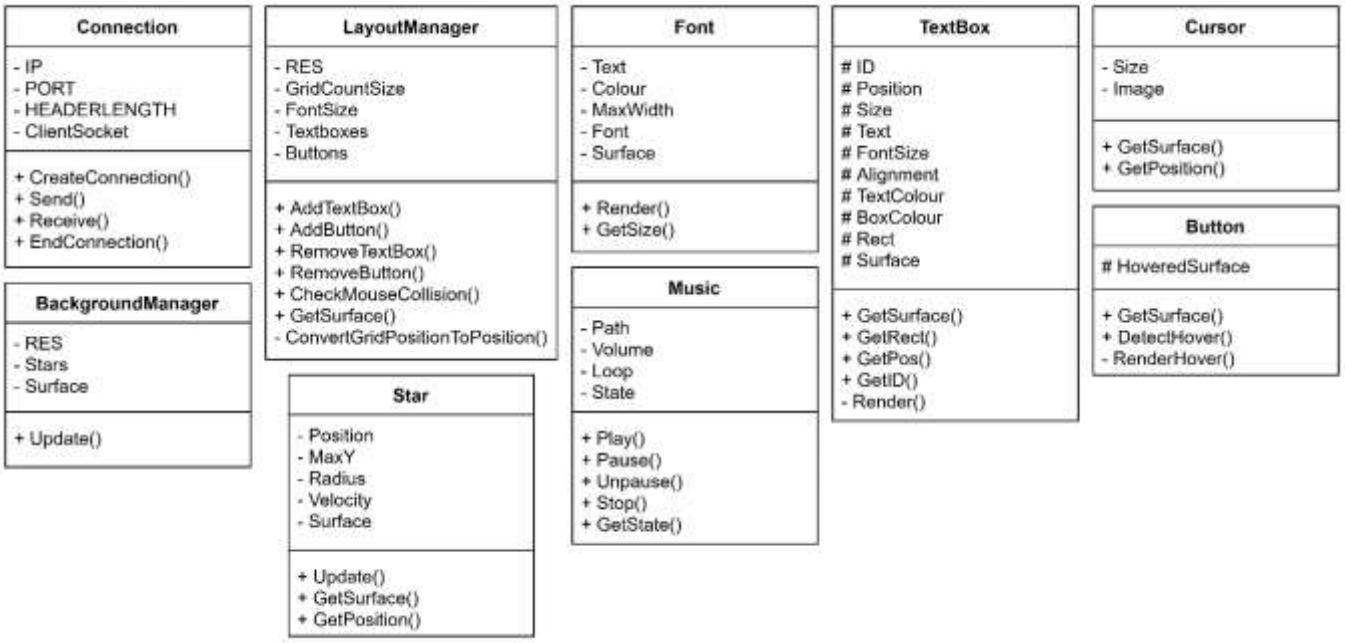
Class Diagrams

The inheritance diagram is important for visualizing the relationships between different classes in a program. It helps to understand how each class is related to others and how they interact with each other. However, the diagram has its limitations as it does not provide a complete picture of each class's methods and attributes.

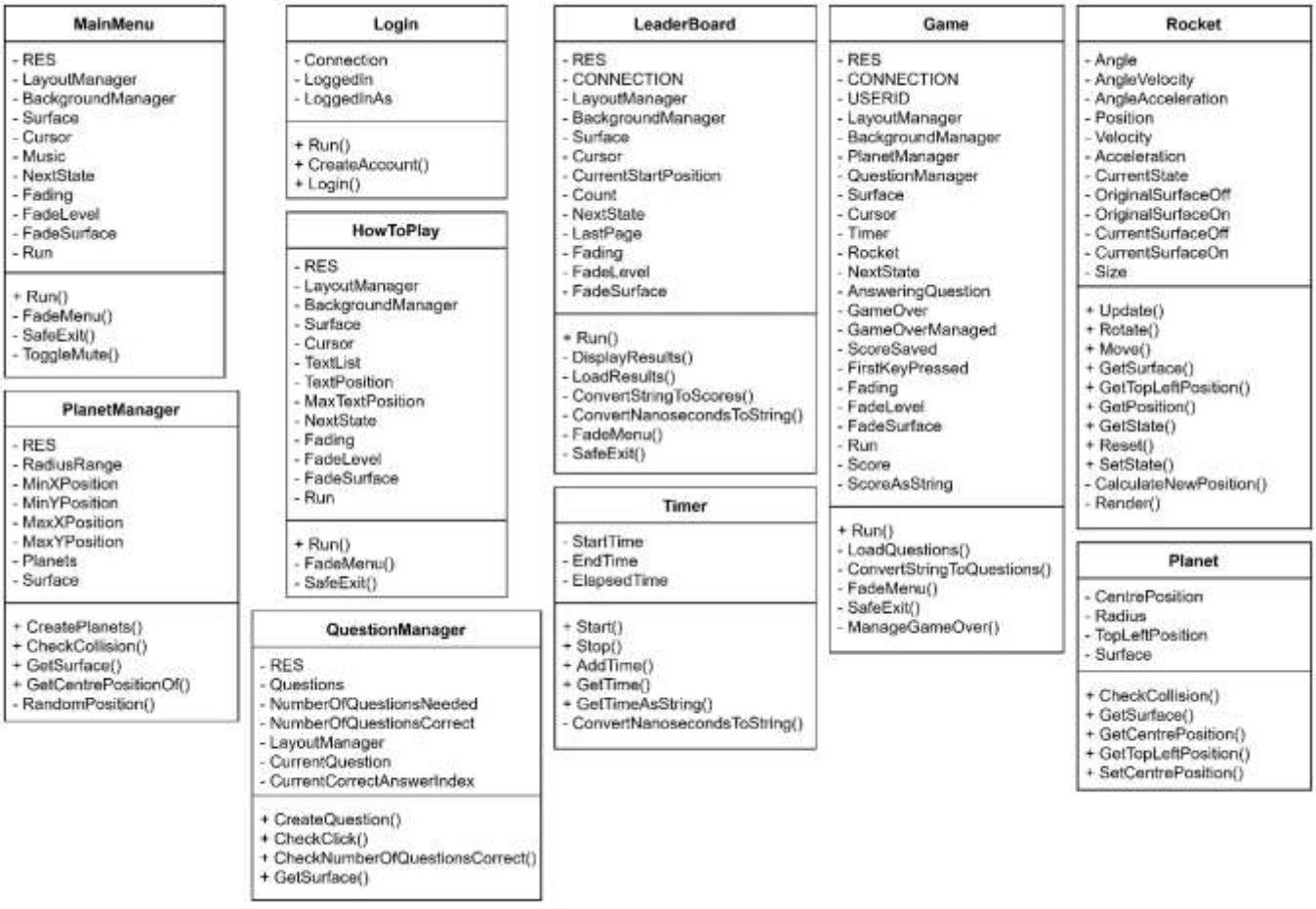
To overcome this, I created class definitions for each class in the program. By examining the class definitions, you can get a better understanding of the class structure and the functions and data that are encapsulated within each class.

The class definitions provide a detailed breakdown of the methods and attributes associated with each class as well as if these are public, protected, or private. They help to explain how each class is designed and how it functions

within the whole program.



The class definitions above show most of the modules which are used in multiple places in the program.



The above class definitions show the main sections of the program across the top as well several more modules that are used in the program.

Data Structures

Throughout the program many data structures will be used. I will carefully choose which data structure to use for a certain variable as this can greatly improve the programs performance. The data structures I intend to use are summarised in the table below.

Data Structure	Example of use (a single example)	Explanation
String	To store the username and password when the user creates a new account.	Any input from the user when on the login stage will be as a string so that they can enter their username and password.
Integer	To store the user's mouse position on the screen.	The user's x and y co-ordinates of their mouse can only be integers.
Float	To store the user's score when they are playing the game.	Their score is based off of how long they have been playing the game for so this can have a decimal part to it.
List	To store all the buttons or textboxes on the screen.	The main menu will have multiple buttons on it so the button objects will be stored within a list.
Dictionary	To store all the constants of the program so they can be managed from a single file.	So that constants can be changed easily, they are all stored in a single JSON file. This file in then imported into a Python dictionary so each constant can be accessed using its key.
Boolean	To store whether or not the user has completed the game.	When playing the game, the GameOver variable stores True of False for if the user has completed the game or not.
Queue	To store the questions which the user will have to answer.	When playing the game, the questions which the user will be shown are stored in a queue. The program gets the next question by taking the next question in the queue.
Bytes	To send data over the socket connection between the client and the server.	The socket connection can only send binary so all data is converted into bytes for it to be sent over the connection.
Object	To store an instance of the Login class.	When the program starts, it creates an instance of the login class which then allows the user to create an account or login. This is stored within an object.
Tuple	To store the position of the mouse / cursor.	The tuple stores the mouse position as the x and y values always change together and will be accessed at the same times.
Vector	To store the position of the rocket on the screen.	The position of the rocket on the screen is stored as a vector which describes the components of displacement of the rocket from the top left corner.

Data Validation

Throughout the program, the user will interact with the program in several different ways. Almost all of these are by pressing keys or using the mouse. The only time the user can input data is when logging in or creating an account. Therefor it is vital that the data inputted by the user is validated. The below table shows the types of validation that will happen:

Validation Check	Description	Fields applicable to	Valid data	Invalid data

Data Type	Data must be in the correct data type.	None – All fields are required as strings and all user input is as a string.	“Henry”	N/A
Length	Data must not be too long or too short.	Username, Password, First Name, Last Name	“Password123”	Fields longer than 1024 characters will cause database error.
Presence	Data must exist and cannot be blank.	Username, Password, First Name, Last Name	Any data that isn’t nothing.	Data that is blank or nothing.

The table below shows what types of validation will be used for each field and how the user will be notified to the error:

Field	Validation Checks	Description	Error Message
Username	Length, Presence	Must not be longer than 1024 characters and must not be nothing.	Invalid username
Password			Invalid password
First Name			Invalid first name
Last Name			Invalid last name

Although the maximum length of each field in the database is 1024 characters, to reduce the workload on the server and client’s internet connection, I will limit all fields to 128 characters.

Database Design

Identifying the Data

Before designing the database, I first planned what would need to be stored. To start I made an overview of what the database will need to store:

- The user’s login and personal details.
- The user’s scores.
- The questions and answers.

Now knowing what the database will be used for, I planned what specific fields will be stored. Relating to the user, I will store their Username, Password and Name. For the questions, it will store the question and answers as well as which answer is correct. Finally, it will store all the user’s scores.

Storing Passwords

Knowing of the risk of storing passwords, I did lots of research into how passwords should be stored in a database. One source I read was an article on LinkedIn[7] which talks about the options for storing passwords. Below is a table which outlines the options.

Name	Description
Plain Text	This is the worst method as if an attacker gains access to the database, they will have immediate access to the password. Although they will already have access to all data in the database, they will be able to use the password to login other sites. This is called credential stuffing.
Encryption	This method is very good as it means the attacker would not have immediate access to the password. But, to encrypt the passwords we need to use a key. Once the attacker gets this key, due to encryption being bidirectional, they will be able to decrypt and have access to all the passwords.
Hashing	The best method is hashing which is a 1-way algorithm that cannot be reversed. This means that the attacker will not be able to get the passwords without brute force.

As stated in the table above, the best method for storing passwords is using hashing. This is the method I will use for storing passwords within my database. I also found that hashing alone has problems such as hash-lookup-tables. To avoid these problems, I will use a salt which is random data added to the password before hashing making each hash completely unique even is 2 users have the same password.

Design 1 – Unnormalized

My first design of the database included the fields I have already stated but they are split into 3 tables, Users, Scores and Questions. Each table would contain the following fields:

Users (Username, PasswordHash, FullName)
Scores (Username, Score)
Questions (Question, CorrectAnswer, Answer1, Answer2, Answer3, Answer4)

For this to be implemented in a database, the tables need to be normalised. This is to minimize data redundancy and dependency and improving data integrity and consistency. By normalising the database, it will become more efficient, flexible, easier to maintain, and easier to query.

Design 2 – 1st Normal Form

To normalise the database into 1st NF there are 3 criteria that must be met:

1 - Each record must have a primary key

For each table in the database, I need to identify what can be the primary key (a field that is unique to every record).

For the Users table, the Username will be unique for each user so this could be used as the primary key. However, I will not use the Username as the primary for these reasons:

- Changeability. If a user's username changes, it would affect all the foreign keys referencing the user's primary key in other tables, making it hard to maintain the consistency of the database.
- Performance. Usernames tend to be longer than integers and that could have an effect on the storage space and performance of the database.
- Data privacy. Due to the possibility of the Username containing personal information, it would not be good to store it in many different places and tables as this could make managing data privacy more difficult.

For these reasons I will instead create another field called User ID which will uniquely identify each user. The User ID will be an integer therefor if the User ID is exposed to anyone, it will have no meaning without access to the database.

For the Scores table, this means that instead of using the Username as the primary key, I will use the User ID and Score to make a composite key that will be the primary key of the table.

For the Questions table the I will make a new field called Question ID which will be used for the same reasons as the User ID.

2 - The data in each field must be atomic

This means that the data in each field should be the smallest, indivisible unit of information. Atomic data cannot be broken down further into smaller pieces of data and needed as it allows for more efficient data management and improves data integrity.

For the Users table, this means the Full Name field will be split into First Name and Last Name.

3 - Each record must have no repeating groups of attributes

This means that a record, or row, in a table should not contain multiple copies of the same information or type of information.

This affects the Questions table as the Answers are a repeated group of attributes. Therefore, I will create a new table called Answers which will store the Answers for each question. Each answer will have a Boolean field which stores if the answer is the correct answer, the Question ID as a composite key with the answer to make the primary key.

The new database design in 1st normal form looks like this:

Users (<u>UserID</u>, Username, PasswordHash, FirstName, LastName)
Scores (<u>UserID</u>, <u>Score</u>)
Questions (<u>QuestionID</u>, Question)
Answers (<u>QuestionID</u>, <u>Answer</u>, <u>IsCorrect</u>)

Design 3 – 2nd Normal Form

For the database to be in 2nd normal form, the requirement is that there are no partial dependencies.

This affects the Answers as the IsCorrect field is only partial dependant on the composite key. To fix this, I will add a new field called AnswerID which will become the primary key.

It also affects the Scores table as Score is only partially dependant on the UserID. Therefore, A Score ID will be needed to uniquely identify each user's score.

This means the new database design is as follows:

Users (<u>UserID</u>, Username, PasswordHash, FirstName, LastName)
Scores (<u>ScoreID</u>, UserID, Score)
Questions (<u>QuestionID</u>, Question)
Answers (<u>AnswerID</u>, QuestionID, Answer, IsCorrect)

Design 4 – 3rd Normal Form

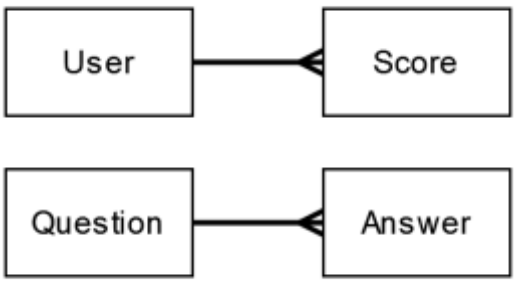
The requirement for 3rd normal form is that there must be no non-key dependencies. A non-key dependency occurs when a non-primary key column in a table is dependent on a column or set of columns that is not part of the primary key. This does not affect any of the tables meaning the final design is the same as Design 3.

Design 5 – Final Design

Although the database design is now fully normalised and in 3rd normal form, to make the database more standardised, I will rename the tables to describe what a single record is instead of what the whole table contains. This means the new, and final, database design is:

User (<u>UserID</u>, Username, PasswordHash, FirstName, LastName)
Score (<u>ScoreID</u>, UserID, Score)
Question (<u>QuestionID</u>, Question)
Answer (<u>AnswerID</u>, QuestionID, Answer, IsCorrect)

The database design has an entity-relationship diagram as follows:



Each User will have multiple Scores for every time they play a game, similarly, each Question will have 4 Answers. The UserID will be a foreign key in the Score table, likewise, the QuestionID will be a foreign key in the Answer table. There is no relation between (User or Score) and (Question or Answer) because they are unrelated and aren't required for my program.

SQL Queries

To use the database within the program, I will need several SQL queries. These will allow the program to create, modify, and retrieve data from the database. Below are the SQL queries that my program will use:

To add a user:

INSERT INTO user (Username, PasswordHash, FirstName, LastName) VALUES ("henryw", 4376463363, "Henry", Walker);

To add a question and answer:

INSERT INTO question (QuestionText) VALUES ("Which one of these is energy?");

INSERT INTO answer (QuestionID, AnswerText, IsCorrect) VALUES (23, "67 J", 1);

To add a score:

INSERT INTO score (UserID, Score) VALUES (5, 34956459600);

To get a question and its answers:

SELECT QuestionText, AnswerText FROM question, answer WHERE question.QuestionID = answer.QuestionID AND QuestionID = 26;

To get the details for the leader board:

SELECT Username, Score, FirstName, LastName FROM user, score WHERE score.UserID = user.UserID ORDER BY Score ASC LIMIT 10 OFFSET 0;

Request-Response Design (Client-Server Design)

Queries

As my program will be using a database to store the questions, login details, and scores, the program will need to have a way to insert and access data from the database. To do this, there will be a server which the program will connect to which will allow the program to request data to be inserted or accessed. To make the server secure, it must have a limited number of ways in which it can be used. To limit its use, I will only allow certain requests to be made. Each request will have a unique code which will be used to identify the type of request as well as a number of parameters which will be unique to each type of request. The table below outlines all the queries the server will accept:

Query ID	Query Name	Purpose	Parameters	Response Required
100	Add User	To add a new user to the database.	Username, Password, First Name, Last Name	No
101	Add Question	To add a new question to the database.	Question, Correct Answer, Wrong Answer 1, Wrong Answer 2, Wrong Answer 3	No
102	Add Score	To add a new score to the database.	User ID, Score	No
103	Check Login	To check the login details when someone tries to log in.	Username, Password	Yes
104	Get Random Questions	To get random questions from the database for the user to answer.	Number of random questions	Yes
105	Get Scores	To get the scores for the leader board.	Start Position, Count	Yes
106	Check Username	To check when someone is signing up, that the username hasn't already been used.	Username	Yes

The Query ID will be very important as it will be used to identify the requests type before the rest of the request is processed. It needs to be 3 digits long (a fixed number) so that the server knows to read the first 3 characters and use that to determine the type of request.

The table above contains every query that the program will need to make to the server for it to run correctly. If any maintenance needs to happen to the database, this will be done directly in the database which would require physical access to the device. This greatly increases the security of the database as it removes the risk of SQL injection attacks.

Request-Response Structure

When the request and responses are being sent, they can only be sent a single string. This means that all the data will be sent together so there will need to be a way to separate the data back into its original form. Therefore, throughout the program separators will be used to represent the start and end of different pieces of data. The below table shows the separators that will be used and what for:

Separator	Purpose
##	To split the parameters in the request.
&&	To separate items in a list.
$$	To represent a new line in text for a textbox or a button.

The separators all use character patterns that are highly unlikely to be needed so that they don't cause problems when the user inputs data or questions are created. To make sure errors can't happen by the user inputting a separator, all user inputs will be validated to make sure they do not contain any of these character patterns.

Example Requests and Responses

Description	Request	Response
Add a user to the database	100##henryw23##Pass123##Henry##Walker	"None"
Check the login details	103##henryw23##123456	"False"
Get questions	104##8	"Take 2$$add 5$$times by 3$$what do you get?&&2&&32&&12&&21&&28…

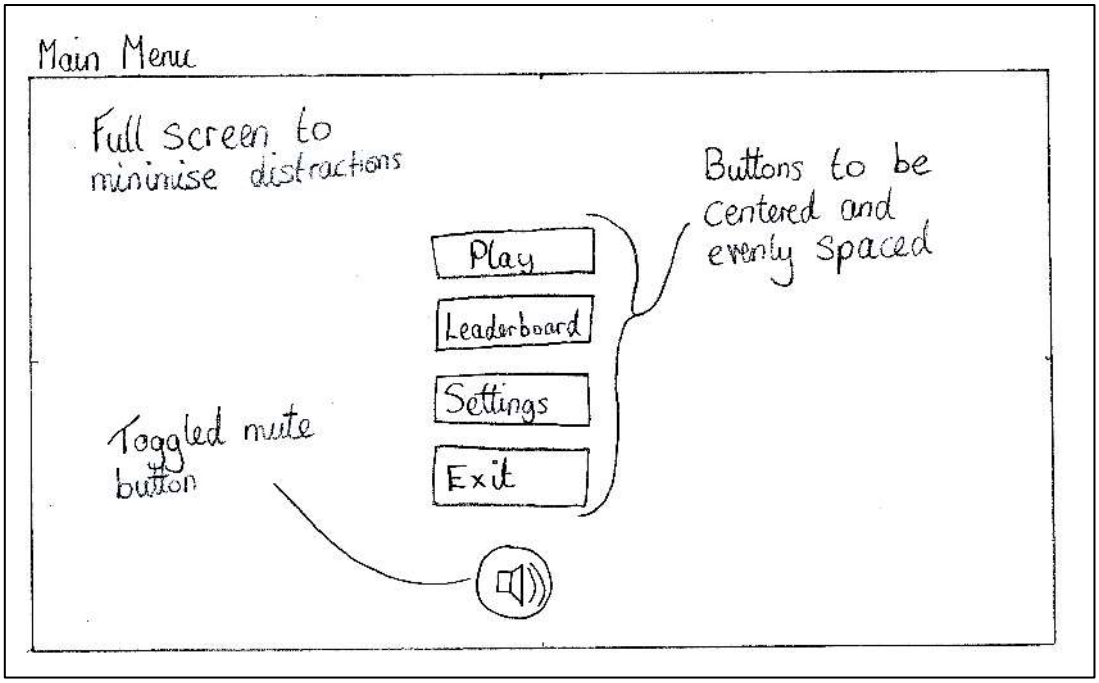
The last response shows how multiple separators will be used at the same time without causing problems.

Layout Designs

Before creating the program, I need to design how the program will look. The overall look of the program should be simplistic and intuitive so that the user doesn't require putting in a lot of time or effort to understand how to use the program.

Main Menu Design

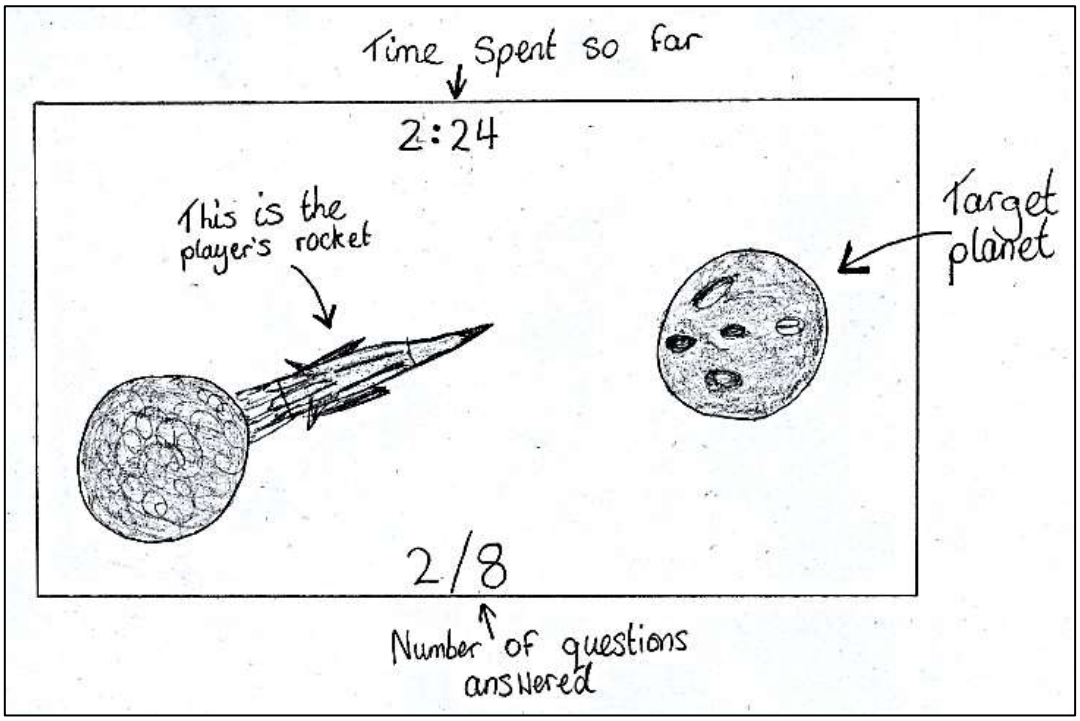
The first part of the program the user will see is the main menu and this will allow the user to navigate to the different sections within the program. Below is an initial drawing as to how the main menu could be laid out:



What is good about this design is that it's simple and intuitive due to having the buttons and nothing else. What I don't like is that the toggle mute button doesn't match all the other buttons. The final design will have rectangular buttons centred in the middle of the screen.

Game Design

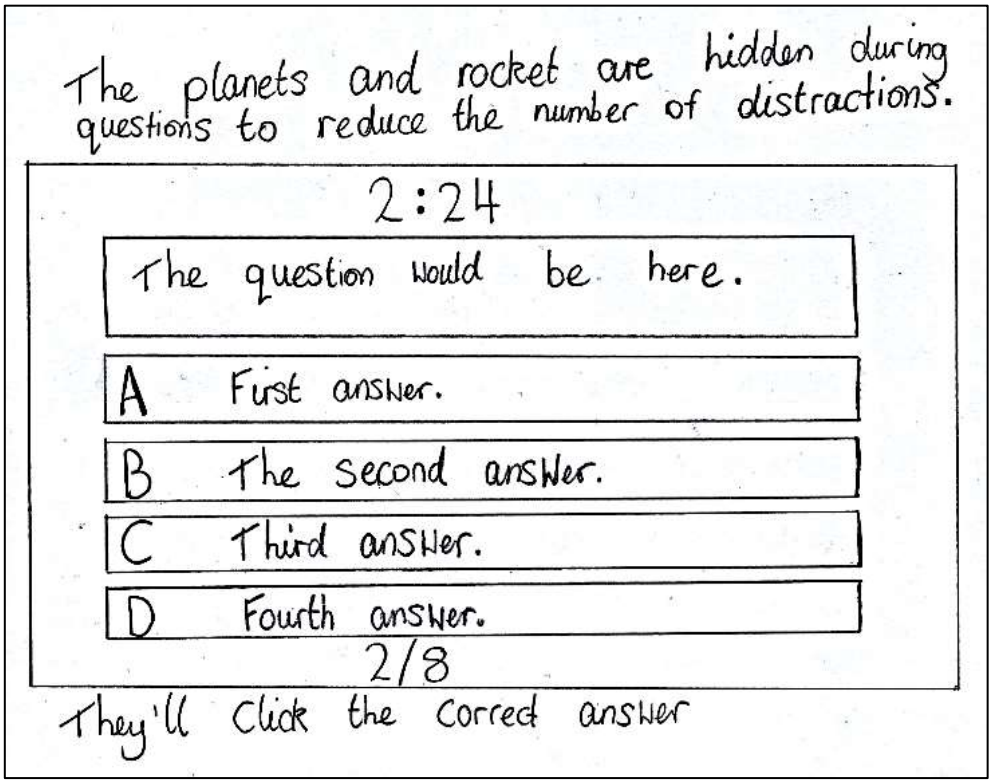
When the user is playing the game there are 2 different 'scenes', they will either be flying the rocket or answering a question. To get a good understanding on what the game should look like, I discussed it with my client. I asked what the user should be shown while playing the game and their response was: "Only give them what they need to know. They don't really need to be able to see all the planets, just the next one. So show them like the time and how many questions they have left". Based off that conversation the following drawing is the design for when the user is flying the rocket:



The time and number of questions answered will be displayed throughout both 'scenes' and when flying the rocket only the current planet, next planet, and rocket will be shown.

I also discussed with my client about what the user should be shown when answering a question. Their response was "Like before, they should only be shown what is nessecary. Just the question and answers is enough. But make sure

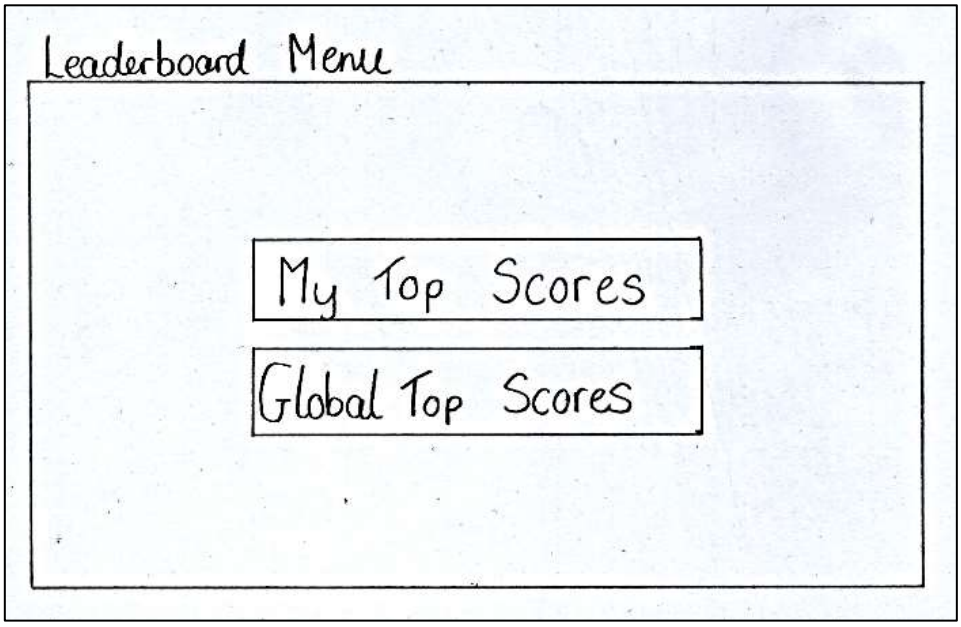
the questions and answers are short and quick to answer otherwise it could be too difficult for the user." Using this, I designed the following for when the user is answering a question:



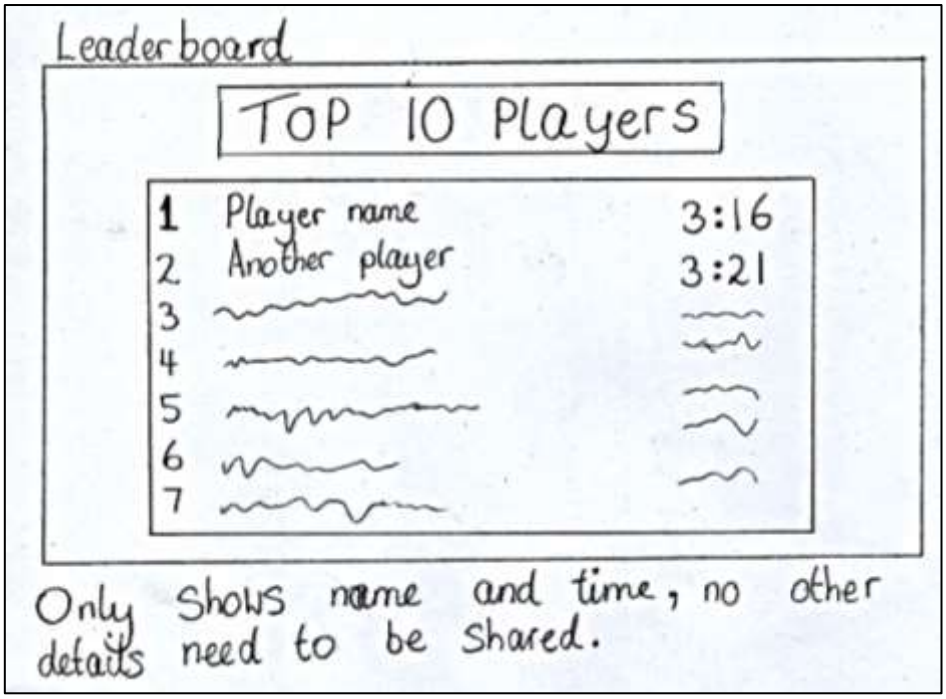
The question and answers will be in the same shaped box as the buttons in the main menu and the answers will be large making it easier for the user to click them quickly.

Leader Board Design

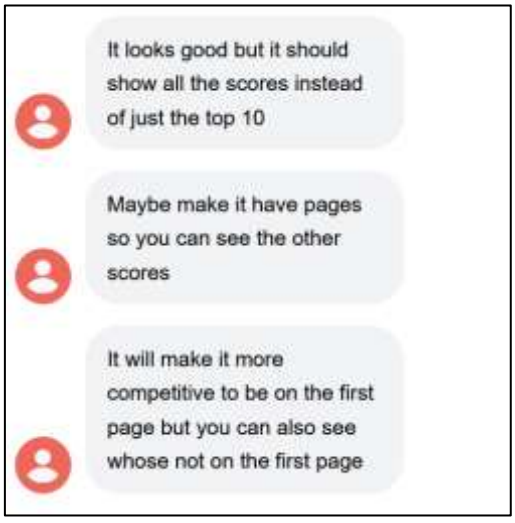
From the main menu or once the user has completed the game, they will be able to view the leader board. Initially, I was planning on having 2 different leader boards, one to show everyone's scores and one to show the current user's scores. I designed this menu to allow the user to choose which one they wanted to see:



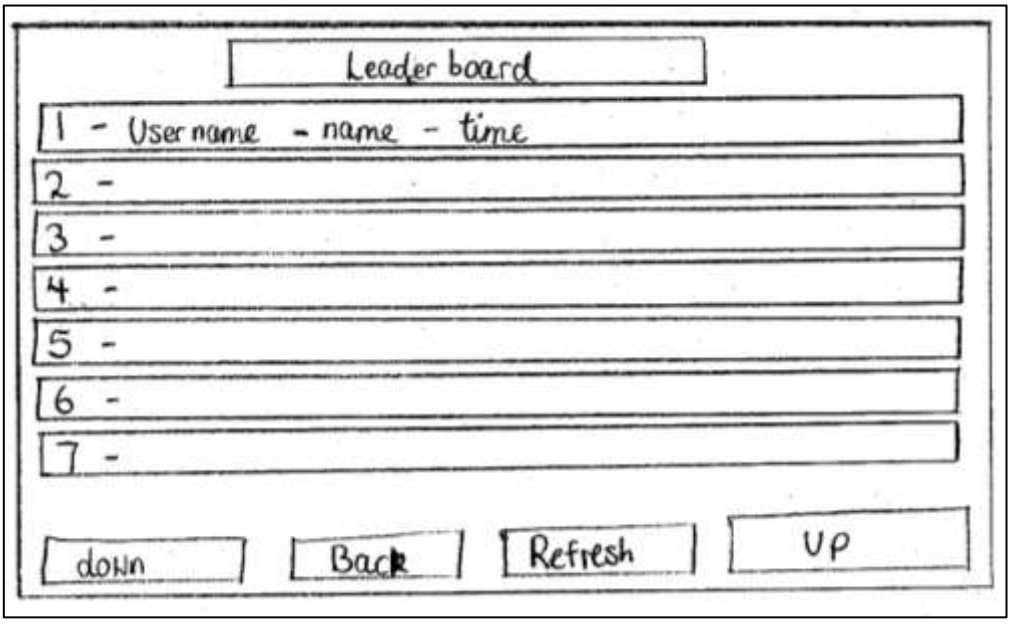
After talking with my client, they said that the competitive side of the program is more important and that having 2 leader boards could make it more complicated than it needs to be. Following what my client said, I designed the leader board as follows:



I went back to my client with this design asking for feedback and this is their response:



Following the feedback from my client, I redesigned the leader board based off what was discussed. The final design for the leader board is as follows:



Each page will have 10 scores on it with buttons to refresh the scores, change pages, and go back to the main menu. Each row will show one result and will display: the position, username, name, and time.

Stages of Design

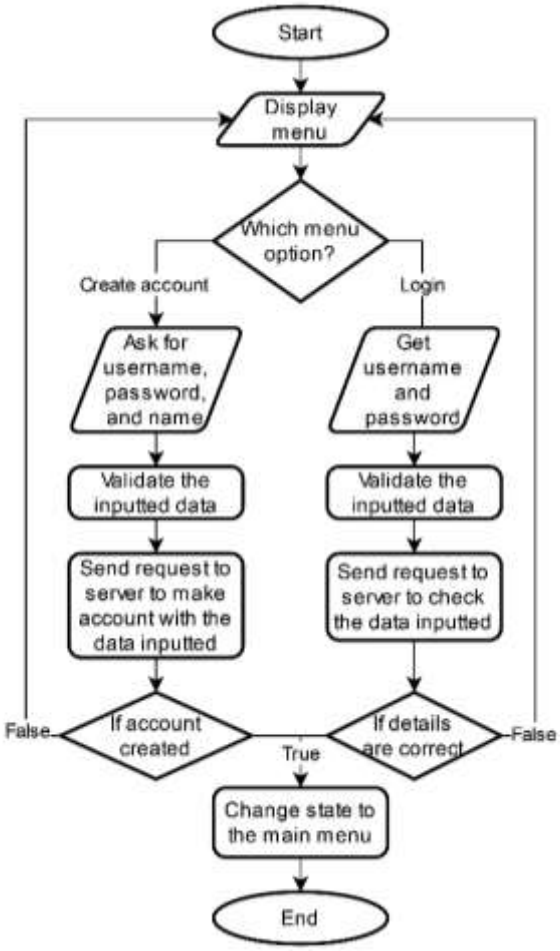
To make the program I have broken it down into smaller sections to work on so that it is more manageable, easier to code, and easier to debug when testing. These are the stages of design:

1. Login System
2. Main Menu
3. Game
4. Leader Board
5. Server

1 – Login System

Before the user can access the program, they will first need to create an account or log into an existing account. This account will be used to save results needed for the leader board.

The login System will follow the following flowchart:



The user will not be able to get into the program without either creating an account or logging into an existing account. I designed it like this so that all results will get saved in the database and can be seen on the leader board.

To create the Login System, I will follow these steps:

1. Create a menu where the user chooses between Login or Create Account.
2. Create a function validate inputted data.
3. Create the Login function which gets the username and password, validates the inputs then sends request to the server.

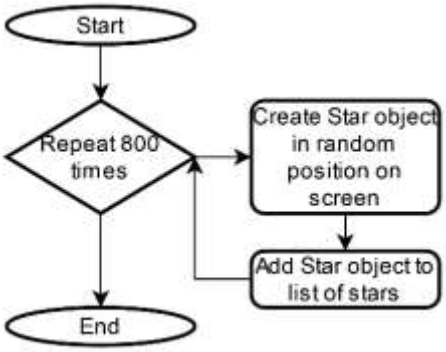
4. Create the Create Account function which gets the user's details, validates them, and sends them to the server.
5. Check response from the server and either: Return the user to the menu or take them to the main menu.

2 - Main Menu

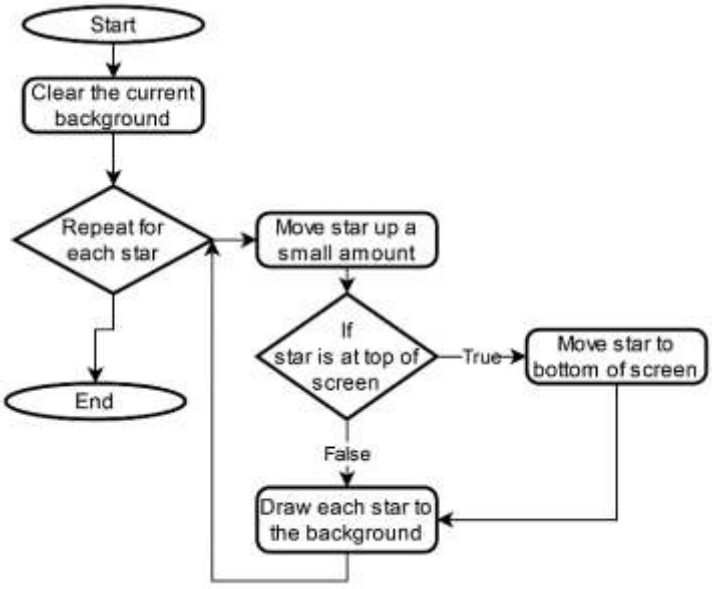
The first part of my program the user will see once they have logged in is the main menu. The user will use the main menu to choose between: Playing the game, seeing the leader board, exiting the game, and muting the music. Using the layout from the Layout Designs section, the buttons will be rectangular and be centred in the middle of the screen.

The main menu will have several classes that need to be used throughout the program, these are: Textboxes, buttons, and the background. I will code these first so I can then use them to make the main menu.

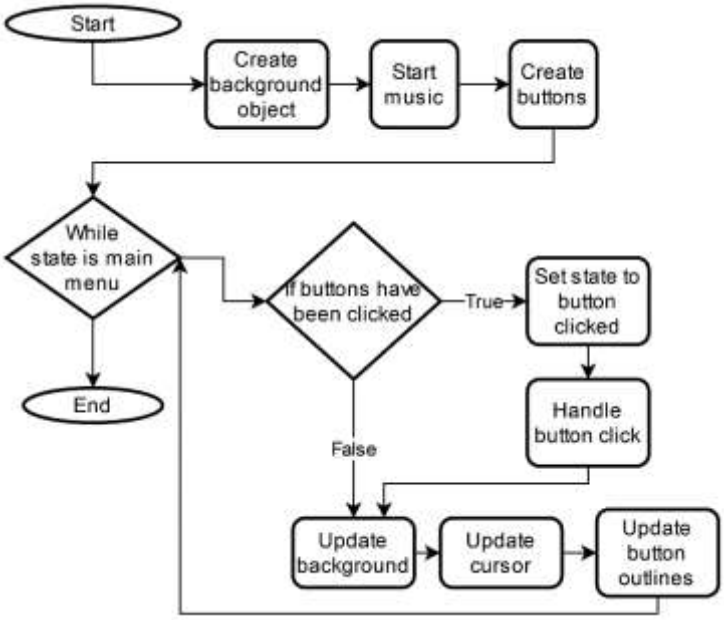
To make the game seem a bit more space-like, the background will have lots of small flying stars. These will be tiny circles that move up -the screen. This background will be used throughout the whole program to connect each section together giving the user a sense of familiarity. The flowchart belows shows how the program will create the stars when the main menu is started:



Once the stars have been created, each time the Main Menu loop cycles (each frame), each star will be updated using the following flowchart:



The main code for the main menu will run using a while loop that runs every frame until the state of the program changes. The code will follow this flowchart:



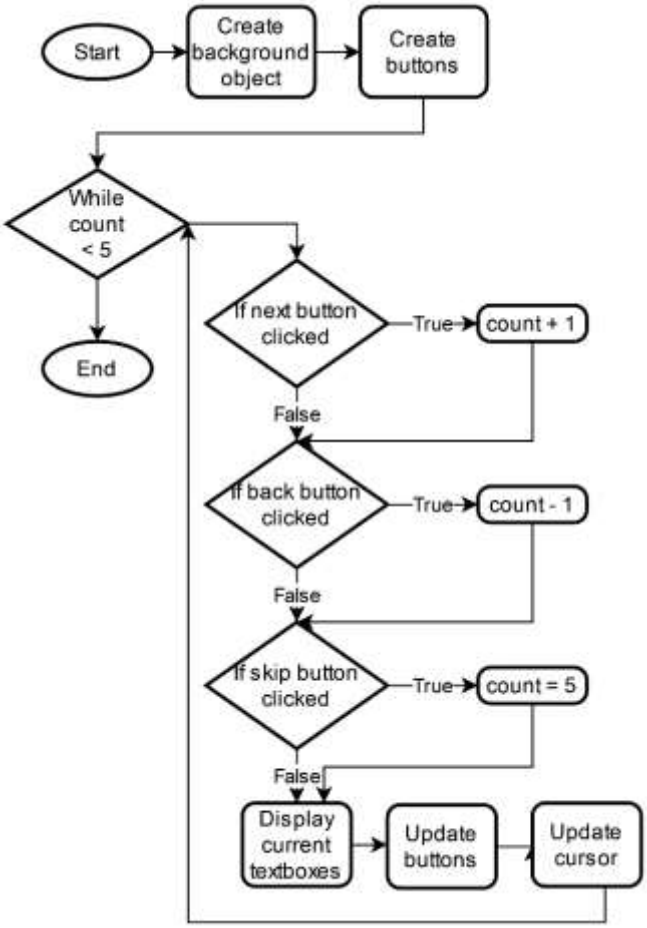
Depending on the button that was clicked, different things will need to happen. If play, leader board, or exit are clicked then only the state will be changed to the corresponding button. If the mute button is clicked, then the music will be toggled between on and off.

To create the main menu, I first need to create the modules mentioned above. Therefore, my steps to making the menu is as follows:

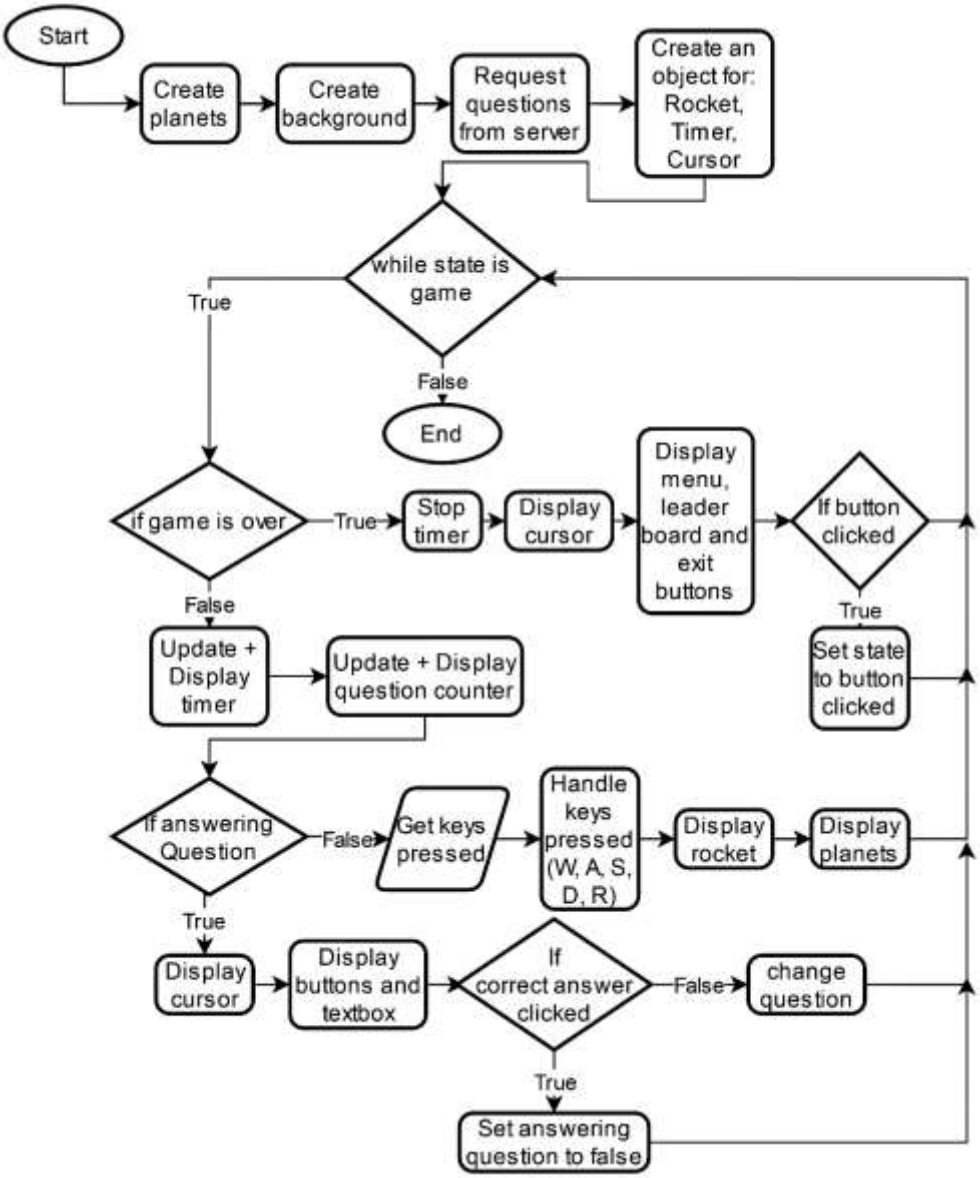
1. Create the background module.
2. Create the textbox module.
3. Create the button module which inherits form the textbox module.
4. Create the music module.
5. Create the main menu using the above modules.

3 – Game

The game is going to be the largest part of the program as it is where the user will spend most of the time when using the program. Once the user clicks the play button in the main menu, they will first be shown instructions on how to play the game. These instructions will be stored within the program as they will not need to change. This section will be called the “How to Play” section and they can either press the next button to read more instructions or skip to go straight to the game. The instructions will run using this flowchart:



After they have completed the instructions, they will be in the game and the timer starts counting. This section will be split into 2 smaller sections: the rocket-flying part, and the answering questions part. When the user is flying the rocket, they will have to fly from the planet they are currently at to the other planet on the screen. To make this harder, they will have positional momentum as well as rotational momentum meaning once they stop pressing a key, they will carry on 'sliding' (like on ice). When they reach the planet, they will be shown a question as 4 multiple choice answers. They will have to click the correct answer to move on and fly to the next planet. If they get the question wrong, then they have to keep answering questions until they get one correct. Once they answer 8 questions correctly, the timer stops and it shows their time with buttons to take them to the main menu, leader board or exit. The game will follow this flowchart:

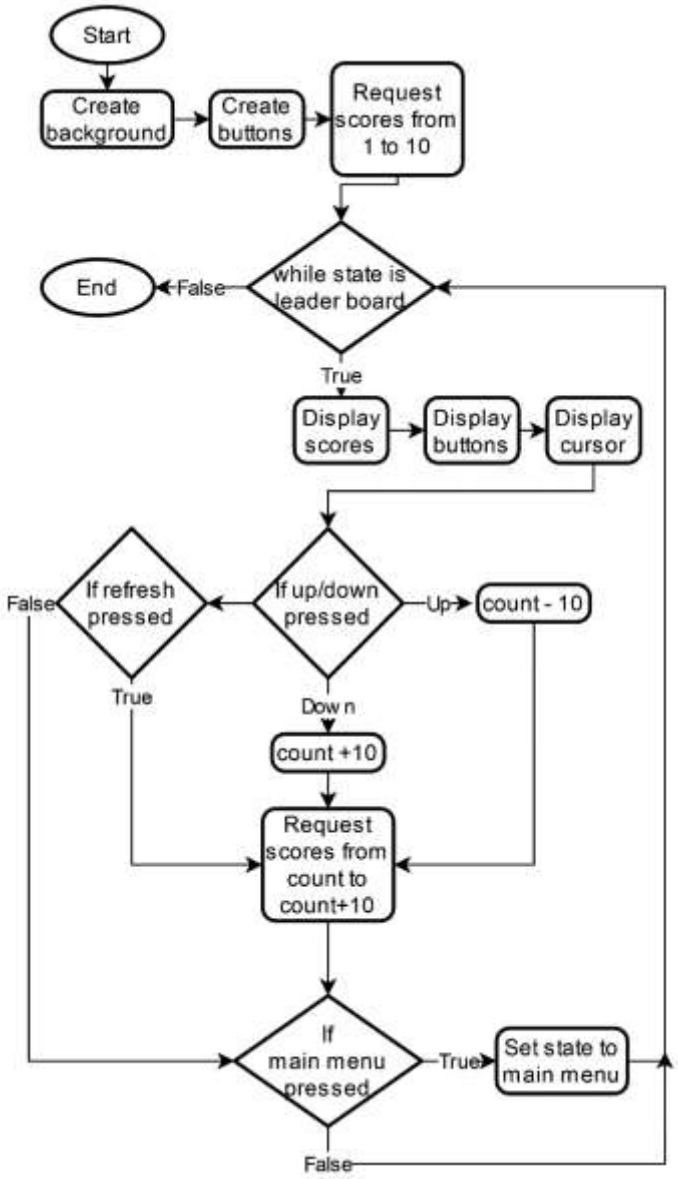


When I come to coding the game section, to make it more manageable, easier to debug, and easier to test, I will code it part-by-part in this order:

1. Create the “How to Play” section.
2. Create the “Game” section.
    a. Create the rocket class.
    b. Make the rocket respond to inputs.
    c. Create the planet class.
    d. Create the planets in random positions and sizes.
    e. Create the question manager class.
    f. Make a question appear when rocket is detected near the planet.
    g. Make timer and question counter.
    h. Display time and menu buttons once 8 questions have been correctly answered.

4 – Leader Board

Once the user has completed the game or from the main menu, they will be able to look at the leader board. This will show all scores stored in the database is order from quickest time to slowest time. Each page will show 10 results and the page can be changed using buttons on the screen. There will also be a button to refresh the results and another to exit back to the main menu. The following flowchart shows how the code for the leader board will run:



Based off the above flowchart, the steps I will take to code the leader board are:

1. Create a function to retrieve scores from the server.
2. Create a function to create a textbox for each result (row).
3. Create the main loop to display the results and respond to button presses.

5 – Server

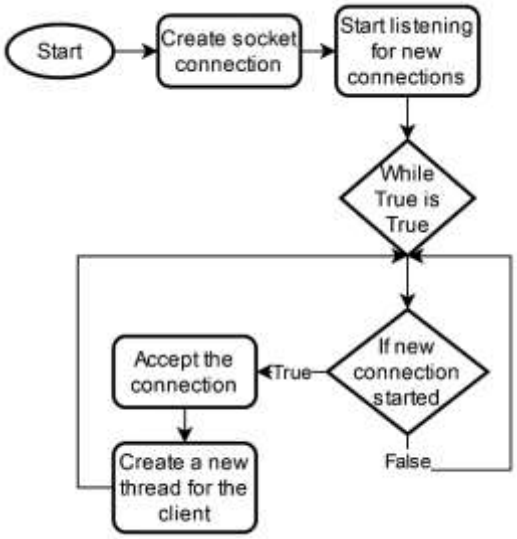
The server is the most vital part of the program as without it, nothing will work – you can't login, you can get the questions to play the game, you can't save your score, and you can't see the leader board. Due to it being such a vital part of the program, it needs to be well designed so that it doesn't crash and can handle errors when they arise. The main functions of the server are to:

- Create new accounts.
- Check login details are correct.
- Select random questions for the user.
- Save the user's scores.
- Respond with the results for the leader board.

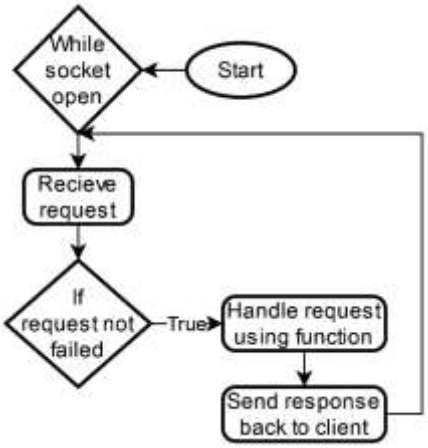
The server will be split into 2 sections, one which has specific functions to interact with the database, and the other section will receive the requests, process them, and send a response.

The code that interacts with the database does not run together, instead it is functions that are called when required.

On the other hand, the code for the server will run using the following flowchart:



As shown above, the main part of the server will make a new thread on the CPU for each client that connects. This is needed because the server must wait for the client to send a request which means for the server to be able to handle multiple clients (connections), it must be able to switch between clients to handle requests as soon as they come in. This is what a thread does – it allows me to run the same function multiple times simultaneously. Once a new thread is created, the following flowchart will be followed for each client:



Above, where it says 'handle request using function' this is where the corresponding function will be executed to query the database using the correct SQL query.

File Structure

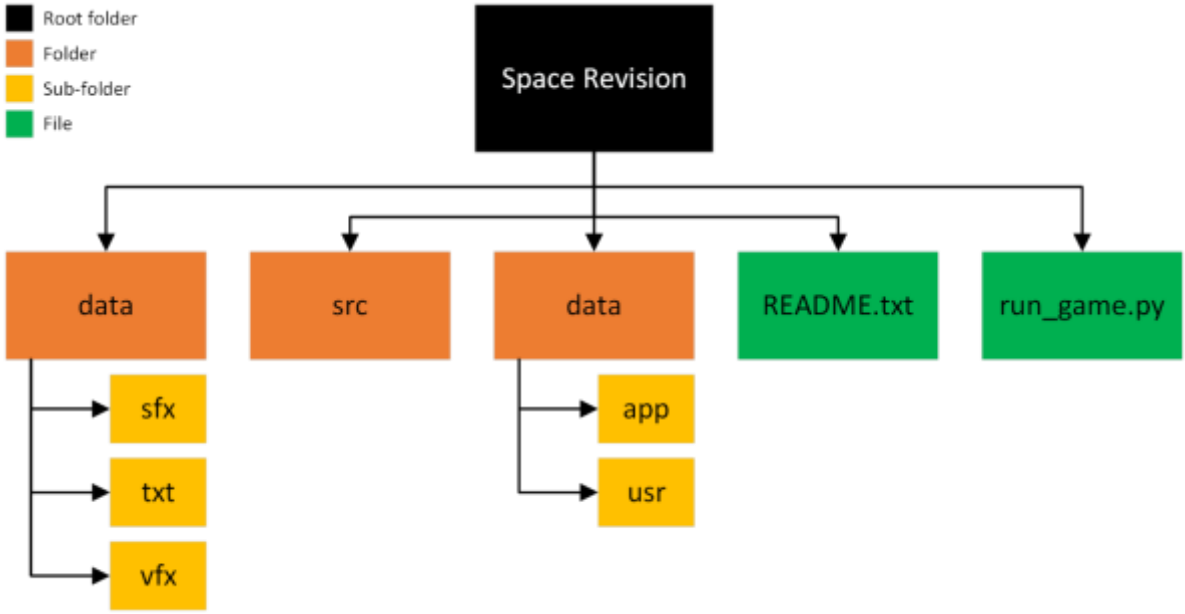
As the code will be open-source and free to use or modify, I will make the file structure easy for others to understand and use themselves. Before coding anything, on the 14th of August 2022, I spent a few hours researching common and standard ways to organise python files and games. After realising there isn't a "standard" as python has so many uses, I found PyWeek. It is a Python game-development competition which has some good tips for structuring Python games. From reading the guidance document[8] and looking at several of the previous winning games, I came up with the following …:

1. There will be a run_game.py file in the main folder so that users aren't confused by having lots of options. This file will be used to start the game and will also check that the user has the correct version of Python.
2. There will be a README file within the main folder as well so that important information can be shared with the user. This will include:
    a. The accepted Python version.

b. The accepted PyGame version.
c. Information for users who wish to modify the code.

3. The final program will be downloadable in the form of either the full Python source-code or an executable. This will be up to the user and their intended use. The full source-code will be for users looking to understand and modify the program whereas the executable won't require Python or PyGame being installed so will be better for someone looking to just use the program.
4. There will be folders for specific groups of files so that it's easy for others to know where to find a certain file. These folders will be for assets (graphics, sounds and text), Python files, and data (app data and user data).

Using the above rules, the file system I created looked like this:



Technical Solution

Overview

To give you a better understanding of the program and what it achieves, I have compiled lists of the most important parts of the project.

Most Complex Algorithms

Firstly, the list below explains what the most complicated algorithms are, and what they do. It also contains page numbers as to where each of these can be found in the technical log as well as anywhere else that is relevant.

Algorithm	What it does, and why its complex	Page numbers
Calculating new position of the rocket	It uses trigonometry to calculate the new position of the rocket based off of its current speed and current direction. This is explained in the modelling of the analysis, as well as in the game.py file.	
Connection module	Uses a socket connection to connect to the server over the internet. The complex part is that it must convert data to binary, and then generate a header to the data so that it can be received correctly.	
The server	The server manages the connections and starts a thread for each client. Multi-threading is a complex technique and is vital so that the server functions. The server must be able to send and receive data all the time so must manage what errors may occur.	
The query manager	The query manager is complex as it connects to the database and must use the inputs to get the correct data from it. The GetRandomQuestions function within the query manager is rather complex as it gets random questions from the database and then shuffles the answers. It does this in a way in which the answers won't be in the same place and also no questions will be repeated.	
The background manager	The background manager uses complex mathematical calculations to determine the number of stars that should be displayed on the screen at one time.	

Technical Skills

Below is a table which gives an overview of what models have been used, and where. These models are taken directly from "Table 1: Example technical skills". Again, I have included page numbers as to where these are programmed and talked about.

Group	Model Used	Where It Was Used	Page Numbers
A	Complex data model in database (e.g., several interlinked tables)	The MySQL database has several tables within it and as shown in the design section, these tables are interconnected.	
	Hash tables, lists, stacks, queues, graphs, trees, or structures of equivalent standard	Query_manager.py – Hashing is used to securely store the passwords or users. Backgroundmanager.py – Lists are used to store the star objects.	

		game.py – Within the question manager, the questions are stored within a queue (Within Python this is implemented as a list but is a queue due to how it is used). Questions are only ever taken from the front of the queue.	
	Files(s) organised for direct access.	As described in the design section under the File Structure, there are many files within the program which are accessed directly by the program while it is being used.	
	Complex scientific / mathematical / robotics / control / business model	Game.py – Within the rocket class and layout manager class, complex mathematical models are used to abstract the problems so that they can form a simpler interface for the program to interact with. Main.py – As described in the Design section, the program follows a complex control model where it hands control between different sections and files in a way that is managed by the user.	
	Complex user-defined use of object-orientated programming (OOP) model, e.g., classes, inheritance, composition, polymorphism, interfaces	As described in the Design section, the inheritance diagram shows how the program uses inheritance, composition, and aggregation. What it doesn't show is the interfaces for each class which are then shown in the class definitions diagrams. Inheritance is used in the button class to inherit from the textbox class. When this is done, it also overrides a method to provide a different purpose (Polymorphism). Lastly, polymorphism is also used within the textbox class which can use different algorithms depending on if the input is a string or a list.	
	Complex client-server model	This is described thoroughly in the Design section where it outlines how the client and server are an integral part of the program. The client connects to the server at multiples times throughout the program, and the server connects to other interfaces to perform complex requests and form responses.	
B	Multi-dimensional arrays	Game.py – Within the game, 2D arrays are used to store the position of the rocket and planets.	
	Dictionaries	When the constants file is loaded, it is stored within a Python dictionary. This is used within many sections of the program. (Example: main_menu.py)	

As well as the models shown above, the below table outlines the algorithms I have used within my program and where these have been implemented. These algorithms are also taken directly from "Table 1: Example technical skills". Again, page numbers are included to guide you to where these are in the document.

Group	Algorithm Used	Where It Was Used	Page Numbers
A	Cross-table parameterised SQL	Query_manager.py – To get the scores, names, and usernames for the	

		leader board it queries the Score and User tables in the same query.	
	List operations	Background Manager (background.py) – It stores all the stars within a list. It uses append to add stars to the list.	
	Stack/Queue Operations	server.py, game.py – The server gets questions from the database and then shuffles them before sending them to the client. The game then turns the questions into a Queue as it takes the next question from the queue when needed.	
	Hashing	Query_manager.py – When a user creates an account or logs in, their password will be hashed using a hashing algorithm so that it can be stored securely.	
	Complex user-defined algorithms (e.g., optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty	Background Manager (background.py) – The background manager uses a complex algorithm to manage the stars so that they look like an endless loop. Query_manager.py – To get random questions from the database, it uses a complex algorithm to request the questions, shuffle them, and order them for the game to be able to use.	
	Merge sort or similarly efficient sort	Query_manager.py – The query manager uses an efficient sorting algorithm within the database to sort the scores before sending them to the client for being displayed in the leader board.	
	Dynamic generation of objects based on complex user-defined use of OOP model	main.py – When the program starts it generates an instance of the Login class, but once the user is logged in, the generation of other classes is entirely dependant on the user's inputs. This creates a complex OOP model as planned in the design section.	
	Server-side scripting using request and response objects and server-side extensions for a complex client-server model.	leaderboard.py – The leader board sends requests to the server to get the scores needed for the current page to display the scores to the user. server.py – The server receives the requests and decodes them before passing them to the query manager. Query_manager.py – This is a server-side extension that processes the request and forms a response.	
B	Single table or non-parameterised SQL	Query_manager.py – When the user is attempting to login, it queries the database for the Password Hash of	

		the username inputted. This query is to the User table.	
	Writing and reading from files	Consts.json – Constants are stored within a JSON file which is read a number of times throughout the program. Images, audio, and fonts – The program reads files to get the images for the rocket and planets as well the music for the main menu and the font for the text. The server writes to files by inserting data into the database.	
	Simple user defined algorithms (eg a range of mathematical/statistical calculations)	game.py – Within the Timer class, several mathematical calculations are used to convert the time from nanoseconds into minutes, seconds, and milliseconds. game.py – Within the Rocket class, complex mathematical calculations are used to calculate the new position of the rocket as well as to rotate the rocket. These calculations include Pythagoras's theorem and trigonometry.	
	Generation of objects based on simple OOP model.	Backgroundmanager.py – As well as the complex generation of objects in main.py, the background manager generates objects containing instances of the Star class using a simple OOP model as described in the Technical Log.	
C	Linear search	LayoutManager.py – The layout manager uses linear search when deleting a textbox or button to iterate through each textbox or button until it finds the correct textbox or button where it can then remove it from the list.	

Coding Styles

As well as the models and algorithms, the style of coding I have used is an important part. Therefore, the table below explains the coding styles as well as how or where I have used them. These characteristics are taken directly from "Table 2: Coding styles".

Style	Characteristic	How it has been achieved
Excellent	Modules (subroutines) with appropriate interfaces.	As shown in the inheritance diagram and class definition diagrams, there are many classes each with interfaces for other parts of the program to interact with.
	Loosely coupled modules (subroutines) – module code interacts with other parts of the program through its interface only.	This is shown within the technical solution below where it can be seen that all interactions are through interfaces. Also shown in the inheritance diagram is what interactions happen between classes and subroutines.
	Cohesive modules (subroutines) – module code does just one thing.	By looking at the class definitions, you will be able to see that each method of every class does a specific thing.

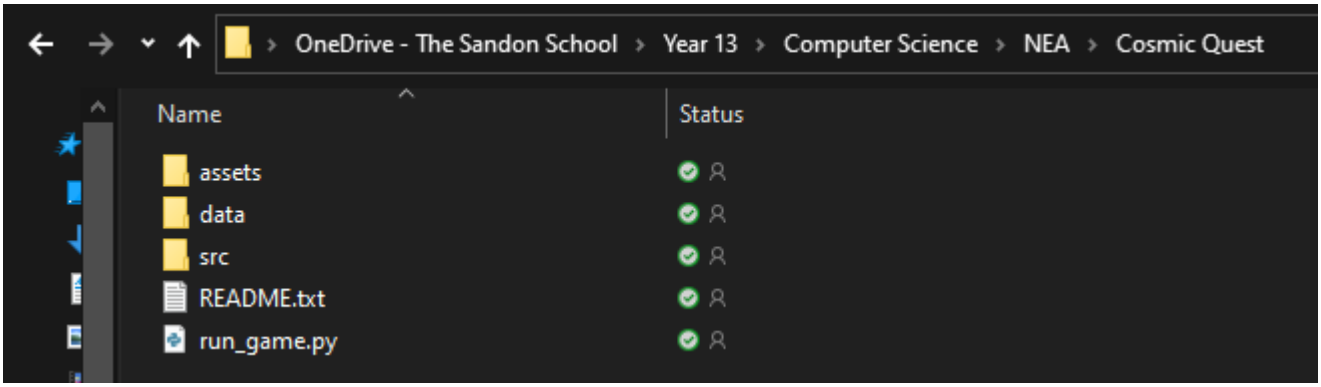
	Modules (collections of subroutines) – subroutines with common purpose grouped.	Again, by looking at the class definitions, you can see that collections of subroutines are grouped with classes.
	Defensive programming.	The most vulnerable part of the program is the login system as this is the only place where the user can input text into the program. This part of the code has been programmed to detect many different types of threats such as typing separators (##, $$, or &&) and extremely long inputs.
	Good exception handling.	Within in the server, exceptional exception handling is used as this needs to keep running even if errors occur. Therefore, exception handling is used wherever an error could occur and deals with it, so the server keeps running.
Good	Well-designed user interface	As shown in the screenshots of the program, you can see that the user interface is easy to use and from the design section, you can see it went through various designs to get to a well thought out user interface.
	Modularisation of code	This is described in the design section where it explains how the program is split into various modules (sections) to make it easier to code, debug, and run.
	Good use of local variables	As shown in the class definition diagrams and in the technical log, the vast majority of variables are local and stored with its class.
	Minimal use of global variables	There are only 2 global variables within the whole program, and these are both used to start the server.
	Managed casting of types	Type casting in used in many places throughout the code to make sure data is in the correct data type before being used.
	Use of constants	All constants are stored within a constants file so that they can easily be changed.
	Appropriate indentation	This is shown within the technical log where you can see that indentation is used correctly throughout all of the program.
	Self-documenting code	Instead of trying to compress the code into long, complicated lines, everything is spread out and self-documenting due to the use of meaningful variable names.
	Consistent style throughout	By looking at the technical log, you can see that the style of coding is very consistent and easy to interpret.
	File paths parameterised	Wherever file paths are used they are stored within the constants file which means they are parameterised allowing for them to be changed easily.
Basic	Meaningful identifier names	The technical log and class definition diagrams shows clearly that all variables have very meaningful identifier names which tell you exactly what they store.
	Annotation used effectively where required.	Where necessary the code is commented so that further explanation can be given as to what it does.

Technical Log

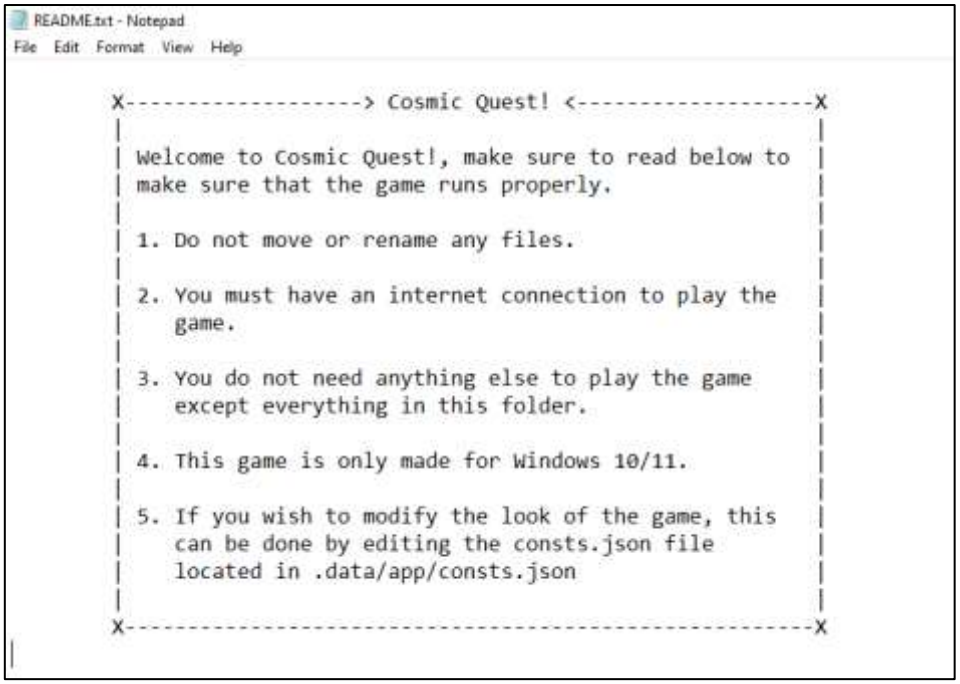
Using the sections I talked about in the part labelled “Modular Design”, I worked on an individual section at a time.

Creating folders, README.txt and run_game.py

14th August 2022: Before programming anything, I created the folders and files outlined in the “File Structure” of the Design. By creating the folders first, it means I will be able to stay organised once I start programming and making new files.



I also created the README.txt file which is a simple 'User Manual' which outlines the requirements and any other important information. This is what the file contains:



Once I created the folders, I went on to program the run_game.py and the main.py files. The sole purpose of the run_game.py file is to have an easy way to start the program without being confused as to which file to open. It is the only Python file within the root folder so cannot be mistaken for anything else.

The one thing it does do, is make sure that the file has been directly ran and not imported. This is done on like 6 which checks if __name__ is set to __main__ which it is if the file has been directly executed.

```python
# file: run_game.py

import sys
from src.main import Main

if __name__ == '__main__':
    Main()
    sys.exit()

```

Coding main.py

16th August 2022: Once I created the run_game.py file, I went onto coding the main.py file. As mentioned in the "Program Control Flowchart" section, this file is what will manage the control of the program and switch between the different sections (files).

I started by programming the creation of a function which creates the Pygame window and variables. This is what I coded:

```
1   # file: main.py
2
3   import pygame
4
5   def __InitialisePygame():
6       """Create Pygame window and CLOCK.
7       """
8
9       # Initialise pygame modules
10      pygame.init()
11      pygame.display.init()
12      pygame.font.init()
13
14      # Create pygame CLOCK
15      CLOCK = pygame.time.Clock()
16      # Create pygame window fullscreen
17      WINDOW = pygame.display.set_mode((0,0), pygame.FULLSCREEN)
18      # Change the windows caption
19      pygame.display.set_caption("Cosmic Quest!")
20      # Hide the users cursor
21      pygame.mouse.set_visible(False)
22      # Get the size of the pygame window
23      RES = pygame.display.get_window_size()
24      # Make the pygame window appear on top of everything else
25      return RES, WINDOW, CLOCK
26
```

The function returns the Pygame WINDOW object which is used to display add text and items to the window, the size of the window (RES), and the Pygame CLOCK object which is used to control the FPS of the game.

I then went on to code the main loop. The program uses the variable *GameState* to pass control to the correct section. So *GameState* starts as being set to "login" so that the user must login before starting the game. The while loop will keep running until one of the other sections exits the program. The loop will run the section and once it has finished running, it will return the next state so that the main.py file can hand over control. The code below shows how I coded this to start with:

```
26
27  def Main():
28      """Runs the program and is the main program loop.
29      """
30
31      # The first game state is login
32      GameState = "login"
33
34      # Start the main program loop
35      while True:
36
37          # If game state is to login...
38          if GameState == "login":
39
40              # ---> User logs in, returns the Users ID <---
41
42              # Once logged in, start the pygame window
43              RES, WINDOW, CLOCK = __InitialisePygame()
44              # Go to the main menu
45              GameState = "main-menu"
46
47          # If game state is the main menu...
48          elif GameState == "main-menu":
49
50              # ---> User chooses where they go next <---
51              # Returns the next state
52              GameState = "Returned value"
53
```

Once all the other sections of the program were coded, I came back to main.py to add link the different sections of code together.

```python
# file: main.py

import pygame
import win32gui
import win32con

from src.login              import Login
from src.main_menu          import MainMenu
from src.how_to_play        import howToPlay
from src.game               import Game
from src.leaderboard        import LeaderBoard

from src.modules.connection import Connection

def InitialisePygame():
    """Create Pygame window and CLOCK.
    """

    # Initialise pygame modules
    pygame.init()
    pygame.display.init()
    pygame.font.init()

    # Create pygame CLOCK
    CLOCK = pygame.time.Clock()
    # Create pygame window fullscreen
    WINDOW = pygame.display.set_mode((0,0), pygame.FULLSCREEN )
    # Change the windows caption
    pygame.display.set_caption("Cosmic Quest!")
    # Hide the users cursor
    pygame.mouse.set_visible(False)
    # Get the size of the pygame window
    RES = pygame.display.get_window_size()
    # Make the pygame window appear on top of everything else
    win32gui.SetWindowPos(pygame.display.get_wm_info()['window'], win32con.HWND_TOPMOST, 0,0,0,0,
win32con.SWP_NOMOVE | win32con.SWP_NOSIZE)
    return RES, WINDOW, CLOCK

```

The main difference is that each section must be imported from the corresponding python file. The rest of the Pygame initialisation is the same.

When testing the Pygame initialisation method, I found that the smallest screen size that the program works with is 750 x 750 pixels. This screen size is very small so it will be extremely unlikely that the program will ever get anywhere near these values.

```
37
38 def Main():
39     """Runs the program and is the main program loop.
40     """
41
42     # Gets the user to enter the IP of the server before doing anything
43     print("                                        ")
44     print("----------> Cosmic Quest! <----------")
45     print("                                        ")
46     print("     Enter the IP of the server      ")
47     print("                                        ")
48     IP = input("   -->   ")
49
50     # Create the CONNECTION to the server using the Connection module
51     CONNECTION = Connection(IP, 5050, 10)
52
53     # The first game state is login
54     GameState = "login"
55
56     # Start the main program loop
57     while True:
58
59         # If game state is to login...
60         if GameState == "login":
61             # Create Login instance
62             login = Login(CONNECTION)
63             # Run the login, and it returns the USERID
64             USERID = login.Run()
65             # Once logged in, start the pygame window
66             RES, WINDOW, CLOCK = InitialisePygame()
67             # Go to the main menu
68             GameState = "main-menu"
69
```

Now, when the Main function runs it first asks the user to enter the IP address of the server. This is needed as without this it won't be able to login or connect to the database to get questions. It then creates a connection to the server using the Connection module and this object is passed into each section of code. When the while loop starts, it first takes the user to the login section which will return the UserID of the user who logs in. It then initialises pygame and takes them to the main menu.

```
69
70          # If game state is the main menu...
71          elif GameState == "main-menu":
72              # Create instance of the main menu
73              MainMenuObject = MainMenu(RES, CONNECTION, USERID)
74              # Run the main menu and it will return the next state
75              GameState = MainMenuObject.Run(WINDOW, CLOCK)
76
77          # If game state is how to play...
78          elif GameState == "how-to-play":
79              # Create instance of how to play
80              HowToPlayObject = howToPlay(RES, CONNECTION, USERID)
81              # Run it and it will return the next game state
82              GameState = HowToPlayObject.Run(WINDOW, CLOCK)
83
84          # If game state is the game...
85          elif GameState == "game":
86              # Create instance of the game
87              GameObject = Game(RES, CONNECTION, USERID)
88              # Run the instance and game state will be returned
89              GameState = GameObject.Run(WINDOW, CLOCK)
90
91          # If game state is the leader board...
92          elif GameState == "leaderboard":
93              # The leaderboard instance will be created
94              LeaderBoardObject = LeaderBoard(RES, CONNECTION, USERID)
95              # The instance will be ran and return the next state
96              GameState = LeaderBoardObject.Run(WINDOW, CLOCK)
97
```

The other game states from then on are all the exact same, each one is given the size of the screen, the connection object, and the UserID.

Coding login.py

19th August 2022: The first section to code is the login. The main purpose of this section will be to:

- Allow the user to create a new account.
- Allow the user to log into an existing account.

Once either of these have happened, they will be able to move onto the next section, the main menu. To make this section more secure and less vulnerable to exploitation, it will be terminal-based meaning the only way the user can interact is by inputting text.

```
# file: login.py

from src.modules.connection import Connection

def Error(Message:str):
    # Display error message
    print("                                    ")
    print("------------------------------------")
    print(f"     {Message}")
    print("------------------------------------")
    print("                                    ")
    input("     Press enter to continue")

class Login:
    def __init__(self, Connection:Connection):
        """Initialise the connection and states for the login.

        Args:
            Connection (Connection): The connection object of the server.
        """

        self.__Connection = Connection
        self.__LoggedIn = False
        self.__LoggedInAs = None

```

Firstly, I made an Error function which will be used if the user inputs something that isn't in the correct format. This will simply show a message and allow them to press enter to continue.

I then made a class within the program called Login which is what will be called from the main.py file. The class has a constructor method which will create the needed variables, and the Run method which will complete the section's main purpose.

The constructor method takes in 1 attribute, which will be the Connection object to send and receive data from the server.

```
25
26      def Run(self):
27          """Get the user to log into the server or create account.
28
29          Returns:
30              UserID (int): UserID of the logged in user.
31          """
32
33          # The user must login before they can move on
34          while not self.__LoggedIn:
35              # Display the menu options
36              print("                                        ")
37              print("----------> Cosmic Quest! <----------")
38              print("                                        ")
39              print("     1.  Create an account              ")
40              print("     2.  Login                          ")
41              print("                                        ")
42              print("     Type number of option,             ")
43              print("     and press enter                    ")
44              print("                                        ")
45              Option = input("   -->   ")
46
47              # If they choose to create an account...
48              if Option == "1":
49                  self.CreateAccount()
50
51              # If they choose to login...
52              elif Option == "2":
53                  self.Login()
54
55              # If they don't choose 1 or 2...
56              else:
57                  Error("Invalid input")
58
59              # If they have now logged in or created an account...
60              if self.__LoggedIn and self.__LoggedInAs != None:
61                  return self.__LoggedInAs
62
```

The Run method starts by displaying a welcome menu with the user's options they can make and getting the user's choice.

This choice then needs to be checked against the options and then run the correct method. This while loop will only stop once the user has logged in correctly and the method returns the User ID back to main.py.

Once they have entered something it will run the corresponding function.

Once they have completed the section, they are on it will then check if they have successfully logged in, if they have the program will return the UserID to the main.py file.

```
62
63      def CreateAccount(self):
64          """Get user to create an account"""
65          print("                                        ")
66          print("----------> Create Account <----------")
67          print("                                        ")
68          print("          Enter a username              ")
69          print("                                        ")
70          Username = input("    -->    ")
71
72          # Check if username already exists in database
73          self.__Connection.Send(f"106##{Username}")
74
75          # if username is already taken or contains illegal characters...
76          if (self.__Connection.Receive() == "False") or ("##" in Username) or ("$$" in Username)
    or ("&&" in Username):
77              # Display error messsage and return to main menu
78              Error("Invalid username")
79              return
80
81          # If username is ok, get them to enter a password
82          print("                                        ")
83          print("          Enter a password              ")
84          print("                                        ")
85          Password = input("    -->    ")
86
87          # if password contains illegal characters...
88          if ("##" in Password) or ("$$" in Password) or ("&&" in Password):
89              # Display error messsage and return to main menu
90              Error("Invalid password")
91              return
92
```

```
92
93          print("                                        ")
94          print("          Enter your first name         ")
95          print("                                        ")
96          FirstName = input("    -->    ")
97
98          # if first name contains illegal characters...
99          if ("##" in FirstName) or ("$$" in FirstName) or ("&&" in FirstName):
100             # Display error messsage and return to main menu
101             Error("Invalid first name")
102             return
103
104         print("                                        ")
105         print("          Enter your last name          ")
106         print("                                        ")
107         LastName = input("    -->    ")
108
109         # if last name contains illegal characters...
110         if ("##" in LastName) or ("$$" in LastName) or ("&&" in LastName):
111             # Display error messsage and return to main menu
112             Error("Invalid last name")
113             return
114
```

```
114
115         # Once all fields have been entered and verified
116         # Add the user to the database
117         self.__Connection.Send(f"100##{Username}##{Password}##{FirstName}##{LastName}")
118         self.__Connection.Receive()
119
120         # Check login details with database
121         self.__Connection.Send(f"103##{Username}##{Password}")
122         # Response is either False or UserID
123         Response = self.__Connection.Receive()
124
125         # If the user wasn't added to the database properly...
126         if Response == "False":
127             # Display error message return to main menu
128             Error("Error occured")
129             return
130
131         # If they successfully logged in, set logged in to True
132         self.__LoggedInAs = Response
133         self.__LoggedIn = True
134
```

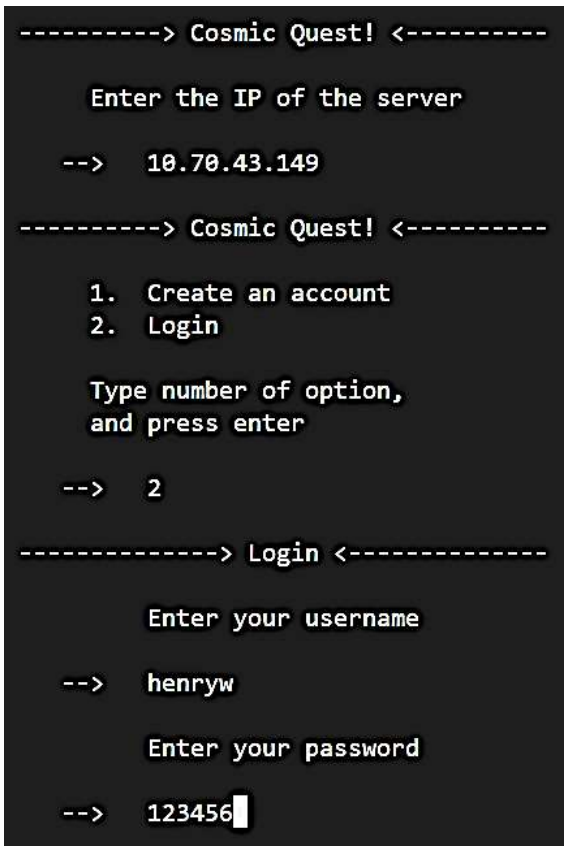
When the user selects Create Account, they will be asked to enter a username which will then be sent to the server to check if the username is already taken, if it isn't then the username will be validated for illegal characters. Next the user enters a password, first name, and last name all of which get validated and then lastly it sends all the details to the server for processing. Once it has done this it checks that the user has been created by logging into the account and getting the UserID.

```
134
135     def Login(self):
136         """Get user to login to an existing account.
137         """
138
139         print("                                        ")
140         print("----------> Create Account <----------")
141         print("                                        ")
142         print("         Enter your username            ")
143         print("                                        ")
144         Username = input("   -->   ")
145
146         # if username contains illegal characters...
147         if ("##" in Username) or ("$$" in Username) or ("&&" in Username):
148             # Display error messsage and return to main menu
149             Error("Invalid username")
150             return
151
152         print("                                        ")
153         print("         Enter your password            ")
154         print("                                        ")
155         Password = input("   -->   ")
156
157         # if password contains illegal characters...
158         if ("##" in Password) or ("$$" in Password) or ("&&" in Password):
159             # Display error messsage and return to main menu
160             Error("Invalid password")
161             return
162
163         # Check login details with database
164         self.__Connection.Send(f"103##{Username}##{Password}")
165         # Response is either False or UserID
166         Response = self.__Connection.Receive()
167
168         # If login was not correct...
169         if Response == "False":
170             # Display error message and return main menu
171             Error("Incorrect login")
172             return
173
174         # If they successfully logged in, set logged in to True
175         self.__LoggedInAs = Response
176         self.__LoggedIn = True
177
```

If the user selects Login from the menu, then they will be asked to enter a username and password which will then be validated for illegal characters and then it is sent to the server. The server will then respond with either the UserID if it logged in successfully or False if it did not.

To make sure the methods worked as intended I tested the various sections. First, I tested the menu and creating an account:

Once I had created an account, I then tested logging into the account:



Coding the modules

26th August 2022: After coding the login system, I started working on the main menu. As shown in the Design section, this would be simple and have buttons in the centre of the screen with small circles moving across the screen to represent stars.

Coding music.py

When the user is on the main menu, music will be playing. This will be playing using Pygame's mixer module. After reading the Pygame documentation for how the mixer module works, I started coding the main methods of the Music class. These were Play, Pause, Un-pause, Stop, and GetState. I then tested the class with a .mp3 file which did not work. I went back to the Pygame documentation and then found that the module has problems with .mp3 files and that the most stable format is .wav. After reading this I converted the theme song to the .wav format and tested the code again. This time it worked, and each method worked as well. The code I wrote is below:

```python
# file: music.py

from pygame import mixer

# Initialise the pygame mixer
mixer.init()

class Music:
    def __init__(self, Path:str, Volume:float, Loop:int):
        """Play a sound using pygames mixer.

        Args:
            Path (str): filepath to the audio.
            Volume (float): A float between 0.0 and 1.0
            Loop (int): How many times to repeat the audio. -1 loops forever.
        """

        self.__Path = Path
        self.__Volume = Volume
        self.__Loop = Loop
        self.__State = "loaded"

        # Load the audio and set the volume
        mixer.music.load(self.__Path)
        mixer.music.set_volume(self.__Volume)

    def Play(self):
        """Play the loaded audio.
        """

        mixer.music.play(self.__Loop)
        self.__State = "playing"

    def Pause(self):
        """Pause the audio.
        """

        mixer.music.pause()
        self.__State = "paused"

    def Unpause(self):
        """Unpause the audio.
        """

        mixer.music.unpause()
        self.__State = "playing"

    def Stop(self):
        """Stop the audio.
        """

        mixer.music.stop()
        self.__State = "stopped"

    def GetState(self):
        """Get the current state of the audio.

        Returns:
            String: Current state of the audio.
        """

        return self.__State

```

Coding BackgroundManager and Star classes

Throughout all the program the background will be as described in the design stage where small circles move across the screen to represent the stars. Each individual star is a separate instance of the Star class. Once I coded the first version of the background and tested it, I found that if I used a specific number of stars for the background, on smaller displays the stars would be very close together and on larger displays the stars would be very sparse. This was because there was a larger space to fit the same number of stars.

To make the density of stars the same for different screen sizes, I used the following formula:

$$0.4 \times ScreenWidth = NumberOfStars$$

This is implemented on line 26 of *Background.py*.

The next problem I faced was that once a star reaches the top of screen, it would still exist and continue to be updated. This led to the list of stars becoming very large which in turn slowed the program and caused the FPS to drop dramatically. To avoid this, instead of creating more and more stars, once a star has reached the top of the screen it gets reset and moved back to the bottom of the screen – this means no more stars will be created or deleted.

When the program is fading to a different section, for example going from the main menu to the leader board, as well as fading, the stars in the background accelerate and disappear once they reach the top of the screen. To achieve this, the Update methods of the *BackgroundManager* and the *Star* both require the *Fading* attribute so that they can deal with this accordingly. The *BackgroundManager*'s *Update()* method uses the *Fading* attribute to stop displaying a star if it is off the screen and *Fading* is True. This is implemented on line 56 of *Background.py*. The *Star*'s *Update()* method uses the *Fading* attribute to increase the star's velocity if *Fading* is True and to only reset the position to the bottom of the screen if *Fading* is False. This is implemented on line 103 and 110 of *Background.py*.

At the start of the file *Background.py*, there are 3 modules imported: *Pygame*, *random* and *JSON*. *Pygame* is needed as the *BackgroundManager* and *Star* classes both create *Pygame Surfaces*. The *random* module is used to generate a random position and velocity for each star. The *JSON* module is used to load the constants file which is needed for the size, velocity, and colour of each star as well as the star density (per pixel).

```
1 # file: background.py
2
3 import pygame
4 import random
5 import json
6
7 # Opens the file that contains the constants
8 with open("data\\app\\consts.json") as file:
9     consts = json.load(file)
```

The *BackgroundManager* class is instantiated in every part of the program – the main menu, the game, and the leader board. The constructor of the *BackgroundManager* class requires the resolution of the Pygame Window so that it can create the *Pygame Surface* and then go on to create the stars.

```
11 class BackgroundManager:
12     def __init__(self, Res:tuple):
13         """Used to manage background by creating and updating it.
14 
15         Args:
16             Res (Width, Height): Resolution of the window
17         """
18 
19         self.__Res = Res
20 
21         # Creates a list that will store the Star objects
22         self.__Stars = []
23 
24         # Creates the transparent pygame surface that the stars will be added to
25         self.__Surface = pygame.Surface(self.__Res, pygame.SRCALPHA, 32)
26 
27         # Create stars
28         for _ in range(0, int(consts["mm-stars-ratio"] * self.__Res[0])):
29 
30             # Random position for each star
31             TempX = random.randint(0, self.__Res[0])
32             TempY = random.randint(0, self.__Res[1])
33 
34             # Add a new star to the list of stars
35             self.__Stars.append(Star([TempX, TempY], self.__Res[1]))
```

The only method of the *BackgroundManager* class is *Update()*. This method is called each cycle of the current program loop (each frame). This means that between each frame, this method runs the *Update()* method of each star and displays it if the program isn't fading.

```
37     def Update(self, Fading:bool):
38         """Moves each star in the background and returns the surface
39 
40         Args:
41             Fading (bool): True if program is fading to a different section
42 
43         Returns:
44             Pygame Surface: Surface containing the updated stars
45         """
46 
47         # Fill the surface with the background colour
48         self.__Surface.fill(consts["mm-bg-colour"])
49 
50         # Update stars and draw them onto the background surface
51         for _, TempStar in enumerate(self.__Stars):
52             TempStar:Star
53 
54             # Update the current star and returns True if "Fading" and star is off
   screen
55             TempStarIsOffScreen = TempStar.Update(Fading)
56 
57             # If window is fading to a different section but current star is still
   visible OR isn't "Fading"...
58             if (Fading and not TempStarIsOffScreen) or (not Fading):
59 
60                     # Add the current star to the pygame surface
61                     TempStarSurface = TempStar.GetSurface()
62                     TempStarPosition = TempStar.GetPosition()
63                     self.__Surface.blit(TempStarSurface, TempStarPosition)
64         return self.__Surface
```

The *Star* class is only ever instantiated within the *BackgroundManager* class and is used to encapsulate the position, velocity, and Pygame Surface of each star. During the initialisation of a star, it requires the attributes *Position* and *MaxY*.

```
66 class Star:
67     def __init__(self, Position:list, MaxY:int):
68         """Encapsulates attributes about each star in the background
69 
70         Args:
71             Position (list): (X, Y)
72             MaxY (int): Height of the screen
73         """
74 
75         self.__Position = Position
76         self.__MaxY = MaxY
77 
78         # Get radius and range of velocities from the constants file
79         self.__Radius = consts["mm-stars-radius"]
80         TempRange = consts["mm-stars-vel-range"]
81 
82         # Generate a random velocity from the range in the constants file
83         self.__Velocity = random.randint(TempRange[0], TempRange[1])
84 
85         # Create a transparent surface the size of the star
86         self.__Surface = pygame.Surface((self.__Radius*2, self.__Radius*2),
   pygame.SRCALPHA, 32)
87 
88         # Get colour of star from the constants file
89         TempColour = consts["mm-stars-colour"]
90 
91         # Draw the star (a cirlce) to the stars surface
92         pygame.draw.circle(self.__Surface, TempColour, (0,0), self.__Radius)
```

Position is needed in the star's *Update()* method so that when the *BackgroundManager* is being updated, it can get the position of the star using its *GetPosition()* method. *MaxY* is also required so that when the star is being updated and it detects it has reached the top of the screen (when its y = 0), it knows where to reset itself to. It detects it has reached the top by checking the following condition:

$$\mathrm{Y} \leq -(2 \times \mathrm{Radius})$$

This condition is used so that the star is completely off the screen before it is reset to the bottom of the screen. This is implemented on line 107 of *Background.py*. To reset the position, it sets the stars position using this formula:

$$Y = MaxY + (2 \times Radius)$$

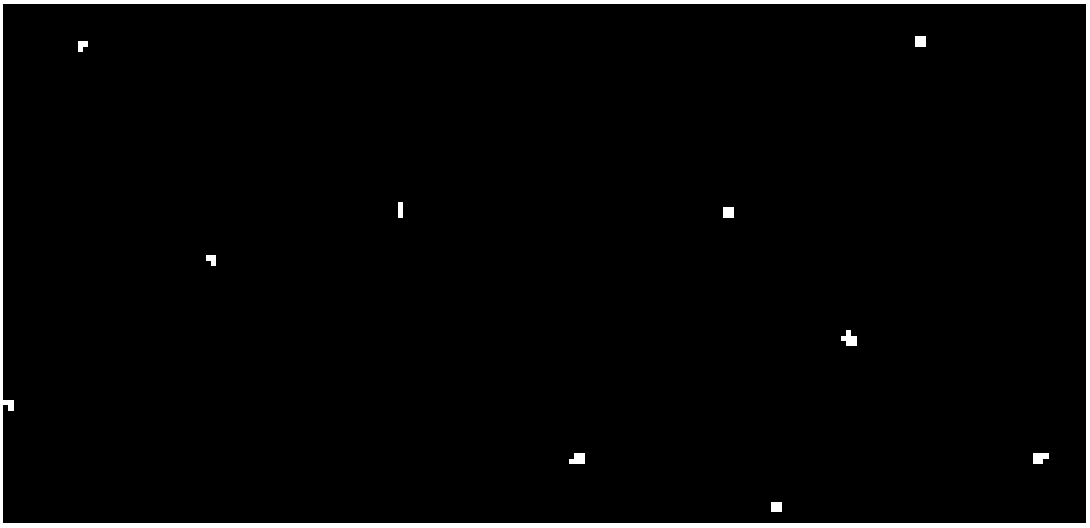
Again, the reason 2 times the radius is added on is so that the star starts from below the screen and moves onto it, preventing the star from disappearing while still visible. This is implemented on line 111 of *Background.py*. The *Update()* method the star returns True when the window is fading and the star is off the screen, otherwise it returns False.

```
 94     def Update(self, Fading:bool):
 95         """Updates the star's position
 96 
 97         Args:
 98             Fading (bool): True if program is fading to a different section
 99 
100         Returns:
101             bool: True when "Fading" and star off the screen. False when not
    "Fading" or not at top of screen.
102         """
103 
104         # If window is fading, increase the velocity
105         if Fading:
106             self.__Velocity *= 1.1
107 
108         # If star is above the top of the screen, move it to below the bottom
109         if self.__Position[1] <= -(self.__Radius * 2):
110 
111             # As long as not Fading, reset star to bottom of screen
112             if not Fading:
113                 self.__Position[1] = self.__MaxY + self.__Radius * 2
114 
115             # Returns True when Fading and if the star is off the screen
116             else:
117                 return True
118 
119         # Moves star by its velocity
120         self.__Position[1] += self.__Velocity
121         return False
```

The *GetSurface()* and *GetPosition()* methods of the *Star* class are used within the *Update()* method of the *BackgroundManager* class. When called, they simply return the star's surface or position.

```
123     def GetSurface(self):
124         """Get surface of the star
125 
126         Returns:
127             Pygame Surface: Surface containing the star
128         """
129 
130         self.__Surface:pygame.Surface
131         return self.__Surface
132 
133     def GetPosition(self):
134         """Get current position of the star
135 
136         Returns:
137             (X, Y): Centre position of star
138         """
139 
140         # Converts the position from a list to a tuple
141         return (self.__Position[0], self.__Position[1])
142 
```

By creating an instance of the *BackgroundManager* and calling its *Update()* method, the desired effect is created where white dots move at different speeds up the screen on a black background. Below is a screenshot showing a closeup of the effect:



As you can see the stars are in random positions and they move up the screen.

Coding Font class

The Font class is used extensively throughout the program and is a vital part of it. Its purpose is to render text into a pygame surface that can then be added to the window. The class will make use of the Pygame font class to render the font while my Font class will encapsulate the necessary attributes which other parts of the program will need to make use of.

```
 1 # file: font.py
 2
 3 import pygame
 4 import json
 5
 6 # Opens the file that contains the constants
 7 with open("data\\app\\consts.json") as file:
 8     consts = json.load(file)
 9
10 # Initialise the pygame font module
11 pygame.font.init()
12
```

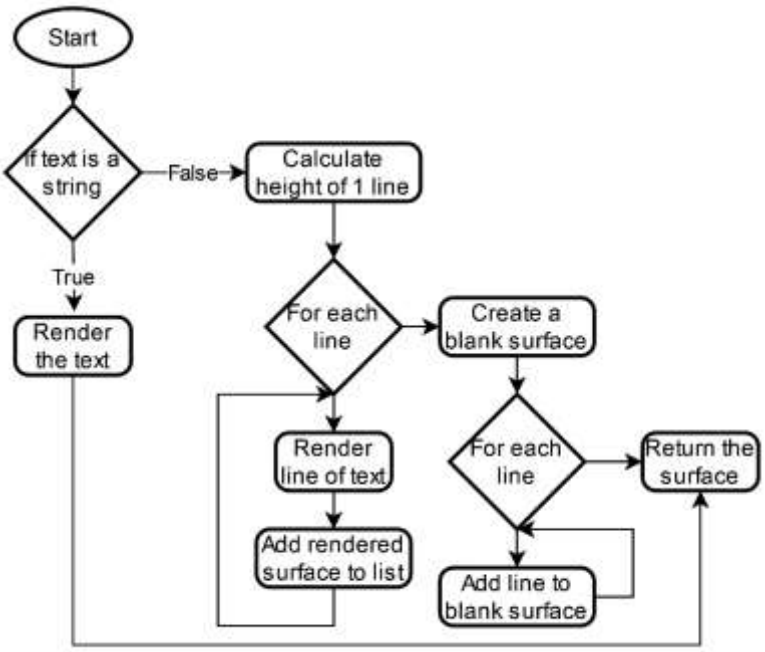
Much like all the other files, pygame and json are needed to render the text as well as open the constants file. However, unlike the other files, I must initialise the pygame font module as this isn't loaded by default.

```
12
13 class Font:
14     def __init__(self, Text, Colour:tuple, FontName:str, FontSize:int, MaxWidth:int):
15         """Create and render text in pygame.
16
17         Args:
18             Text (str OR list): Text that will be rendered. List contains each line.
19             Colour (tuple): (R, G, B) values.
20             FontName (str): Name of the font to render.
21             FontSize (int): Size of the font.
22             MaxWidth (int): Maximum width that can be rendered.
23         """
24
25         self.__Text = Text
26         self.__Colour = Colour
27         self.__MaxWidth = MaxWidth
28
29         self.__Font = pygame.font.Font(FontName, FontSize)
30
```

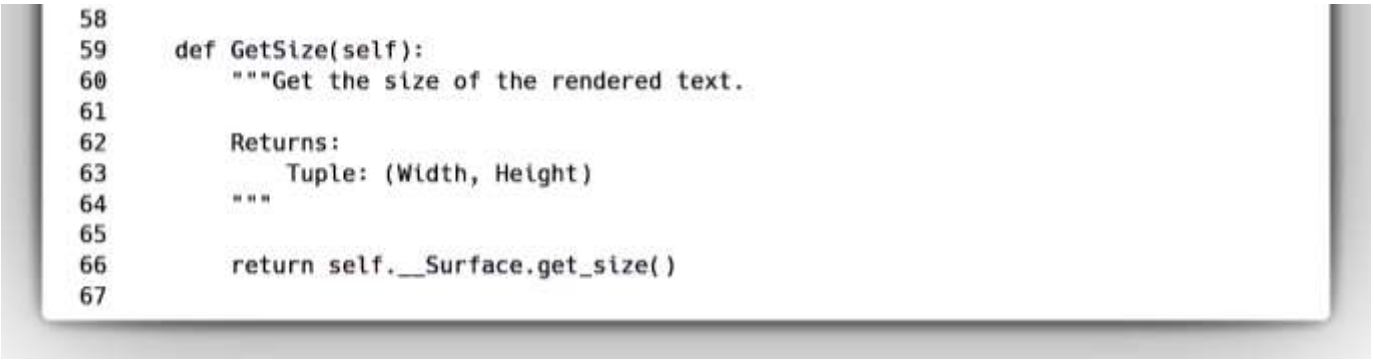
The constructor method of the Font class takes in various parameters which are required for the Font. Most of these are self-explanatory, the only one that isn't is MaxWidth. This parameter is needed when creating the blank pygame surface as the font has not been rendered at this point. Without it, the text could get cut off the right side if the surface isn't wide enough. At the end of the initialisation is it creates a pygame Font object using the FontName and FontSize.

```
30
31      def Render(self):
32          """Render the text and return surface.
33
34          Returns:
35              Pygame Surface: Surface containing the rendered text.
36          """
37
38          # If Text is only one line (a string)
39          if type(self.__Text) == str:
40              self.__Surface = self.__Font.render(self.__Text, True, self.__Colour)
41
42          # If Text is multiple lines
43          else:
44              LineSurfaces = []
45              Height = self.__Font.size(self.__Text[0])[1]
46
47              # Render each line and add to the list of surfaces
48              for line in self.__Text:
49                  LineSurfaces.append(self.__Font.render(line, True, self.__Colour))
50
51              # Create a blank, transparent surface
52              self.__Surface = pygame.Surface((self.__MaxWidth,
    Height*1.1*len(self.__Text)), pygame.SRCALPHA, 32)
53
54              # Add each rendered line, to the new surface
55              for i, lineSurface in enumerate(LineSurfaces):
56                  self.__Surface.blit(lineSurface, (0,(Height*1.1)*i))
57          return self.__Surface
58
```

The main function of the Font class is the Render method. This is what adds the text to the blank pygame surface and in the correct position. To explain how this function works better, I made the following flowchart:



As you can see, it uses 2 for loops. The reason 2 are needed is because the total height of the combined lines is unknown until all the text is rendered. On line 56 there is a constant of 1.1 which is multiplied by Height and the line number. The 1.1 is needed to make sure there is a gap between each line.



```
58
59      def GetSize(self):
60          """Get the size of the rendered text.
61
62          Returns:
63              Tuple: (Width, Height)
64          """
65
66          return self.__Surface.get_size()
67
```

The other method of the Font class is GetSize which as it says, returns the size of the rendered text.

Coding TextBox class

Much like the background, textboxes will be used extensively in my program. Each textbox will be made up of the text and a box which will outline and contain the text. Every textbox will have the same colour and layout to provide a consistent GUI.

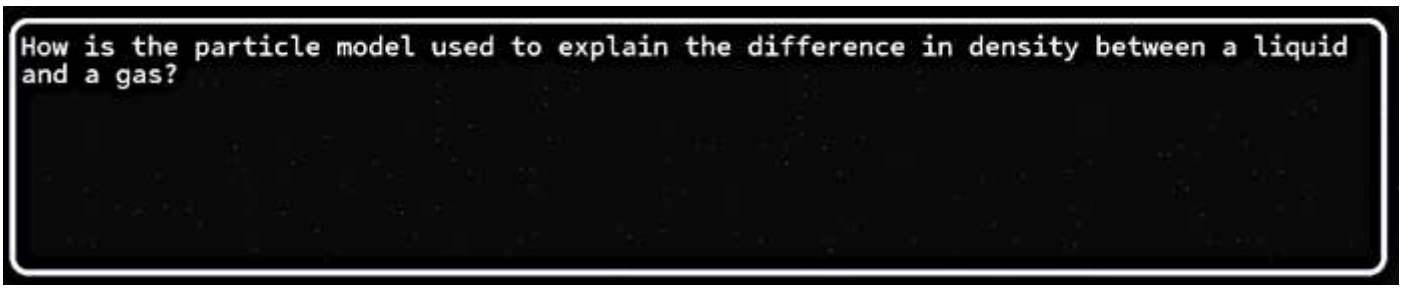
To start, the TextBox class will need several parameters. The ID will be a string which is used to identify what the textbox stores to either update it or delete it if needed. The position will be a tuple containing the x and y positions of the textbox. Much like position, size will contain the pixel size of the textbox within a tuple. Text will either be a list of strings, or a single string. This is because the textbox can have multiple lines of text if required but can also be just a single line. FontSize is an integer which is used to set the size of the text. After making the TextBox class and while working on the How to Play section, I decided that it would look better if I could align the text to the centre of the textbox. Therefore, the Alignment parameter is a string which is either ‘L’ or ‘C’, used to align the text to the left of the textbox or to align it in the centre.

When a TextBox is instantiated, it sets its attributes and then calls it’s render method.

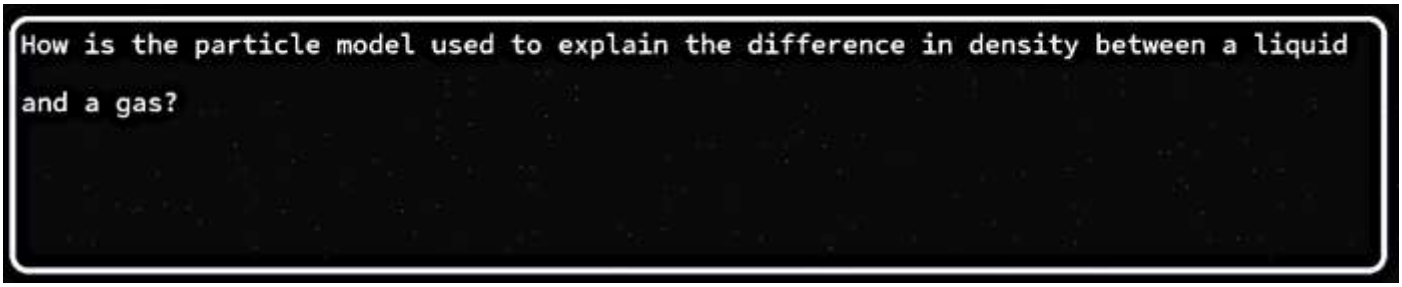
```python
# file: textbox.py

import pygame
import json

from src.modules.font import Font

# Opens the file that contains the constants
with open("data\\app\\consts.json") as file:
    consts = json.load(file)

class TextBox:
    def __init__(self, ID:str, Position:tuple, Size:tuple, Text, FontSize:int, Alignment = "C"):
        """
        A textbox will render 1 or more lines of text with an outline around it.

        Args:
            ID (str): Identifies the textbox
            Position (tuple): (X, Y)
            Size (tuple): (Width, Height)
            Text (str OR list): Text that will be rendered
            FontSize (int): Size of the font in the textbox
        """

        self._ID = ID
        self._Position = Position
        self._Size = Size
        self._Text = Text
        self._FontSize = FontSize
        self._Alignment = Alignment

        self._TextColour = tuple(consts["text-colour"])
        self._BoxColour = tuple(consts["textbox-outline-colour"])

        self._Rect = pygame.Rect(self._Position[0], self._Position[1], self._Size[0],
self._Size[1])

        # If Text is a string (1 line), it'll be turned into a list with 1 item
        if type(self._Text) == str:
            self._Text = [self._Text]

        self.__Render()

```

The Textboxes render method is used to turn the text into a pygame surface. It iterates over each line of text in the list of text and then: First gets the rendered text, then calculates it's position in the textbox, and then adds the text into the Textbox surface.

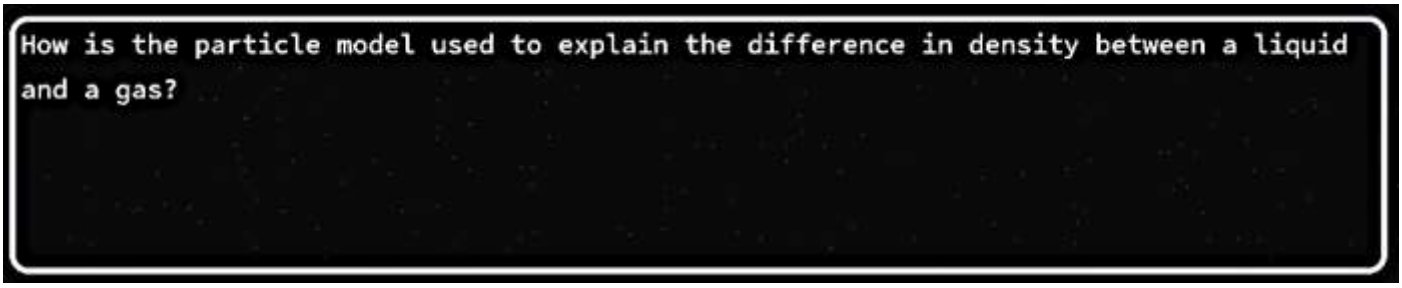
To calculate the position of the text if the text is being left justified, the x position is a set distance from the edge so that the outline isn't too close. For the y position, it takes the lines number multiplies it by 12 and adds on the Font Size multiplied by 1.3 and the line number. This is needed because the position is based off where the above line finishes but also that as the font gets bigger, the gap between lines should as well. The number 1.3 was decided after testing many times with different values, it gives the most natural and simple gap to read. Below shows the textbox with no gap:



This is the textbox with a gap of 1.8:



Then finally this was the textbox with a gap of 1.3 and this is how it looks now:



If the text is being centre justified, it is a bit different for the x value. To get the x position, it takes the centre of the textbox's x position and subtracts half the width of the line of text. This means that each line has a different x position so that they all line up in the centre.

After adding all the text, it then draws the outline which a rectangle with rounded corners.

```
42
43      def __Render(self):
44          # Create a transparent pygame surface
45          self._Surface = pygame.Surface(self._Size, pygame.SRCALPHA, 32)
46
47          # For each line of text...
48          for i, Line in enumerate(self._Text):
49              # Create a Font object and render it to get the surface
50              font = Font(Line, self._TextColour, consts["pth-font-regular"], self._FontSize,
    self._Size[0]).Render()
51              # Get size of the rendered line
52              RenderedFontSize = font.get_size()
53
54              # If alignment is set to the Left
55              if self._Alignment == "L":
56                  # Set the position of the line to the left side of the surface
57                  pos = (12, 12*(i+1) + self._FontSize*1.3*i)
58
59              # If alignment is set to the Centre
60              elif self._Alignment == "C":
61                  # Set position of the line to the centre of surface
62                  pos = (self._Size[0]/2 - RenderedFontSize[0]/2, 12*(i+1) + self._FontSize*1.3*i)
63
64              # Add the line to the surface at the calculated position
65              self._Surface.blit(font, pos)
66
67          # Draw outline around the textbox
68          TempRect = (0, 0, self._Size[0], self._Size[1])
69          OutlineWidth = consts["textbox-outline-width"]
70          Radius = consts["textbox-radius"]
71          pygame.draw.rect(self._Surface, self._BoxColour, TempRect, OutlineWidth, Radius)
72
```

Once the textbox has been created it will need to be accessed other times to check its position or to delete it. Therefore, there are several public methods for the Textbox.

```
72
73      def GetSurface(self):
74          """Get the pygame surface of the textbox.
75
76          Returns:
77              Pygame Surface: Surface containing the textbox.
78          """
79
80          return self._Surface
81
82      def GetRect(self):
83          """Get the rect (x, y, width, height) of the textbox.
84
85          Returns:
86              Pygame Rect: Object containing the position and size of textbox.
87          """
88
89          return self._Rect
90
91      def GetPos(self):
92          """Get position of the textbox.
93
94          Returns:
95              tuple: (X, Y) of the top left of the textbox.
96          """
97
98          return self._Position
99
100     def GetID(self):
101         """Get the ID of the textbox.
102
103         Returns:
104             string: ID of the textbox.
105         """
106
107         return self._ID
108
```

These public methods simply return different attributes of the textbox.

Some attributes in the class are set from the constants file which is imported at the start. This is so that I can quickly change adjust variables in one place and I don't need to change it in many places in the code. This was especially useful when working out the best values for the colour, font size, radius of round corners and thickness of outline.

As seen throughout the textbox class, every attribute is protected instead of private. This is because the textbox class will be inherited when creating the button class next.

Coding Button class

Following on the from the TextBox class, I made the Button class. Like textboxes, buttons are also an essential part of the program. They will follow the same consistent look as the textboxes and this will be achieved by inheriting the TextBox class. The only difference will be is that the Button's outline will change colour when the user hovers the cursor over the button, and it will also be able to be clicked.

```python
# file: button.py

import pygame
import json

from src.modules.textbox import TextBox

# Opens the file that contains the constants
with open("data\\app\\consts.json") as file:
    consts = json.load(file)

class Button(TextBox):
    def __init__(self, ID:str, Position:tuple, Size:tuple, Text, FontSize:int, Allignment = "C"):
        """A button will render a textbox which can change if the mouse if hovering over it and detect being clicked.

        Args:
            ID (str): Identifies the button
            Position (tuple): (X, Y)
            Size (tuple): (Width, Height)
            Text (str OR list): Contains text of the button
            FontSize (int): Size of text
        """

        # This will create self.surface by running the TextBox's init method.
        super().__init__(ID, Position, Size, Text, FontSize, Allignment)

        # This will create the hovered surface
        self.__RenderHover()

```

The code for the Button class begins by inhering the TextBox class and then taking in the same parameters as the TextBox class. It then uses the super() function to initialise the attributes within the init of the TextBox class. In doing this, it also renders the textbox. Next it calls the RenderHover method which is coded below:

```python

    def __RenderHover(self):
        """Render the buttons hovered surface.
        """

        # Create a copy of the textbox surface
        self._HoveredSurface = self._Surface.copy()

        # Turn the outline to the hovered colour as well as slightly thicker outline
        HoverColour = consts["button-hover-outline-colour"]
        TempRect = (0, 0, self._Rect[2], self._Rect[3])
        OutlineWidth = consts["button-outline-width"]
        Radius = consts["textbox-radius"]

        # Draw the outline to the hovered surface
        pygame.draw.rect(self._HoveredSurface, HoverColour, TempRect, OutlineWidth, Radius)

```

This method creates a new pygame surface which is a copy of the original surface, but the outline is a different colour. This is so that the user knows that the button is clickable.

```
45
46      def GetSuface(self, Hovered:bool):
47          """Get the surface of the button depending on whether the mouse is hovering over it.
48
49          Args:
50              Hovered (bool): True if mouse is hovering False if not.
51
52          Returns:
53              Pygame Surface: Surface containing the button
54          """
55
56          # If mouse is hovering then return the hovered surface
57          if Hovered:
58              return self._HoveredSurface
59
60          # If mouse is not hovering, return the normal surface
61          else:
62              return self._Surface
63
64      def DetectHover(self, MousePosition:tuple):
65          """Detect if the mouse is hovering over the button.
66
67          Args:
68              MousePosition (tuple): (X, Y) of the mouse.
69
70          Returns:
71              bool: True if mouse is hovering over button, False if not.
72          """
73
74          # Checks if the mouse is over the button using pygames collision checker
75          return pygame.Rect.collidepoint(self._Rect, MousePosition[0], MousePosition[1])
76
```

The GetSurface method is already present in the TextBox class so when it is defined in the Button class, the method is overwritten. I did this because instead of just returned the normal pygame surface of the button, I want it to return a different surface if the user's cursor is hovering over the button. Therefore, it takes in a variable that is used to return the correct pygame surface.

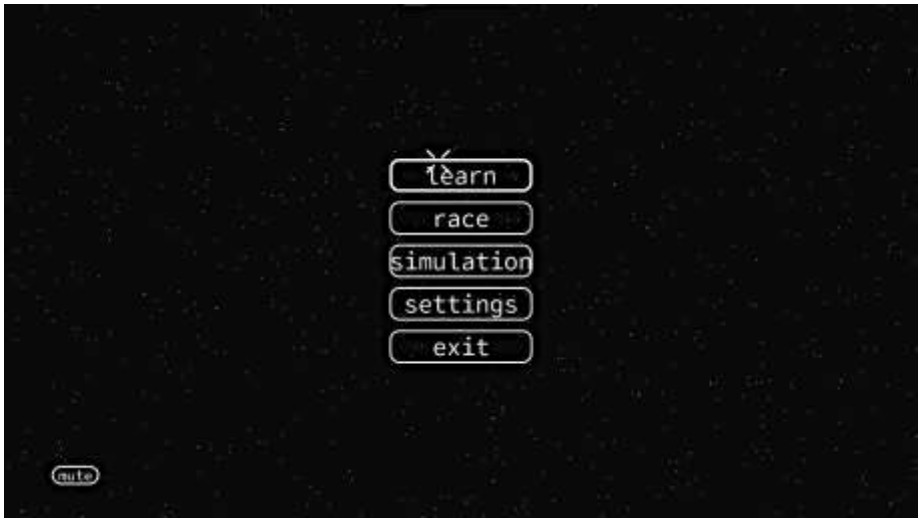
The DetectHover method is used to check if the user's mouse is colliding with the button. Originally, I coded this function myself by comparing the x and y positions of the mouse with each edge of the button and this worked perfectly. The code for this is below:

```
77          XCondition = MousePosition[0] > self._Position[0] and MousePosition[0] < self._Position[0] + self._Size[0]
78          YCondition = MousePosition[1] > self._Position[1] and MousePosition[1] < self._Position[1] + self._Size[1]
79
80          if XCondition and YCondition:
81              return True
82          else:
83              return False
```

This code worked perfectly to the pixel however by timing how long this function took compared to the built-in pygame function, it was slower. Therefore, I took the decision to switch to using the built-in function as every millisecond counts when it is being ran every frame.

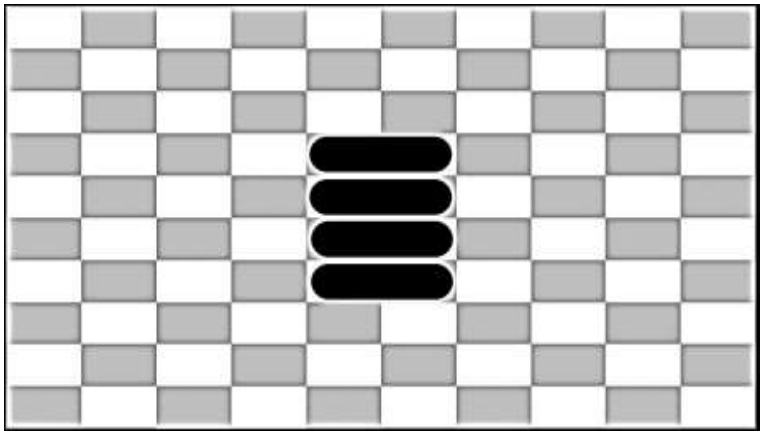
Coding LayoutManager class

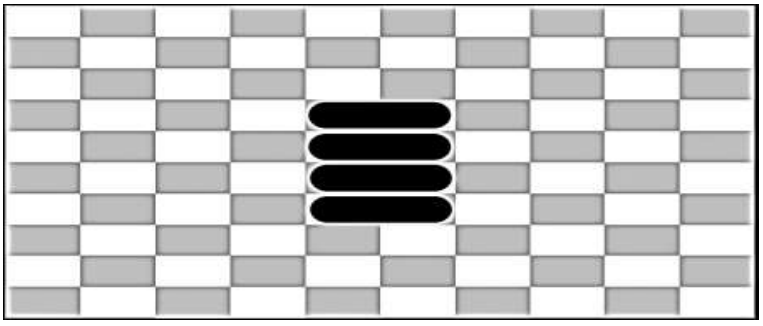
This class wasn't originally planned until I had made the first version of the main menu. The reason I made this class was to make it a lot simpler to add textboxes and buttons to the screen as well as making them line up in the correct positions for any size of screen. Before making this class, the main menu looked like this:

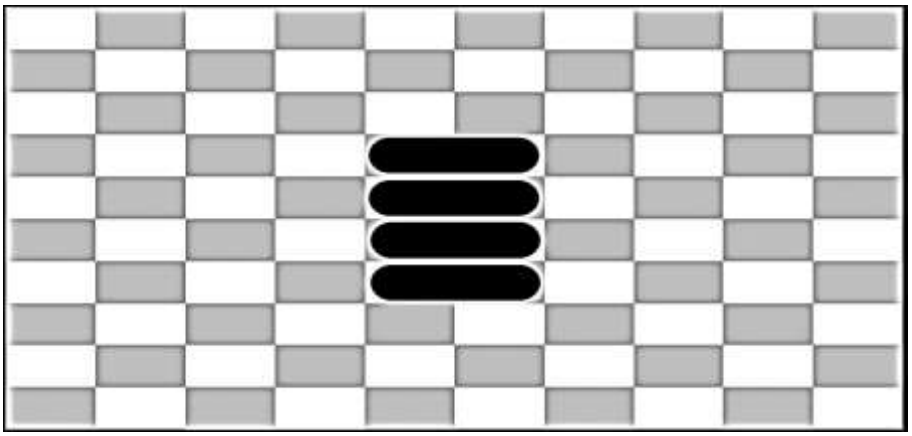


This is how it looked on my computer but if the size of the monitor is different from mine, then the buttons are no longer in the centre of the screen:

The LayoutManager class will remove this problem by basing everything off a grid system. This will split the screen into a grid which can be used to place buttons and textboxes in certain positions which change depending on the screen size.







As you can see from the 3 images above, using the grid system means the buttons can change size as well as being in the correct position.

```python
# file: layoutmanager.py

import pygame

from src.modules.textbox import TextBox
from src.modules.button import Button

class LayoutManager:
    def __init__(self, Res:tuple, GridCountSize:tuple, FontSize:int):
        """Create a layout manager which manages textboxes and buttons.

        Args:
            Res (tuple): (Width, Height)
            GridCountSize (tuple): (WidthCount, HeightCount)
            FontSize (int): FontSize for a 1920x1080 screen
        """

        self.__Res = Res
        self.__GridCountSize = GridCountSize

        # Calculate actual FontSize based off of size for a screen with width 1920
        self.__FontSize = round(FontSize*self.__Res[0]/1920)

        TempWidth = round(self.__Res[0] / self.__GridCountSize[0])
        TempHeight = round(self.__Res[1] / self.__GridCountSize[1])
        self.__GridSquareSize = (TempWidth, TempHeight)

        self.__Textboxes = []
        self.__Buttons = []

```

The LayoutManager class will make use of the Button and TextBox classes so these are imported at the top. The constructor method takes in several parameters. It takes in the resolution of the screen which is needed to work out the exact absolute position of the textboxes and buttons. GridCountSize is the number of rows and columns in the grid and FontSize is the size of the font used for the buttons and textboxes. This FontSize has to change depending

on the size of the screen so when being passed in, it is the size the text should be when displayed on a 1920 x 1080p screen. This is why on line 22 it then multiplies the FontSize by the ratio of ssssssscreen width to 1920.

Line 24, 25, and 26 calculates the size of each rectangle in the grid which will be needed multiple times.

It ends by then creating 2 lists which will store all the textboxes and the other will store all the buttons. The reason these need to be separate is because the buttons will need to be checked for if they have been clicked.

```
30
31      def __CovertGridPositionToPosition(self, GridPosition:tuple):
32          """Convert a position in the grid into a position on the screen.
33
34          Args:
35              GridPosition (tuple): (GridX, GridY)
36
37          Returns:
38              Position (tuple): (X, Y)
39          """
40
41          # Calculate the x and y positions
42          TempX = GridPosition[0] * self.__GridSquareSize[0]
43          TempY = GridPosition[1] * self.__GridSquareSize[1]
44          return (TempX,TempY)
45
```

The only private method of LayoutManager converts a position in the grid to an absolute position on the screen. This will be very important when making the textboxes and buttons as these classes only use absolute positions. It calculates the x and y values by multiplying the grid position but the size of a single grid rectangle.

```
45
46      def AddTextBox(self, ID:str, GridPosition:tuple, GridSize:tuple, Text, Allignment = "C"):
47          """Add a textbox to the layout manager.
48
49          Args:
50              ID (str): Identifies the textbox
51              GridPosition (tuple): (GridX, GridY)
52              GridSize (tuple): (GridWidth, GridHeight)
53              Text (str OR list): Text that will be rendered
54          """
55
56          # Create the textbox
57          TempPosition = self.__CovertGridPositionToPosition(GridPosition)
58          TempSize = self.__CovertGridPositionToPosition(GridSize)
59          TempTextBox = TextBox(ID, TempPosition, TempSize, Text, self.__FontSize, Allignment)
60          # Add textbox object to the list of textboxes
61          self.__Textboxes.append(TempTextBox)
62
63      def AddButton(self, ID:str, GridPosition:tuple, GridSize:tuple, Text, Allignment = "C"):
64          """Add a button to the layout manager.
65
66          Args:
67              ID (str): Identifies the button
68              GridPosition (tuple): (GridX, GridY)
69              GridSize (tuple): (GridWidth, GridHeight)
70              Text (str OR list): Text that will be rendered
71          """
72
73          # Create the button
74          TempPosition = self.__CovertGridPositionToPosition(GridPosition)
75          TempSize = self.__CovertGridPositionToPosition(GridSize)
76          TempButton = Button(ID, TempPosition, TempSize, Text, self.__FontSize, Allignment)
77          # Add button object to the list of buttons
78          self.__Buttons.append(TempButton)
79
```

The interface of the LayoutManager will be very simple and the 2 most common methods will be AddButton and AddTextBox. Each of these take the same parameters and these are all self-explanatory. Both methods are almost identical as they both calculate the absolute positions and sizes and then creates the corresponding object, finally then adding the new object to the corresponding list.

```
79
80      def RemoveTextBox(self, ID:str):
81          """Remove a textbox from the layout manager using its ID.
82
83          Args:
84              ID (str): ID of the textbox to remove.
85
86          Returns:
87              bool: True if textbox removed. False if textbox not found.
88          """
89
90          TextBoxIndex = None
91          # For each textbox...
92          for i, TempTextBox in enumerate(self.__Textboxes):
93              TempTextBox:TextBox
94
95              # If it's the textbox we are looking for
96              if TempTextBox.GetID() == ID:
97                  # Set TextBoxIndex to index of correct textbox
98                  TextBoxIndex = i
99                  # Stop the for loop
100                 break
101
102         # If TextBox is not found...
103         if TextBoxIndex == None:
104             return False
105
106         # If TextBox is found...
107         else:
108             # Delete the the textbox
109             del self.__Textboxes[TextBoxIndex]
110             return True
111
```

```
111
112     def RemoveButton(self, ID:str):
113         """Remove a button from the layout manager using its ID.
114
115         Args:
116             ID (str): ID of the button to remove.
117
118         Returns:
119             bool: True if button removed. False if button not found.
120         """
121
122         ButtonIndex = None
123         # For each button...
124         for i, TempButton in enumerate(self.__Buttons):
125             TempButton:Button
126
127             # If it's the button we are looking for...
128             if TempButton.GetID() == ID:
129                 # Set button index to current index
130                 ButtonIndex = i
131                 # Break the for loop as it's been found
132                 break
133
134         # If button isn't found, return False
135         if ButtonIndex == None:
136             return False
137
138         # If button is found...
139         else:
140             # Delete the button from the list
141             del self.__Buttons[ButtonIndex]
142             return True
143
```

As well as adding textboxes and buttons, if they need to be updated or removed, they will be deleted as this is much more efficient that attempting to change an already rendered textbox or button. These 2 methods are again almost identical and are quite simple. They first perform a linear search of the list of objects and checks to see if the target

ID is the same as the current ID. If they match, then it breaks from the list and deletes the object then returning True. If the object isn't found, then it returns False.

```
143
144     def CheckMouseCollision(self, MousePosition:tuple):
145         """Check if the mouse is over any of the buttons.
146
147         Args:
148             MousePosition (tuple): (X, Y) Position of the mouse.
149
150         Returns:
151             string: Button ID of button the mouse is over.
152             None: If mouse is not over any of the buttons.
153         """
154
155         # For each button...
156         for TempButton in self.__Buttons:
157             TempButton:Button
158
159             # If the mouse is over the current button...
160             if TempButton.DetectHover(MousePosition):
161                 return TempButton.GetID()
162
163         # If the mouse isn't over any of the buttons return None
164         return None
165
```

To make the LayoutManager easier to use, instead of the interface only checking if the mouse is colliding with a certain button, it checks every button in one function. It takes in the position of the mouse and then uses the buttons DetectHover method to check if the mouse is colliding with any of the buttons. The method will either return the ID of the button which is being hovered over or None.

```
165
166     def GetSurface(self, MousePosition:tuple):
167         """Get the layout surface including all the textboxes and buttons.
168
169         Args:
170             MousePosition (tuple): (X, Y) Position of the mouse.
171
172         Returns:
173             Pygame Surface: Surface containing all textboxes and buttons.
174         """
175
176         # Create a transparent pygame surface
177         Surface = pygame.Surface(self.__Res, pygame.SRCALPHA, 32)
178
179         # Render TextBoxes
180         for TempTextBox in self.__Textboxes:
181             TempTextBox:TextBox
182
183             # Get surface of current textbox
184             TempSurface = TempTextBox.GetSurface()
185             # Add textbox to the surface at its position
186             Surface.blit(TempSurface, TempTextBox.GetPos())
187
188         # Render Buttons
189         for TempButton in self.__Buttons:
190             TempButton:Button
191
192             # Detect if the mouse is hovering over the button
193             TempHovered = TempButton.DetectHover(MousePosition)
194             # Get the surface depending on whether or not the mouse is over button
195             TempSurface = TempButton.GetSuface(TempHovered)
196             # Add button to the surface at its position
197             Surface.blit(TempSurface, TempButton.GetPos())
198         return Surface
199
```

The last method the LayoutManager is GetSurface which adds all the textboxes and buttons to a pygame surface while also outlining any buttons that the cursor is hovering over. After creating a blank pygame surface it loops through every textbox and adds it to the surface. Once all the textboxes have been added it loops through the buttons first checking if the mouse is over the button, then getting the corresponding surface, then finally adding it to the surface. Lastly it returns the pygame surface.

Coding Cursor class

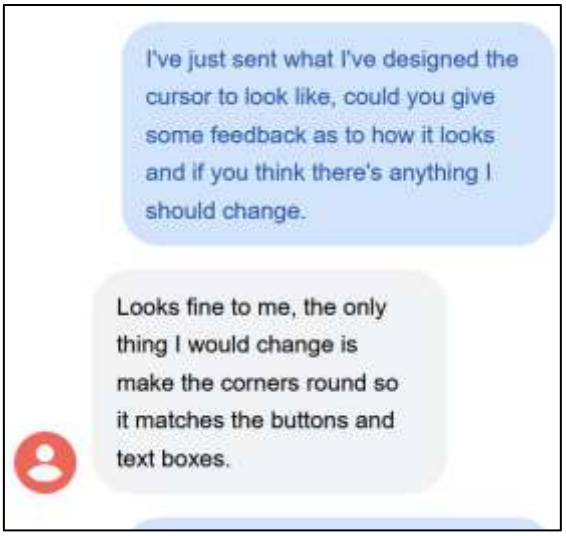
Having looked and played various space-themed games as part of my research, I had come up with a good idea for the design of the cursor. The design of the cursor is very important as it is what the user uses to interact with program and if it is poorly designed, they may struggle to play the game.

Before coding the cursor, I decided to make the cursor using PowerPoint as this allows for high quality exports, it's quick and easy to use, as well as being copyright free because I made it myself. The first design for the cursor looked like this:

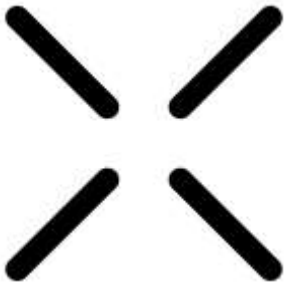
When I tested this design in the program, it was rather thin and difficult to see. This is not good as this program is being designed for a large range of people and should be accessible to as many people as possible. The second design for the cursor was this:

As you can see, it appears "bolder". This is better as when testing it, it was a lot clearer and easy to see. While testing I also sometimes struggled to see what was under the cursor. This led me to the third iteration:

For this iteration, there is a larger gap in the centre which allows the user to see more of what is under the cursor. After testing this design, I was happy with it but I went to my client to get feedback on it. Below is the message exchange between me and my client about the design of the cursor:



I agree with his feedback that by rounding the corners it would make the design more consistent with the rest of what I've made so far. Therefore, the final cursor design is this:



Now I had the design for the cursor, I started coding it. Below is the code for the Cursor class:

```
1 # file: cursor.py
2
3 import pygame
4 import json
5
6 # Opens the file that contains the constants
7 with open("data\\app\\consts.json") as file:
8     consts = json.load(file)
9
```

Firstly, it opens the constants file which will be required.

```
 9
10 class Cursor:
11     def __init__(self):
12         """Creates a cursor object.
13         """
14
15         self.__Size = consts["mm-cursor-size"]
16         self.__Image = pygame.image.load(consts["pth-mm-cursor"])
17         self.__Image = pygame.transform.smoothscale(self.__Image, (self.__Size,
   self.__Size))
18
```

In the constructor method of the Cursor class, it creates the size attribute and the image attribute by loading the cursor image and then scaling it to the size previously retrieved from the constants file.

```
18
19      def GetPosition(self, MousePosition:tuple):
20          """Get the position of the top left corner of the cursor.
21
22          Args:
23              MousePosition (tuple): the x and y position of the mouse
24
25          Returns:
26              tuple: (X, Y) of top left corner of cursor
27          """
28
29          # Calculate the top left of cursor based off of the Mouse position which is
    the centre
30          TempX = MousePosition[0] - 0.5*self.__Size
31          TempY = MousePosition[1] - 0.5*self.__Size
32          return (TempX, TempY)
33
```

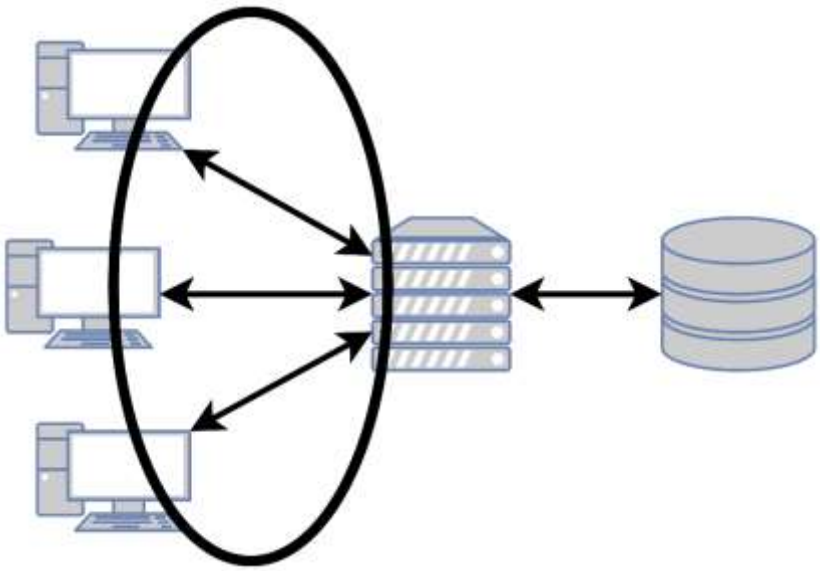
The first method of the cursor is GetPosition, originally, this method just returned the mouse position. This led to the centre of the cursor image not actually being where the program thought the cursor was. This is because pygame draws images from the top left corner, so I amended this method is used to get the position of the top left corner of the cursor. To get this, it must subtract half the cursor's size from the current mouse position.

```
33
34      def GetSurface(self):
35          """Get the pygame surface of the cursor.
36
37          Returns:
38              Pygame Surface: Surface containing image of cursor.
39          """
40
41          return self.__Image
42
```

The last method of the cursor is GetSurface which is required when adding the cursor to the window and returns the image of the cursor.

Coding Connection class

Although I won't be coding the Server yet, I am coding the connection which will be used to send data between the client and the server.



In the client-server diagram above the connection class will provide the interface for the program to send and receive data to and from the server.

```
1  # file: connection.py
2
3  import socket
4
5  class Connection:
6      def __init__(self, IP:str, PORT:int, HEADERLENGTH:int):
7          """Create a socket connection to the server.
8
9          Args:
10             IP (str): The IP address to create the connection to.
11             PORT (int): The PORT to create the connection over.
12             HEADERLENGTH (int): The length of the header.
13         """
14
15         self.__IP = IP
16         self.__PORT = PORT
17         self.__HEADERLENGTH = HEADERLENGTH
18
19         self.CreateConnection()
20
21     def CreateConnection(self):
22         """Create the socket the IP and PORT specified.
23         """
24
25         # Create the socket
26         self.__ClientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
27         # Connect the ip and port to the socket
28         self.__ClientSocket.connect((self.__IP, self.__PORT))
29
```

Firstly, the Connection class will take in the IP to connect to – this will be the IP of the server. The port to connect to will also be passed in and this will always be set to 5050. The Last parameter is the header length which I will explain below when it is used.

The CreateConnection method creates a socket object which is used to establish a network connection between two different devices over the internet. The AF_INET parameter specifies that this is an Internet Protocol v4 address socket, and SOCK_STREAM specifies that this is a TCP socket, which provides a reliable, stream-based connection. Finally, on line 28 it connects the socket to the server by using the specified IP address and port number.

```
29
30     def Send(self, Data:str):
31         """Send a string over the socket.
32
33         Args:
34             Data (str): The data to be sent.
35         """
36
37         try:
38             # Encode the data into binary
39             ByteData = Data.encode('utf-8')
40             # Generate the binary header by calculating length of data
41             ByteHeader = f"{len(ByteData):<{self.__HEADERLENGTH}}".encode('utf-8')
42             # Send the header and the data over the socket
43             self.__ClientSocket.send(ByteHeader + ByteData)
44         except:
45             print("[SendError] - Socket closed")
46             self.__ClientSocket.close()
47
```

To send data over the socket connection it must be converted into binary however when the data is received, it needs to know how many bits of data to read. Therefore, the length of the data must also be sent with the data so that the server knows how many bits to read.

This method is rather complex so I will break it down line-by-line:

Line 39: First the data which is being sent is converted into binary. Specifically, it converts the string into the UTF-8 encoding format.

Line 41: Now we need to make the header which will tell the receiver how long the data is. This header MUST be the same length all the time so that the binary can be read correctly. This is why we have the header length which says how many characters long the header is. This line of code uses Python f-strings to 1. Get the length of the data. 2. Fill the string with blank space until it's the length of header length. 3. Convert the resulting string into binary using the same function as with the data.

Line 43: Lastly it combines the header and the data (in binary) and then sends it all over the socket.

As an example, if we call the Send function with Data being "Hello World!", then the following will happen when this is executed:

1. Converts "Hello World!" to binary → 01001000 01100101 01101100 …
2. Get length of the binary → 96
3. Fill that number to take up the header length (10) → "96••••••••" (• represents a space)
4. Convert the header into binary →00111001 00110110 00100000 …
5. Combine the header with the data and send over the socket.

In the example above, the full data received is: "96••••••••Hello World!" (• represents a space). The server reads the first 10 characters to work out how long the data is, and then reads that number of bits from the socket. If this wasn't done, then multiple data packets could be read at the same time if they were sent at the same time.

Within the code, exception handling is used because if socket has been closed, the send function will raise an error. Instead of crashing and closing the program, we'll close the socket from the client-side so that it doesn't raise any more errors.

```
47
48      def Receive(self):
49          """Receives data from the socket.
50
51          Returns:
52              False: Connection has been closed
53              str: Data which has been sent from socket
54          """
55
56          try:
57              # Get length of the data in binary
58              DataHeader = self.__ClientSocket.recv(self.__HEADERLENGTH)
59              # If received no data, server closed the connection
60              if not len(DataHeader):
61                  print("[RecieveError] - Socket closed")
62                  self.__ClientSocket.close()
63                  return False
64
65              # Convert data length from binary into decimal
66              DataLength = int(DataHeader.decode('utf-8').strip())
67              # Receive data and convert from binary into str
68              Data = self.__ClientSocket.recv(DataLength).decode('utf-8')
69              return Data
70
71          except:
72              # If error while receiving data then close socket
73              print("[RecieveError] - Socket closed")
74              self.__ClientSocket.close()
75              return False
76
77      def EndConnection(self):
78          """Close the socket.
79          """
80
81          self.Send("CloseSocket")
82          self.__ClientSocket.close()
83
```

Once a request has been sent to the server and it has sent back a response, the Receive method will be used to read this data and decode it. It starts by reading the first HeaderLength of characters from the socket. If this is then None or False, it means it hasn't received any data because the socket has been closed therefore it then closes the socket client-side and returns False to signify that the data couldn't be received.

If the header is read correctly, it will then decode the binary into a string, it will remove the leading and trailing spaces, and then convert this into an integer. This integer is then used to read the correct number of bits from the socket where it then decodes the data, finally returning the data as a string.

Again, if an error occurs during this it means the socket has been closed and therefore it deals with this by closing the socket client-side.

The last method of the Connection is EndConnection, this method is needed so that the server is aware the socket is being closed and that an error hasn't occurred. It first sends the string of "CloseSocket" to the server so that it is aware the socket is being terminated. It then closes the socket client-side.

To make sure the connection module worked properly, I tested it by making a chatroom which allowed 2 users to send messages to each other. Below is when I tested it with my client:

```
Person 1: Hi
Person 2: Hello
Person 2: Second message
Person 1: Reply
Person 2: Reply again
PS C:\Users\henry>
```

The module worked perfectly, and the data had no errors in it which meant it would work as intended for my program.

Coding main_menu.py

12th September 2022: Now that all the modules have been coded, it means that I can start coding the main menu. As described in the LayoutManager module, the first version of the main menu is not what I ended up going with.

The first version of the main menu, as seen above, used absolute co-ordinates which meant if the screen was a different size, the buttons would no longer be in the middle. That is why I then made the LayoutManager module.

Therefore, for the second version of the main menu, I started from scratch as a lot would be changing.

```
# file: main_menu.py

import pygame
import json

from src.modules.music          import Music
from src.modules.layoutmanager  import LayoutManager
from src.modules.cursor         import Cursor
from src.modules.background     import BackgroundManager
from src.modules.connection     import Connection

# Load the constants from the constants file
with open("data\\app\\consts.json") as file:
    CONSTS = json.load(file)

DisplayFPS = False

```

The main menu file starts by importing all the modules it will use. It will use the Music module to play music in the background as described in the Design section, the LayoutManager will be used to manage the buttons and textboxes, the Cursor will allow the user to click the buttons, the background manager will add the stars to the background, and the connection will be used to make sure the program is future-proofed for if the main menu gets updated and requires the use of the server.

It then loads the constants and sets a testing variable DisplayFPS to False.

```

class MainMenu:
    def __init__(self, RES:tuple, CONNECTION:Connection, USERID:str):
        """Initialise the main menu.
        """

        self.__RES = RES
        # Connection and UserID aren't used but are here for future-proofing
        _ = CONNECTION
        _ = USERID

        # Create layout manager with grid size (8, 11) and font size 45
        self.__LayoutManager = LayoutManager(self.__RES, (8,11), 45)
        # Create the background manager
        self.__BackgroundManager = BackgroundManager(self.__RES)
        # Create a transparent pygame surface
        self.__Surface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
        self.__Cursor = Cursor()
        # Create and start playing the music
        self.__Music = Music(CONSTS["pth-mm-theme"], 1, -1)
        self.__Music.Play()

        # Add the title textbox to the layout manager
        self.__LayoutManager.AddTextBox("title", (2.1,1.1), (3.8, 0.8), "Cosmic Quest!")

        # Add the menu buttons to the layout manager
        self.__LayoutManager.AddButton("play",          (3.1,4.1), (1.8,0.8), "Play")
        self.__LayoutManager.AddButton("leaderboard",   (3.1,5.1), (1.8,0.8), "Leader Board")
        self.__LayoutManager.AddButton("exit",          (3.1,6.1), (1.8,0.8), "Exit")
        self.__LayoutManager.AddButton("mute",          (3.1,7.1), (1.8,0.8), "Mute")

```

The main class of the main menu is the MainMenu class which when initiated will create the LayoutManager, BackgroundManager, Surface, Cursor and Music objects. It will then add the title and buttons to the layout manager.

```
47
48      def Run(self, WINDOW:pygame.Surface, CLOCK:pygame.time.Clock):
49          """Run the main menu loop
50
51          Args:
52              Window (surface): The main pygame window surface.
53              Clock (clock): The pygame clock used for the window.
54
55          Returns:
56              NextState (str): The next state to switch to.
57          """
58
59          self.__NextState = "how-to-play"
60          self.__Fading = False
61          self.__Run = True
62
63          # Start the program loop for pygame
64          while self.__Run:
65
66              # Get position of the mouse from pygame
67              MousePosition = pygame.mouse.get_pos()
68              # Get the keys which have been pressed
69              KeysPressed = pygame.key.get_pressed()
70
71              # If users presses ESC key, quit the game
72              if KeysPressed[pygame.K_ESCAPE]:
73                  self.__SafeExit()
74
```

Once the object has been initiated, the run method will be executed which will start the loop. Each time it loops it will get the position of the mouse and if any keys have been pressed. If the Esc key has been pressed then it will close the game by using the SafeExit() method.

```
74
75              for event in pygame.event.get():
76
77                  # If user presses x in the corner
78                  if event.type == pygame.QUIT:
79                      self.__SafeExit()
80
81                  # If user presses a mouse button...
82                  if event.type == pygame.MOUSEBUTTONUP:
83                      # Check if the mouse is over any of the buttons
84                      TempButtonClicked = self.__LayoutManager.CheckMouseCollision(MousePosition)
85                      # If mouse has clicked play button...
86                      if TempButtonClicked == "play":
87                          self.__NextState = "how-to-play"
88                          self.__FadeMenu()
89                      # If mouse has clicked the mute button...
90                      elif TempButtonClicked == "mute":
91                          self.__ToggleMute()
92                      # If mouse has clicked exit button...
93                      elif TempButtonClicked == "exit":
94                          self.__SafeExit()
95                      # If mouse has clicked any other button...
96                      elif TempButtonClicked != None:
97                          # Set next state to the ID of the button clicked
98                          self.__NextState = TempButtonClicked
99                          self.__FadeMenu()
100
```

On line 75, this for loop will check each type of pygame event to see if the window has been closed or if a mouse button has been pressed. If a mouse button has been pressed it will check if a button has been clicked. If a button has been clicked it will trigger the corresponding if statement. Once a button has been clicked that moves to a different section, it will change the variable NextState and then call the FadeMenu() method.

```
100
101             # Add FPS to layout
102             if DisplayFPS:
103                 # Get current fps
104                 TempFPS = round(CLOCK.get_fps(), 2)
105                 # Update the fps textbox in the layout manager
106                 self.__LayoutManager.RemoveTextBox("fps")
107                 self.__LayoutManager.AddTextBox("fps", (0.1,0.1), (0.8,0.8), str(TempFPS))
108
109             # Add background to the surface
110             TempBackgroundSurface = self.__BackgroundManager.Update(self.__Fading)
111             self.__Surface.blit(TempBackgroundSurface, (0,0))
112
113             # Add the layout (buttons) to the surface
114             TempLayoutSurface = self.__LayoutManager.GetSurface(MousePosition)
115             self.__Surface.blit(TempLayoutSurface, (0,0))
116
117             # Add the cursor to the surface at its position
118             TempCursorSurface = self.__Cursor.GetSurface()
119             TempCursorPosition = self.__Cursor.GetPosition(MousePosition)
120             self.__Surface.blit(TempCursorSurface, TempCursorPosition)
121
```

Once all mouse clicks have been checked, it draws everything to the screen in a layered order with things at the back being drawn first. The blit function in Pygame is used to add things to the screen.

```
121
122             # If menu is fading to a different state...
123             if self.__Fading:
124                 # Overlay the screen with the fading surface
125                 # Increases fade level and sets FadeSurface to that level
126                 self.__FadeLevel += CONSTS["fade-amount"]
127                 if self.__FadeLevel > 255:
128                     self.__FadeLevel = 255
129                 # Fill the fade surface with current fadelevel as the alpha channel
130                 self.__FadeSurface.fill((0,0,0,self.__FadeLevel))
131                 # Adds the FadeSurface to the top of the main surface
132                 self.__Surface.blit(self.__FadeSurface, (0, 0))
133                 # If it's fully faded out...
134                 if self.__FadeLevel >= 255:
135                     # Stop the music
136                     self.__Music.Stop()
137                     # Hand control back to main.py
138                     return self.__NextState
139
140             # Add the surface to the window
141             WINDOW.blit(self.__Surface, (0,0))
142             # Delay the loop using pygames clock
143             CLOCK.tick(CONSTS["fps"])
144             # Update the pygame window with whats been changed
145             pygame.display.update()
146         return self.__NextState
147
```

Finally, on top of everything visible, if a button has been clicked then it will run the if statement on line 123. This will slowly fade the screen to black by overlaying a translucent surface on top of everything. Finally at the end of the loop it adds everything to the window. Then it uses the pygame clock to make the program “sleep” the correct amount of time so that the program runs at the desired Frames Per Second. The pygame display update method refreshes the window to show everything that has been added to it. The final line (146) runs once the loop while loop stops.

```
147
148     def __FadeMenu(self):
149         """Start fading the window to a different state.
150         """
151
152         self.__Fading = True
153         self.__FadeSurface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
154         self.__FadeLevel = 0
155         self.__FadeSurface.fill((0, 0, 0, self.__FadeLevel))
156
157     def __SafeExit(self):
158         """Safely exit the program
159         """
160
161         exit()
162
163     def __ToggleMute(self):
164         """Toggle the music from playing and being paused.
165         """
166
167         if self.__Music.GetState() == "playing":
168             self.__Music.Pause()
169         else:
170             self.__Music.Unpause()
171
```

The other methods of the MainMenu class are FadeMenu(), SafeExit(), and ToggleMute().

The FadeMenu method is called when a button is pressed and creates the necessary variables for the screen to start fading. The SafeExit method is used to future-proof the program so that if data needs to be saved before exiting, it can be done here. Then finally the ToggleMute method toggles the music between playing and stopped by getting the current state and then executing the opposite method.

Coding how_to_play.py

20th September 2022: Once I had coded the main menu, the next part of the program that the user will see is the instructions for how to play the game. These are important as it needs to explain how to play the game as well as what they are aiming to do.

```
 1 # file: how_to_play.py
 2
 3 import pygame
 4 import json
 5
 6 from src.main_menu              import BackgroundManager
 7 from src.modules.cursor         import Cursor
 8 from src.modules.layoutmanager  import LayoutManager
 9 from src.modules.connection     import Connection
10
11 import assets.txt.htp as htp_text
12
13 # Load the constants
14 with open("data\\app\\consts.json") as file:
15     CONSTS = json.load(file)
16
17 class howToPlay:
18     def __init__(self, Res:tuple, CONNECTION:Connection, USERID:str):
19         """Initialise the How to Play state.
20         """
21
```

As with the main menu file, it imports the required modules and loads the constants file. The main class of this file is the HowToPlay class.

```
21
22          self.__RES = Res
23          # Connection and UserID aren't used but are here for future-proofing
24          _ = CONNECTION
25          _ = USERID
26
27          # Create the layout and background managers
28          self.__LayoutManager = LayoutManager(self.__RES, (12,14), 30)
29          self.__BackgroundManager = BackgroundManager(self.__RES)
30          # Create the main pygame surface
31          self.__Surface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
32          self.__Cursor = Cursor()
33
34          # Create variables to manage the text
35          self.__TextList = htp_text.text
36          self.__TextPosition = 0
37          self.__MaxTextPosition = len(self.__TextList)-1
38
39          # Create the textboxes based off of the text position
40          self.__CreateTextBoxes(self.__TextPosition)
41          # Add the navigation buttons to the layout
42          self.__LayoutManager.AddButton("back", (1.1,12.1), (2.8,0.8), "Back")
43          self.__LayoutManager.AddButton("skip", (4.1,12.1), (3.8,0.8), "Skip")
44          self.__LayoutManager.AddButton("next", (8.1,12.1), (2.8,0.8), "Next")
45
```

When this class is instantiated, it will do the same as the main menu class but will have 3 extra attributes to store the text for the instructions, the position in the instructions, and the total number of instructions. It also makes use a method, CreateTextBoxes(), which will be explained below.

```
45
46      def __CreateTextBoxes(self, Index:int):
47          """Create textboxes using the index.
48
49          Args:
50              Index (int): Current position in the list of text.
51          """
52
53          # Delete the current title and text textboxes
54          self.__LayoutManager.RemoveTextBox("title")
55          self.__LayoutManager.RemoveTextBox("body")
56
57          # Create the new title textbox
58          TempTitleText = self.__TextList[Index][0]
59          self.__LayoutManager.AddTextBox("title", (1.1,1.1), (9.8,0.8), TempTitleText)
60
61          # Create the new body textbox
62          TempBodyText = self.__TextList[Index][1]
63          self.__LayoutManager.AddTextBox("body", (1.1,3.1), (9.8,7.8), TempBodyText, "L")
64
```

This method will be used to create the textboxes to display the current instructions. First it will remove any existing textboxes by using the LayoutManager's remove textbox method. It then gets the title and body of the current instruction and creates a textbox for each one. On line 63 you can see the alignment parameter has been used to left justify the text.

```
64
65      def Run(self, WINDOW:pygame.Surface, CLOCK:pygame.time.Clock):
66          """Run the how to play loop
67
68          Args:
69              Window (surface): The main pygame window surface.
70              Clock (clock): The pygame clock used for the window.
71
72          Returns:
73              NextState (str): The next state to switch to.
74          """
75
76          self.__NextState = "game"
77          self.__Fading = False
78          self.__Run = True
79
80          # Start the program loop for pygame
81          while self.__Run:
82
83              # Gets position of the mouse from pygame
84              MousePosition = pygame.mouse.get_pos()
85              # Gets any keys pressed
86              KeysPressed = pygame.key.get_pressed()
87
```

The run method of HowToPlay is much like the main menu class and has the same attributes.

```
87
88              # If user presses ESC key, move them back to the main menu
89              if KeysPressed[pygame.K_ESCAPE]:
90                  self.__NextState = "main-menu"
91                  self.__FadeMenu()
92
93              for event in pygame.event.get():
94                  # If user clicks x to close window
95                  if event.type == pygame.QUIT:
96                      self.__SafeExit()
97
98                  # If user clicks a mouse button
99                  if event.type == pygame.MOUSEBUTTONUP:
100                     # Detect if they clicked a button and if so what button
101                     TempButtonClicked = self.__LayoutManager.CheckMouseCollision(MousePosition)
102
103                     # If they clicked the back button...
104                     if TempButtonClicked == "back":
105                         # As long as its not the first page...
106                         if self.__TextPosition > 0:
107                             self.__TextPosition -= 1
108                             self.__CreateTextBoxes(self.__TextPosition)
109
110                     # If they clicked the skip button...
111                     elif TempButtonClicked == "skip":
112                         self.__NextState = "game"
113                         self.__FadeMenu()
114
```

The only difference between the mouse detections here and in the main menu is that the buttons have different IDs and when one is pressed, they perform different functions.

```
114
115                     # If they clicked the next button...
116                     elif TempButtonClicked == "next":
117                         # Move to next page
118                         self.__TextPosition += 1
119
120                         # If the last page, change button to start button
121                         if self.__TextPosition == self.__MaxTextPosition:
122                             self.__LayoutManager.RemoveButton("next")
123                             self.__LayoutManager.AddButton("next", (8.1,12.1), (2.8,0.8),
    "Start")
124
125                         # Update textboxes as long as they didn't click start button
126                         if self.__TextPosition <= self.__MaxTextPosition:
127                             self.__CreateTextBoxes(self.__TextPosition)
128
129                         # Start game
130                         else:
131                             self.__NextState = "game"
132                             self.__FadeMenu()
133
```

If the next button is pressed it increases the text position by 1 but if it is moving onto the last instruction, it changes the button from saying Next to saying Start. Then if the Start button is pressed, it does the same as the main menu to fade the screen and move to the next section.

```
133
134             # Add background to the surface
135             TempBackgroundSurface = self.__BackgroundManager.Update(self.__Fading)
136             self.__Surface.blit(TempBackgroundSurface, (0,0))
137
138             # Add the layout to the surface
139             TempLayoutSurface = self.__LayoutManager.GetSurface(MousePosition)
140             self.__Surface.blit(TempLayoutSurface, (0,0))
141
142             # Add the cursor to the surface
143             TempCursorSurface = self.__Cursor.GetSurface()
144             TempCursorPosition = self.__Cursor.GetPosition(MousePosition)
145             self.__Surface.blit(TempCursorSurface, TempCursorPosition)
146
```

Once all the mouse and button detections have been completed, it adds everything to the surface in order.

```
146
147             # If menu is fading to a different state...
148             if self.__Fading:
149                 # Overlay the screen with the fading surface
150                 # Increases fade level and sets FadeSurface to that level
151                 self.__FadeLevel += CONSTS["fade-amount"]
152                 if self.__FadeLevel > 255:
153                     self.__FadeLevel = 255
154                 # Fill the fade surface with current fadelevel as the alpha channel
155                 self.__FadeSurface.fill((0,0,0,self.__FadeLevel))
156                 # Adds the FadeSurface to the top of the main surface
157                 self.__Surface.blit(self.__FadeSurface, (0, 0))
158                 # If it's fully faded out...
159                 if self.__FadeLevel >= 255:
160                     # Hand control back to main.py
161                     return self.__NextState
162
163             # Adds the surface to the main window
164             WINDOW.blit(self.__Surface, (0,0))
165
166             CLOCK.tick(CONSTS["fps"])
167             pygame.display.update()
168         return self.__NextState
169
```

If the screen is fading because its moving to the game, it runs the same if statement as the main menu.

```
169
170     def __FadeMenu(self):
171         """Start fading the window to a different state.
172         """
173
174         self.__Fading = True
175         self.__FadeSurface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
176         self.__FadeLevel = 0
177         self.__FadeSurface.fill((0, 0, 0, self.__FadeLevel))
178
179     def __SafeExit(self):
180         """Safely exit the program
181         """
182
183         exit()
184
```

These 2 methods are the same as the MainMenu but can't be inherited from a template as they may need to do different functions to the main menu in the future. (For example, needing to save different variables than the main menu when being closed.)

Coding game.py

27th September 2022: The user will spend the most amount of time playing the game, so this is the most important file of the program.

```
1  # file: game.py
2
3  import pygame
4  import math
5  import random
6  import json
7  import time
8
9  from src.modules.cursor         import Cursor
10 from src.modules.layoutmanager  import LayoutManager
11 from src.modules.background     import BackgroundManager
12 from src.modules.connection     import Connection
13
14 # Load constants from file
15 with open("data\\app\\consts.json") as file:
16     CONSTS = json.load(file)
17
18 DisplayFPS = False
19
```

The file starts the same as the main menu and the how to play files.

```
19
20 class Game:
21     def __init__(self, RES:tuple, CONNECTION:Connection, USERID:str):
22         """Initialise the game.
23         """
24
25         self.__RES = RES
26         self.__CONNECTION = CONNECTION
27         self.__USERID = USERID
28
29         # Create the main pygame surface
30         self.__Surface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
31
32         # Create planet manager and the planets
33         self.__PlanetManager = PlanetManager(self.__RES)
34         self.__PlanetManager.CreatePlanets()
35
36         # Load questions from database
37         NumberOfQuestions = CONSTS["number-of-questions"]
38         Questions = self.__LoadQuestions(NumberOfQuestions)
39         self.__QuestionManager = QuestionManager(self.__RES, Questions)
40
41         self.__LayoutManager = LayoutManager(self.__RES, (12,14), 35)
42         self.__BackgroundManager = BackgroundManager(self.__RES)
43         self.__Timer = Timer()
44         self.__Rocket = Rocket()
45         self.__Cursor = Cursor()
46
```

The main class of this file is the Game class. When this is initialised, it stores the parameters as attributes and then creates the Surface, PlanetManager, QuestionManager, LayoutManager, BackgroundManager, Timer, Rocket, and Cursor objects. It also runs the LoadQuestions method of Game which requests questions from the server using the connection.

```
46
47     def __LoadQuestions(self, NumberOfQuestions:int):
48         """Load questions from database.
49
50         Returns:
51             Questions (list): List containing the questions.
52         """
53
54         # Send request to server for the number of questions
55         self.__CONNECTION.Send(f"104##{NumberOfQuestions}")
56         # Receive the questions
57         TempQuestions = self.__CONNECTION.Receive()
58         TempQuestions = self.__ConvertStringToQuestions(TempQuestions)
59
60         # The data within the TempQuestions still has $$ as new lines.
61         # Split the question data into new lines (items in a list)
62         Questions = []
63         for Question in TempQuestions:
64             # Split the question and answers into lists of the lines
65             TempList = []
66             TempList.append(Question[0].split("$$"))
67             # This is the index of the correct answer
68             TempList.append(int(Question[1]))
69             TempList.append(Question[2].split("$$"))
70             TempList.append(Question[3].split("$$"))
71             TempList.append(Question[4].split("$$"))
72             TempList.append(Question[5].split("$$"))
73             # Add the TempList to the Questions list
74             Questions.append(TempList)
75         return Questions
76
```

The LoadQuestions method sends a request using the connection to get questions from the server. It does this using the query ID of 104. It then uses the separator of ## which the server will use to turn the string into the parameters for the query. It then sends the number of questions it wants.

It then receives the questions from the server and uses the ConvertStringToQuestions method to produce a list of questions (the questions are in string format). The program then needs to convert the string-formatted questions into a list per question so that the data can be used to make the textboxes and buttons.

Each question at this stage is in this format:

$$[Question, CorrectAnswerIndex, Answer1, Answer2, Answer3, Answer1]$$

But the problem is that the questions and answers have $$ within them to represent new lines. Therefore, these need to be split up into a list of each line of text. To do this it loops through each question and splits the question and answers into lists which are then added into another list which stores everything for that question. This question list is then added to a list of all the questions.

```
76
77      def __ConvertStringToQuestions(self, String:str):
78          """Convert the data received from the server into a list of questions.
79
80          Args:
81              String (str): The raw string received from the server.
82
83          Returns:
84              Questions (list): List of lists containing the questions.
85          """
86          # Split the raw string into a list of items
87          SplitString = String.split("&&")
88
89          Questions = []
90          while len(SplitString) != 0:
91              # Group the items into groups of 6
92              # as each question has 6 items
93              TempQuestion = []
94              # Add 6 items to TempQuestion
95              for i in range(6):
96                  TempQuestion.append(SplitString[i])
97              # Delete the 6 items from the SplitString list
98              for i in range(6):
99                  del SplitString[0]
100             # Add the TempQuestion to the Questions list
101             Questions.append(TempQuestion)
102         return Questions
103
```

This method is used to convert the response from the server which is a single string into a list of questions. The way the data is formatted is like this:

$$Q1 \;\&\&\; CorrectIndex1 \;\&\&\; Ans1 \;\&\&\; Ans2 \;\&\&\; Ans3 \;\&\&\; Ans4 \;\&\&\; Q2 \;\&\&\; CorrectIndex2 \;\&\&\; Ans1 \ldots$$

All the questions' data is in a single long list so must be separated and split into the questions. To do this it splits the string using the && separator and the loops through this split list until its length becomes zero. While it is looping it creates a list which will store a question. It then gets the next 6 items in the list and adds them to this question list. Next it deletes these 6 items from the split list. It then adds this question list to the list of all the questions. Once the split list becomes empty because all the questions have been separated, it will return the list of Questions.

```
103
104     def Run(self, WINDOW:pygame.Surface, CLOCK:pygame.time.Clock):
105         """Run the main game loop
106
107         Returns:
108             NextState (str): The next state to switch to.
109         """
110         # Set the rocket to the position of the 1st planet
111         TempRocketPosition = self.__PlanetManager.GetCentrePositionOf(0)
112         self.__Rocket.Reset(TempRocketPosition)
113
114         self.__NextState = "main-menu"
115         self.__AnsweringQuestion = False
116         self.__GameOver = False
117         self.__GameOverManaged = False
118         self.__ScoreSaved = False
119         self.__FirstKeyPressed = False
120
121         self.__Fading = False
122         self.__Run = True
123
124         # Start the main game loop
125         while self.__Run:
126
127             # Default these variables
128             MouseClicked = None
129             self.__Rocket.SetState("off")
130
131             # Get Keys pressed and position of mouse
132             KeysPressed = pygame.key.get_pressed()
133             MousePosition = pygame.mouse.get_pos()
134
```

The main method of the Game class is the Run function which will run the loop used to manage the pygame window and player while the user plays the game. This loop will take them from starting the game to once the game has finished. It starts by defining various variables which will be used throughout the loop.

Once the loop starts, it sets the Rockets state to off, as well as getting the mouses position and if any keys have been pressed.

```
134
135             # If user presses ESC, return them to the main menu
136             if KeysPressed[pygame.K_ESCAPE]:
137                 self.__NextState = "main-menu"
138                 self.__FadeMenu()
139
140             for event in pygame.event.get():
141                 # If user presses x, close the window
142                 if event.type == pygame.QUIT:
143                     self.__SafeExit()
144
145                 # If user presses a mouse button...
146                 if event.type == pygame.MOUSEBUTTONUP:
147                     MouseClicked = True
148
149             # Check key presses only if flying the rocket
150             if not self.__AnsweringQuestion and not self.__GameOver:
151
152                 # If "a" key pressed... (rotate left)
153                 if KeysPressed[pygame.K_a]:
154                     # If they haven't moved yet...
155                     if not self.__FirstKeyPressed:
156                         self.__FirstKeyPressed = True
157                         # Start the timer
158                         self.__Timer.Start()
159                     self.__Rocket.Rotate("anticlockwise")
160
```

First it manages key presses and mouse clicks. If they Esc key is pressed the game will return to the main menu.

If the user clicks a mouse button, then the MouseClicked attribute is set to True which will be needed later on.

If the user is not answering a question and they game isn't over (meaning they are flying the rocket) then it will detect the key presses for controlling the rocket. If 'a' is pressed, then it checks if it is the first key to be pressed. This is so that the timer only starts once the player starts moving. Once it manages that, it then uses the Rotate method of the rocket and passing in the direction of anti-clockwise.

```
160
161             # If "d" key pressed... (rotate right)
162             if KeysPressed[pygame.K_d]:
163                 if not self.__FirstKeyPressed:
164                     self.__FirstKeyPressed = True
165                     self.__Timer.Start()
166                 self.__Rocket.Rotate("clockwise")
167
168             # If "w" key pressed... (forwards)
169             if KeysPressed[pygame.K_w]:
170                 if not self.__FirstKeyPressed:
171                     self.__FirstKeyPressed = True
172                     self.__Timer.Start()
173                 # Set rockets state to ON as its moving forwards
174                 self.__Rocket.SetState("on")
175                 self.__Rocket.Move("forward")
176
177             # If "s" key pressed... (backwards)
178             if KeysPressed[pygame.K_s]:
179                 if not self.__FirstKeyPressed:
180                     self.__FirstKeyPressed = True
181                     self.__Timer.Start()
182                 self.__Rocket.Move("back")
183
184             # If "r" key pressed... (reset position)
185             if KeysPressed[pygame.K_r]:
186                 # Get position of 1st planet
187                 TempPosition = self.__PlanetManager.GetCentrePositionOf(0)
188                 # Reset rocket to that position
189                 self.__Rocket.Reset(TempPosition)
190
```

If 'd' is pressed it does the same as if 'a' is pressed but rotates the opposite direction.

If 'w' is pressed and once it checks if it is the first key pressed, it Sets the rockets state to on which means it will show flames under the rocket to show its moving as well as then moving the rocket and passing in the direction of "forward".

Similarly, it uses "back" when 'd' is pressed but does not set the state to on as it is moving backwards not forwards.

The user can press the 'r' key to reset the rocket back to the first planet if they go the wrong direction or the rocket goes off the screen. When this is pressed it gets the position of the centre of the 1st planet and then uses the Reset method of the rocket to reset the rocket back to that planet.

```
190
191             # Get position of the rocket
192             TempRocketPosition = self.__Rocket.GetPosition()
193             # Check if rocket is colliding with next planet
194             if self.__PlanetManager.CheckCollision(TempRocketPosition) == 1 and not
    self.__AnsweringQuestion:
195                 # Set answeringquestion to True and create question
196                 self.__AnsweringQuestion = True
197                 self.__QuestionManager.CreateQuestion()
198
199             # If user is answering a question...
200             if self.__AnsweringQuestion:
201                 # If user has clicked the mouse...
202                 if MouseClicked:
203                     # Check if user has clicked an answer and if it is correct
204                     CorrectAnswerClicked = self.__QuestionManager.CheckClick(MousePosition)
205
206                     # If correct answer clicked...
207                     if CorrectAnswerClicked:
208                         # Allow user to carry on
209                         self.__AnsweringQuestion = False
210                         self.__PlanetManager.CreatePlanets()
211                         self.__Rocket.Reset(self.__PlanetManager.GetCentrePositionOf(0))
212
```

Once the rocket has been updated, it then checks if it is now colliding with the next planet but only if they aren't answering a question. If they are colliding with the planet, then it sets AnsweringQuestion to True and runs the CreateQuestion method of the QuestionManager.

Now, if the user is answering a question, it will first check if the mouse has been clicked. If it has, it passes the mouse position into the QuestionManager which will check if the user has clicked the correct answer.

If they clicked the correct answer then it will set AnsweringQuestion to False, create new planets, and then Reset the rocket to the new starting planet.

```
212
213                     # If user gets question wrong...
214                     elif CorrectAnswerClicked == False:
215                         # Add time penalty
216                         self.__Timer.AddTime(CONSTS["wrong-answer-time-penalty"])
217                         # Give user another question
218                         self.__QuestionManager.CreateQuestion()
219
220                     # If all questions have been answered...
221                     if self.__QuestionManager.CheckNumberOfQuestionsCorrect():
222                         # Stop the timer
223                         self.__Timer.Stop()
224                         # Disable rocket by setting gameover to True
225                         self.__GameOver = True
226                         # Set mouseClicked to None so a button isn't instantly clicked
227                         MouseClicked = None
228
```

However, if the user answered the question incorrectly, a time penalty will be added to the Timer object which is a value stored within the constants file. It will then create a new question for the user to attempt.

If all the questions have been answered then it stops the timer, sets GameOver to True and sets the MouseClicked to None. MouseClicked needs to be set to None as it will sometimes click a new button that the user didn't intend to click.

```
228
229             # Add background to the surface
230             TempBackgroundSurface = self.__BackgroundManager.Update(self.__Fading)
231             self.__Surface.blit(TempBackgroundSurface, (0,0))
232
233             # If answering a question...
234             if self.__AnsweringQuestion:
235                 # Display the questions
236                 TempSurface = self.__QuestionManager.GetSurface(MousePosition)
237                 self.__Surface.blit(TempSurface, (0,0))
238
239             # If playing the game...
240             elif not self.__AnsweringQuestion and not self.__GameOver:
241                 # Add Planets to surface
242                 TempSurface = self.__PlanetManager.GetSurface()
243                 self.__Surface.blit(TempSurface, (0,0))
244
245                 # Add Rocket to surface
246                 self.__Rocket.Update()
247                 TempSurface = self.__Rocket.GetSurface()
248                 TempPosition = self.__Rocket.GetTopLeftPosition()
249                 self.__Surface.blit(TempSurface, TempPosition)
250
```

The background is then added to the surface as this will always be visible whether they are answering a question, playing the game, or they've finished the game.

If the user is answering a question, it will get the surface from the question manager and then add it to the main surface.

If they aren't answering questions and the game isn't over, then it adds the planets and rocket to the surface.

```
250
251             # If user is playing the game...
252             if not self.__GameOver:
253                 # Remove current timer textbox
254                 self.__LayoutManager.RemoveTextBox("timer")
255                 # Get new time and add new timer textbox
256                 TempTime = self.__Timer.GetTimeAsString()
257                 self.__LayoutManager.AddTextBox("timer", (5.1,13.1), (1.8,0.8), TempTime)
258
259             # If user has finished the game...
260             if self.__GameOver:
261                 # If gameover hasn't been managed (saved score)...
262                 if not self.__GameOverManaged:
263                     self.__ManageGameOver()
264
265                 # If user has clicked mouse...
266                 if MouseClicked:
267                     # Detect what it has clicked
268                     Index = self.__LayoutManager.CheckMouseCollision(MousePosition)
269
270                     if Index == "menu":
271                         self.__NextState = "main-menu"
272                         self.__FadeMenu()
273
274                     elif Index == "leaderboard":
275                         self.__NextState = "leaderboard"
276                         self.__FadeMenu()
277
278                     elif Index == "play-again":
279                         self.__NextState = "how-to-play"
280                         self.__FadeMenu()
281
282                     elif Index == "exit":
283                         self.__SafeExit()
284
```

Then if the user is answering questions or flying the rocket, it will also add the timer to the surface.

Once the user has completed the game, it will check to see if the GameOver has been managed, if not it will do so by calling the method. It will then check for mouse collisions with any of the new buttons which are for the user to move onto another section of the program.

```
284
285             # If display FPS is True...
286             if DisplayFPS:
287                 # Get FPS
288                 TempFPS = round(CLOCK.get_fps(), 2)
289                 # Update the FPS textbox
290                 self.__LayoutManager.RemoveTextBox("fps")
291                 self.__LayoutManager.AddTextBox("fps", (0.1,0.1), (0.8,0.8), str(TempFPS))
292
293             # Add Layout to surface
294             TempSurface = self.__LayoutManager.GetSurface(MousePosition)
295             self.__Surface.blit(TempSurface, (0,0))
296
297             # If answering a question or game is over...
298             if self.__AnsweringQuestion or self.__GameOver:
299                 # Display the cursor
300                 # Add the cursor to the surface
301                 TempSurface = self.__Cursor.GetSurface()
302                 TempPosition = self.__Cursor.GetPosition(MousePosition)
303                 self.__Surface.blit(TempSurface, TempPosition)
304
```

The last things to be displayed are the FPS if its set to True, the LayoutManager which contains the Timer, and the cursor if the user is answering a question or has finished the game.

```
304
305             # If menu is fading to a different state...
306             if self.__Fading:
307                 # Overlay the screen with the fading surface
308                 # Increases fade level and sets FadeSurface to that level
309                 self.__FadeLevel += CONSTS["fade-amount"]
310                 if self.__FadeLevel > 255:
311                     self.__FadeLevel = 255
312                 # Fill the fade surface with current fadelevel as the alpha channel
313                 self.__FadeSurface.fill((0,0,0,self.__FadeLevel))
314                 # Adds the FadeSurface to the top of the main surface
315                 self.__Surface.blit(self.__FadeSurface, (0, 0))
316                 # If it's fully faded out...
317                 if self.__FadeLevel >= 255:
318                     # Hand control back to main.py
319                     return self.__NextState
320
321             # Add the main surface to the window
322             WINDOW.blit(self.__Surface, (0,0))
323             # Delay the loop using pygames clock
324             CLOCK.tick(CONSTS["fps"])
325             # Update the pygame display
326             pygame.display.update()
327         return self.__NextState
328
```

Like the other sections of the program, if the game is about to move to a different section it will fade the screen using this algorithm.

```
328
329     def __FadeMenu(self):
330         """Start fading the window to a different state.
331         """
332
333         self.__Fading = True
334         self.__FadeSurface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
335         self.__FadeLevel = 0
336         self.__FadeSurface.fill((0, 0, 0, self.__FadeLevel))
337
338     def __SafeExit(self):
339         """Safely exit the program
340         """
341
342         exit()
343
```

FadeMenu and SafeExit are the same methods as in the other sections.

```
344     def __ManageGameOver(self):
345         """Gets score and sends it to the server.
346         """
347
348         self.__GameOverManaged = True
349         self.__LayoutManager.RemoveTextBox("timer")
350
351         # Get score as number and as minutes, seconds, milliseconds
352         self.__ScoreAsString = self.__Timer.GetTimeAsString()
353         self.__Score = self.__Timer.GetTime()
354
355         # Display the users score
356         self.__LayoutManager.AddTextBox("score", (4.1,4.1), (3.8,0.8), f"Your time is:
    {self.__ScoreAsString}")
357
358         # Send the users score to the server with their USERID
359         self.__CONNECTION.Send(f"102##{self.__USERID}##{self.__Score}")
360         self.__CONNECTION.Receive()
361         self.__ScoreSaved = True
362
363         if self.__ScoreSaved:
364             # Once score has been saved in the database, show the next options
365             self.__LayoutManager.AddButton("menu",          (4.1,6.1), (3.8, 0.8), "Menu")
366             self.__LayoutManager.AddButton("leaderboard",   (4.1,7.1), (3.8, 0.8),
    "Leaderboard")
367             self.__LayoutManager.AddButton("play-again",    (4.1,8.1), (3.8, 0.8), "Play Again")
368             self.__LayoutManager.AddButton("exit",          (4.1,9.1), (3.8, 0.8), "Exit")
369
```

ManageGameOver is used as soon as GameOver becomes True. First it Removes the timer from the screen, then gets the score from the timer. It displays this score to the user in a text box. It then sends this score to the server using the Query ID of 102 and it sends with it the Users ID. Once the score has been received, it will then create and display buttons to the user to either the menu, leader board, play again, or exit.

Coding the Timer

```
369
370 class Timer:
371     def __init__(self):
372         """Used to record the time of the users game.
373         """
374
375         self.__StartTime = None
376         self.__EndTime = None
377         self.__ElapsedTime = 0
378
379     def Start(self):
380         """Start the timer.
381         """
382
383         self.__StartTime = time.time_ns()
384
385     def Stop(self):
386         """Stop the timer.
387         """
388
389         self.__EndTime = time.time_ns()
390         self.__ElapsedTime += self.__EndTime - self.__StartTime
391         self.__StartTime = None
392
```

The Timer class is used to keep track of how long the user takes to answer the questions. It keeps track of time in nanoseconds so that it is much more precise than just seconds. The 3 attributes, StartTime, EndTime, and ElapsedTime, are used to keep track of time but to also allow the timer to be stopped and started again if necessary.

```
392
393     def AddTime(self, Seconds:int):
394         """Add time to the timer.
395
396         Args:
397             Seconds (int): The number of seconds to add.
398         """
399         # Convert the seconds to nanoseconds then add it on
400         self.__ElapsedTime += Seconds*1000000000
401
402     def GetTime(self):
403         """Get the current time of the timer.
404
405         Returns:
406             Time (int): Time in nanoseconds.
407         """
408
409         # If timer is running...
410         if self.__StartTime is not None:
411             # Get time since start
412             CurrentElapsedTime = time.time_ns() - self.__StartTime
413             TotalTime = self.__ElapsedTime + CurrentElapsedTime
414
415         # If timer hasn't been started...
416         else:
417             TotalTime = self.__ElapsedTime
418         return TotalTime
419
```

The AddTime method is used when the user gets a question wrong as it gives them a time penalty of a couple seconds. It takes in the parameter of the number of seconds to add on so must convert the seconds to nanoseconds and then add that onto the elapsed time.

The GetTime method is used to return the current time of the timer in nanoseconds. Is does this by calculating the difference between the end and start times and adding this to the elapsed time.

```
419
420     def GetTimeAsString(self):
421         """Get the timers time as a string.
422
423         Returns:
424             string: the time in this format 'minutes:seconds.milliseconds'
425         """
426
427         TotalTime = self.GetTime()
428         return self.__CovertNanosecondsToString(TotalTime)
429
430     def __CovertNanosecondsToString(self, Time):
431         """Converts the time into minutes seconds and milliseconds.
432
433         Returns:
434             string: the time like this 'minutes:seconds.milliseconds'
435         """
436
437         # Convert time to milliseconds
438         Time = Time/1000000
439         # Calculate minutes, seconds and milliseconds
440         Minutes = Time // 60000
441         remainder = Time % 60000
442         Seconds = remainder // 1000
443         Milliseconds = remainder % 1000
444
445         # Round everything to an integer
446         # Make each number take up 2 characters
447         Minutes = str(round(Minutes)).zfill(2)
448         Seconds = str(round(Seconds)).zfill(2)
449         Milliseconds = str(round(Milliseconds))[:2].zfill(2)
450         return f"{Minutes}:{Seconds}.{Milliseconds}"
451
```

The GetTimeAsString method is used to get the time on the timer as a formatted string to be displayed to the user in the Timer textbox. It gets the time in nanoseconds and then converts this time into a string using the ConvertNanosecondsToString method.

The ConvertNanosecondsToString method is a private method of the Timer class. It takes in the time in nanoseconds and splits it into minutes, seconds, and milliseconds. It does so by using integer division and the modulo to get the rounded values, and remainders for the different units. It also makes use of the zfill() function which fills the string with leading zeros to make the string the same length every time.

Coding the Planet

```
451
452 class Planet:
453     def __init__(self, CentrePosition:tuple, Radius:int):
454         """Create a planet object with specific centre and size.
455
456         Args:
457             CentrePosition (tuple): (X, Y) of the centre position.
458             Radius (int): Radius of the planet.
459         """
460
461         self.__CentrePosition = CentrePosition
462         self.__Radius = Radius
463         # Caculate the position ofthe top left of the planet
464         self.__TopLeftPosition = (self.__CentrePosition[0] - 0.5*self.__Radius,
    self.__CentrePosition[1] - 0.5*self.__Radius)
465         # Load the image of the planet
466         self.__Surface = pygame.image.load(CONSTS["pth-img-planet"])
467         # Scale the image to the correct size
468         self.__Surface = pygame.transform.smoothscale(self.__Surface, (self.__Radius,
    self.__Radius))
469
```

The Planet class when initiated, has various attributes that are created. Importantly, it has the TopLeftPosition attribute which is important for adding the planet to the screen. Once it loads the image of the planet from the image file, it also scales the image to the correct size using Pygame's built-in scaler.

```
469
470     def CheckCollision(self, Position:tuple):
471         """Detect whether the planet is colliding with a point.
472
473         Args:
474             Position (tuple): (X, Y) of the point.
475
476         Returns:
477             True: If planet is colliding with the point.
478             False: If not colliding.
479         """
480
481         # Calculate using pythagoras' theorem the distance between the centre and the point
482         TempDistance = math.sqrt((Position[0] - self.__CentrePosition[0])**2 + (Position[1] -
    self.__CentrePosition[1])**2)
483
484         # If the Distance is less than the radius...
485         if TempDistance < self.__Radius:
486             # Point is colliding with the planet so return True
487             return True
488
489         # distance bigger than radius...
490         else:
491             # Return False as no collision
492             return False
493
```

The CheckCollision method is used to detect if the rocket has collided with the planet. It does this by using Pythagoras' theorem to calculate the direct distance between the centre of the planet and the centre of the rocket. If this distance is less than the radius of the planet, then it means the rocket is definitely colliding with the planet.

```
493
494     def GetSurface(self):
495         """Get the surface of the planet.
496
497         Returns:
498             Pygame Surface: Surface containing the image of the planet.
499         """
500
501         return self.__Surface
502
503     def GetCentrePosition(self):
504         """Get the centre position of the planet.
505
506         Returns:
507             tuple: (X, Y) of the centre.
508         """
509
510         return self.__CentrePosition
511
```

These methods are needed to return certain variables that will be needed outside the class, and they better encapsulate the Planet class.

```
511
512     def GetTopLeftPosition(self):
513         """Get the position of the top left corner of the planet.
514
515         Returns:
516             tuple: (X, Y) of the top left corner.
517         """
518
519         return self.__TopLeftPosition
520
521     def SetCentrePosition(self, CentrePosition:tuple):
522         """Set the planet to a position.
523
524         Args:
525             CentrePosition (tuple): (X, Y) of new centre.
526         """
527
528         self.__CentrePosition = CentrePosition
529
```

```
529
530 class Rocket:
531     def __init__(self):
532         """Create a rocket object.
533         """
534
535         # Create rockets angle variables
536         self.__Angle = 0
537         self.__AngleVelocity = 0
538         self.__AngleAcceleration = CONSTS["gm-rocket-angle-acceleration"]
539
540         # Create rockets position and speed variables
541         self.__Position = (0,0)
542         self.__Velocity = 0
543         self.__Acceleration = CONSTS["gm-rocket-acceleration"]
544
545         # Off means forwards is not being pressed
546         self.__CurrentState = "off"
547
548         # Load the images of the rocket
549         self.__OriginalSurfaceOff = pygame.image.load(CONSTS["pth-img-rocket-off"])
550         self.__OriginalSurfaceOn = pygame.image.load(CONSTS["pth-img-rocket-on"])
551
```

The Rocket class has various attributes for storing its rotation, position, speed, state, and surfaces. The rocket's angle and position each have a velocity as well as an acceleration which make them act as if they were on ice or in this case as if they were in space. Two different surfaces are loaded as the rocket has 2 states which either have flames coming out the bottom of it, or no flames.

```
551
552         # Scale the images of the rocket
553         self.__Size = (CONSTS["gm-rocket-width"], CONSTS["gm-rocket-height"])
554         self.__OriginalSurfaceOff = pygame.transform.smoothscale(self.__OriginalSurfaceOff,
    self.__Size)
555         self.__OriginalSurfaceOn = pygame.transform.smoothscale(self.__OriginalSurfaceOn,
    self.__Size)
556
557         # Create the Current surfaces
558         self.__CurrentSurfaceOff = self.__OriginalSurfaceOff.copy()
559         self.__CurrentSurfaceOn = self.__OriginalSurfaceOn.copy()
560
561         # Get current size of the rocket
562         self.__Size = self.__CurrentSurfaceOn.get_size()
563
```

The Surfaces are then scaled to the correct size and they are then copied to the CurrentSurface attributes. The OriginalSurfaces will not be changed and are needed because as the surface is rotated, it loses quality and if the same surface is continually rotated, the image becomes very distorted.

```
563
564     def Update(self):
565         """Update the rocket each frame regardless of keys being pressed.
566         """
567
568         # Scale Velocity by frictional constant
569         self.__Velocity *= CONSTS["gm-friction-constant"]
570
571         # Increase angle by the current angle velocity
572         self.__Angle += self.__AngleVelocity
573         # Scale angle velocity by frictional constant
574         self.__AngleVelocity *= CONSTS["gm-friction-constant"]
575
576         # Calculate the new position of the rocket
577         self.__CalculateNewPosition()
578         # Render the new position of the rocket
579         self.__Render()
580
```

The Update method is called every time the loop cycles through and reduces the velocity and angle velocity by a small percentage. This is so that they slow down over time instead of stopping as soon as the user stops pressing a key. It then calculates the new position of the Rocket and finally renders the new rocket.

```
580
581     def Rotate(self, Direction:str):
582         """Rotate the rocket.
583         Args:
584             Direction (str): 'clockwise' or 'anticlockwise'.
585         """
586
587         # Limit Angle velocity so that it doesn't become too fast
588         if self.__AngleVelocity > CONSTS["gm-rocket-max-angle-velocity"]:
589             return
590
591         # If rotating clockwise, decrease AngleVelocity
592         if Direction == "clockwise":
593             self.__AngleVelocity -= self.__AngleAcceleration
594
595         # If rotating anti-clockwise, increase AngleVelocity
596         else:
597             self.__AngleVelocity += self.__AngleAcceleration
598
```

This method is called whenever the user presses the ‘a’ or ‘d’ key. First it checks if the angle velocity is above a certain threshold, and if it is it breaks there and returns. This is so that the user can’t rotate the rocket faster and faster. It then adjusts the angle velocity by a small amount either positively or negatively depending on the direction.

```
598
599     def Move(self, Direction:str):
600         """Move the rocket.
601
602         Args:
603             Direction (str): 'forward' or 'back'.
604         """
605
606         # If forwards...
607         if Direction == "forward":
608             # Set state to on and increase its velocity
609             self.__CurrentState = "on"
610             self.__Velocity += self.__Acceleration
611
612         # If moving backwards
613         elif Direction == "back":
614             # Set state to off
615             self.__CurrentState = "off"
616             # Decrease acceleration by a smaller amount than forwards
617             self.__Velocity -= self.__Acceleration*0.25
618
```

This method is called whenever ‘s’ or ‘w’ are pressed and they adjust the rockets velocity by the acceleration if its forwards, or 25% of the acceleration if its backwards.

```
618
619     def __Render(self):
620         """Render the surface of the rocket.
621         """
622
623         # Covert the angle of the rocket into degrees from radians
624         TempAngle = -math.degrees(self.__Angle)
625
626         # Rotate the Off state image
627         self.__CurrentSurfaceOff = self.__OriginalSurfaceOff.copy()
628         self.__CurrentSurfaceOff = pygame.transform.rotate(self.__CurrentSurfaceOff, TempAngle)
629
630         # Rotate the On state image
631         self.__CurrentSurfaceOn = self.__OriginalSurfaceOn.copy()
632         self.__CurrentSurfaceOn = pygame.transform.rotate(self.__CurrentSurfaceOn, TempAngle)
633
634         # Get the size of the rotated image
635         self.__Size = self.__CurrentSurfaceOn.get_size()
636
```

This method is used to render the rocket to a pygame surface, specifically, it rotates the original images of the rocket to the angle its currently at. The rotate method takes in the angle in degrees, so it converts the angle from radians into degrees before rotating each surface. It then calculates the new size of the surface as this changes depending on the amount of rotation.

```
636
637     def __CalculateNewPosition(self):
638         """Calculate the new position of the rocket due to moving.
639         """
640
641         # Calculate change in x and y depending on Velocity and angle using trigonometry
642         TempAngle = self.__Angle + 0.5*math.pi
643         TempXDistance = math.cos(TempAngle) * self.__Velocity
644         TempYDistance = math.sin(TempAngle) * self.__Velocity
645
646         # Calculate new x and y co-ordinates
647         TempX = self.__Position[0] - TempXDistance
648         TempY = self.__Position[1] - TempYDistance
649
650         # Set the new position of the rocket
651         self.__Position = (TempX, TempY)
652
```

This method is used to calculate the new position of the rocket by using its velocity. First it takes the angle in radians and adds 90 degrees to it (1/2 * pi), this is because the angle must be from the x-axis not directly up which is how it currently is. It then uses trigonometry with sine and cosine to calculate how much the position has changed. This change in x and y are then subtracted or added to the position of the rocket.

```
652
653     def GetSurface(self):
654         """Get the surface of the rocket.
655
656         Returns:
657             Pygame Surface: Surface containing the rocket image.
658         """
659
660         # If user is pressing forwards...
661         if self.__CurrentState == "on":
662             return self.__CurrentSurfaceOn
663
664         # If user isn't pressing forwards...
665         else:
666             return self.__CurrentSurfaceOff
667
668     def GetTopLeftPosition(self):
669         """Calculates and gets top left corner of the rocket.
670
671         Returns:
672             Position (tuple): (X, Y) of top left corner.
673         """
674
675         # Calculate top left corner position based off centre position
676         TempX = self.__Position[0] - self.__Size[0]/2
677         TempY = self.__Position[1] - self.__Size[1]/2
678         return (TempX, TempY)
679
```

The GetSurface method returns the surface of the rocket depending on its current state.

The GetTopLeftPosition method takes the rockets position and subtracts ½ of its width and height from the x and y to get the position of the top left corner which is needed for drawing the surface to the screen in the correct position.

```
679
680     def GetPosition(self):
681         """Get centre position of the rocket.
682
683         Returns:
684             Position (tuple): (X, Y) of the centre.
685         """
686
687         return self.__Position
688
689     def GetState(self):
690         """Get state of the rocket.
691
692         Returns:
693             string: 'on' or 'off'
694         """
695
696         return self.__CurrentState
697
```

These methods simply return the Position and State of the rocket and are needed to better encapsulate the rocket and provide a simpler interface.

```
697
698     def Reset(self, Position:tuple):
699         """Reset rocket to a new position.
700
701         Args:
702             Position (tuple): (X, Y) of new position.
703         """
704
705         # Set new position
706         self.__Position = Position
707
708         # Reset all variables to 0 so rocket doesn't carry on moving
709         self.__Angle = 0
710         self.__AngleVelocity= 0
711         self.__Velocity = 0
712         self.__CurrentState = "off"
713
714     def SetState(self, State:str):
715         """Set the state of the rocket.
716
717         Args:
718             State (str): 'off' or 'on'
719         """
720
721         self.__CurrentState = State
722
```

The Reset method is used to move the rocket to a certain location as well as reset all its attributes back to their default values. This is used when the press ‘r’ or they answer a question correctly.

The SetState method simply sets the current state to the parameter specified.

```
722
723 class PlanetManager:
724     def __init__(self, RES):
725         self.__RES = RES
726
727         # The range of sizes possible [MinRadius, MaxRadius]
728         self.__RadiusRange = CONSTS["gm-planet-radius-range"]
729
730         # Calculate min and max positions for x and y
731         # to stop a planet from being slightly off the screen
732         self.__MinXPosition = self.__RadiusRange[1]
733         self.__MaxXPosition = self.__RES[0] - self.__RadiusRange[1]
734         self.__MinYPosition = self.__RadiusRange[1]
735         self.__MaxYPosition = self.__RES[1] - self.__RadiusRange[1]
736
737     def __RandomPosition(self):
738         """Generate a random position within the min and max values.
739
740         Returns:
741             Position (tuple): (X, Y) of the random position.
742         """
743
744         # Generate random position
745         TempX = random.randint(self.__MinXPosition, self.__MaxXPosition)
746         TempY = random.randint(self.__MinYPosition, self.__MaxYPosition)
747         return (TempX, TempY)
748
```

The PlanetManager class is used to encapsulate everything to do with the planets within a single class and provide a simpler interface for the main loop. The Radius Range is a list which has the minimum radius as well as the maximum radius. Attributes are then created for the minimum and maximum x and y values which are used to make sure the planet is fully visible and isn't touching or over an edge of the screen.

The RandomPosition method is used to generate a random position on the screen within the min and max x and y values.

```
748
749     def CreatePlanets(self):
750         """Create new planets
751         """
752
753         self.__Planets = []
754         TempFar = False
755
756         # This loop makes sure planets aren't very close to each other
757         while not TempFar:
758             # Generate 2 random positions
759             TempPos1 = self.__RandomPosition()
760             TempPos2 = self.__RandomPosition()
761             # Calculate distance between the 2 points
762             TempDistance = math.sqrt((TempPos2[0] - TempPos1[0])**2 + (TempPos2[1] -
    TempPos1[1])**2)
763             # If the distance is large enough...
764             if TempDistance > self.__RES[0]*0.6:
765                 TempFar = True
766
767         # Create 2 random planets with the positions generated
768         for TempPos in [TempPos1, TempPos2]:
769             TempRadius = random.randint(self.__RadiusRange[0], self.__RadiusRange[1])
770             self.__Planets.append(Planet(TempPos, TempRadius))
771
772         # Create a transparent pygame surface
773         self.__Surface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
774
775         # For both planets...
776         for TempPlanet in self.__Planets:
777             TempPlanet:Planet
778
779             # Add the planets to the pygame surface
780             TempPlanetSurface = TempPlanet.GetSurface()
781             TempPlanetTopLeftPosition = TempPlanet.GetTopLeftPosition()
782             self.__Surface.blit(TempPlanetSurface, TempPlanetTopLeftPosition)
783
```

The Create Planets method is used to generate 2 new planets. First it overwrites any existing planets by creating an empty list called Planets. It then generates 2 random positions. To make sure that the planets aren't very close to each other, it calculates the distance between them and if they are too close it will generate new positions until they are far enough apart. It then generates random sizes for the planets using the min and max radiuses and creates 2 new planet objects in the Planets list. It then gets the surface of each planet and adds them to the Planet Manager's surface.

```
783
784     def CheckCollision(self, Position:tuple):
785         """Check if a position is colliding with a planet.
786
787         Args:
788             Position (tuple): (X, Y) of point to check.
789
790         Returns:
791             int: 0 or 1 indicating which planet the point collides with.
792             False: if point doesn't collide with either planet.
793         """
794
795         # For each planet...
796         for i, TempPlanet in enumerate(self.__Planets):
797             TempPlanet:Planet
798
799             # If point collides with the planet...
800             if TempPlanet.CheckCollision(Position):
801                 return i
802
803         # If point doesn't collide with either planet...
804         return False
805
```

The CheckCollision method is used to check if the rocket is colliding with either planet and if it is, then it returns the index of the planet its colliding with. It does this by looping through each planet and calling it's Check Collision method.

```
805
806     def GetSurface(self):
807         """Get the surface of the planet manager.
808
809         Returns:
810             Pygame Surface: Surface containing both planets.
811         """
812
813         return self.__Surface
814
815     def GetCentrePositionOf(self, PlanetNumber:int):
816         """Get the centre position of a planet.
817
818         Args:
819             PlanetNumber (int): 0 or 1 of planet.
820
821         Returns:
822             Position (tuple): (X, Y) of centre position of planet.
823         """
824
825         TempPlanet = self.__Planets[PlanetNumber]
826         TempPlanet:Planet
827
828         return TempPlanet.GetCentrePosition()
829
```

These methods simply return the Surface of the Planet Manager or the Centre Position of a specific planet.

Coding the Question Manager

```
829
830 class QuestionManager:
831     def __init__(self, RES:tuple, Questions:list):
832         self.__RES = RES
833         self.__Questions = Questions
834
835         self.__NumberOfQuestionsNeeded = 8
836         self.__NumberOfQuestionsCorrect = 0
837         # Create a layout manager
838         self.__LayoutManager = LayoutManager(self.__RES, (12,14), 30)
839         self.__CurrentQuestion = None
840
```

The Question Manager class is used to create questions, create textboxes, create buttons, and manage the user's inputs when they are answering a question. It takes in the Resolution of the screen and a list of Questions as parameters as well as then creating attributes to store the number of questions needed to finish the game and the number of questions the user has answered correctly. Within the class it also creates a Layout Manager object which is used to contain the textboxes and buttons.

```
840
841     def CreateQuestion(self):
842         """Create the next question.
843
844         Returns:
845             True: If question created successfully.
846             False: If number of questions needed has been reached.
847         """
848
849         # If number of questions needed has been reached...
850         if self.CheckNumberOfQuestionsCorrect() or len(self.__Questions) == 0:
851             return False
852
853         # Delete any current items from the layout
854         self.__LayoutManager.RemoveTextBox("question")
855         self.__LayoutManager.RemoveButton("0")
856         self.__LayoutManager.RemoveButton("1")
857         self.__LayoutManager.RemoveButton("2")
858         self.__LayoutManager.RemoveButton("3")
859
```

When the Create Question method is called it first checks if the user has already answered enough question or if there aren't any more questions and if either of these are True then it returns False. After that it deletes any existing textboxes and buttons using their IDs.

```
859
860         # Copy the next question to the current question
861         self.__CurrentQuestion = self.__Questions[0].copy()
862         # Delete that question from the question bank so that it doesn't get duplicates
863         del self.__Questions[0]
864
865         # Add the question textbox
866         self.__LayoutManager.AddTextBox("question", (1.1,1.1), (9.8,3.8),
    self.__CurrentQuestion[0], "L")
867         # Add the answer buttons
868         self.__LayoutManager.AddButton("0", (1.1,5.1),  (9.8,1.8), self.__CurrentQuestion[2],
    "L")
869         self.__LayoutManager.AddButton("1", (1.1,7.1),  (9.8,1.8), self.__CurrentQuestion[3],
    "L")
870         self.__LayoutManager.AddButton("2", (1.1,9.1),  (9.8,1.8), self.__CurrentQuestion[4],
    "L")
871         self.__LayoutManager.AddButton("3", (1.1,11.1), (9.8,1.8), self.__CurrentQuestion[5],
    "L")
872
873         # Set the index of the correct answer
874         self.__CurrentCorrectAnswerIndex = self.__CurrentQuestion[1]
875         return True
876
```

It then gets the current question into a variable called CurrentQuestion so it can then remove the question from the question bank. It then creates a textbox which displays the question and buttons which store each answer option. It also creates an attribute to store the index of the correct answer.

```
876
877     def CheckClick(self, MousePosition:tuple):
878         """Check if the mouse has clicked the correct answer.
879
880         Args:
881             MousePosition (tuple): (X, Y) of the mouse.
882
883         Returns:
884             True: If correct answer clicked.
885             False: If incorrect answer clicked.
886             None: If no button clicked.
887         """
888
889         # Index of button clicked
890         Index = self.__LayoutManager.CheckMouseCollision(MousePosition)
891
892         # If no button clicked...
893         if Index == None:
894             return None
895
896         # If button clicked is correct answer...
897         if int(Index) == self.__CurrentCorrectAnswerIndex:
898             self.__NumberOfQuestionsCorrect += 1
899             return True
900
901         # If incorrect answer clicked
902         else:
903             return False
904
```

Check Click is used when the user clicks the mouse, and the program needs to see if they clicked a button and if it was the correct answer. First it checks if they did click any of the buttons by using the LayoutManagers Collision method. If none of the buttons were clicked it returns None, but if a button was clicked and it's the correct index then it adds 1 to the number of questions correct and returns True, anything else returns False.

```
904
905     def CheckNumberOfQuestionsCorrect(self):
906         """Check if user has answered the required number of questions correctly.
907
908         Returns:
909             True: If all questions needed have been answered.
910             False: If not enough questions have been answered.
911         """
912
913         # If number of questions correct matches number of questions needed...
914         if self.__NumberOfQuestionsCorrect == self.__NumberOfQuestionsNeeded:
915             return True
916
917         # If they don't match...
918         return False
919
```

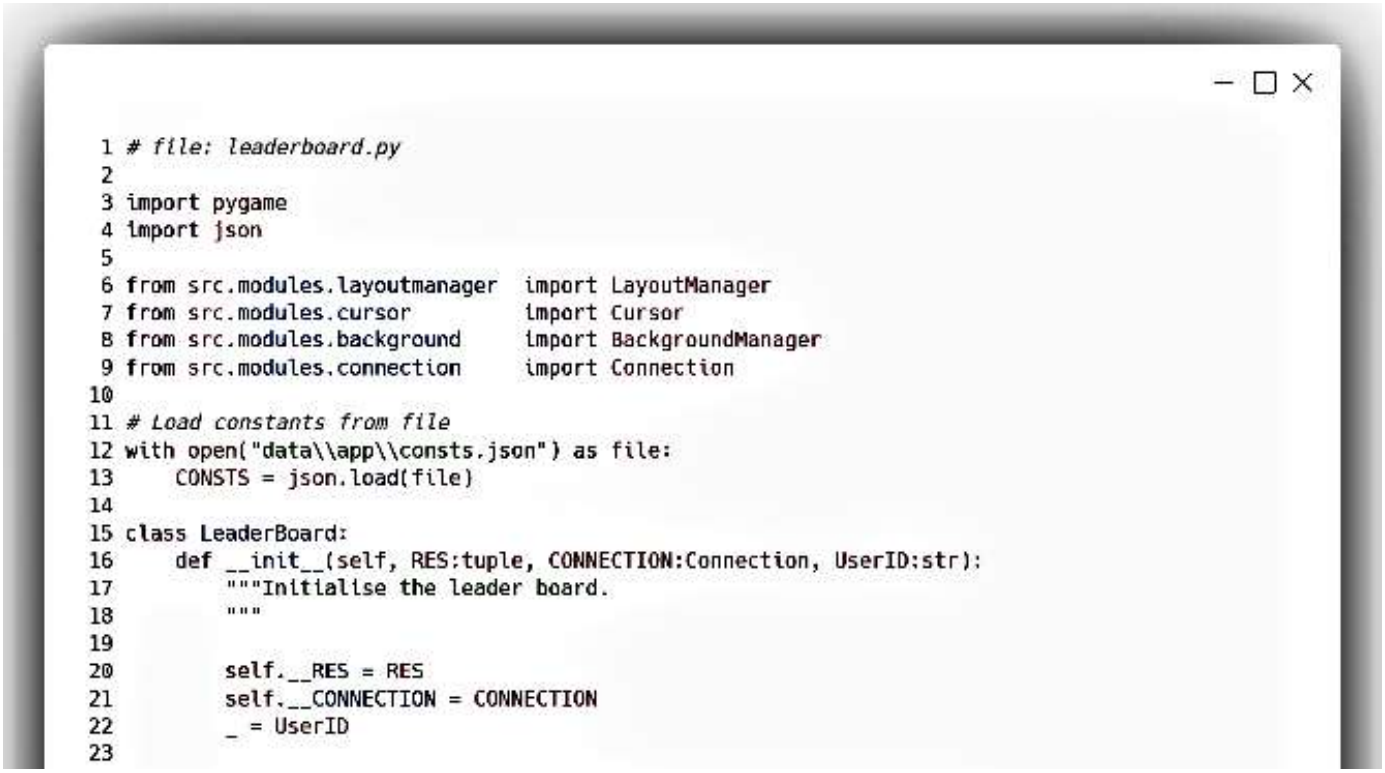
This method is used to return True or False depending on if the user has answered enough questions correctly or not.

```
919
920     def GetSurface(self, MousePosition:tuple):
921         """Get the surface of the questions.
922
923         Args:
924             MousePosition (tuple): (X, Y) of the mouse.
925
926         Returns:
927             Pygame Surface: Surface containing the question.
928         """
929
930         return self.__LayoutManager.GetSurface(MousePosition)
931
```

This method is used to get the surface which contains all the buttons and textboxes. It gets this from the Layout Manger and uses the mouse position to outline any buttons that the mouse is hovering over.

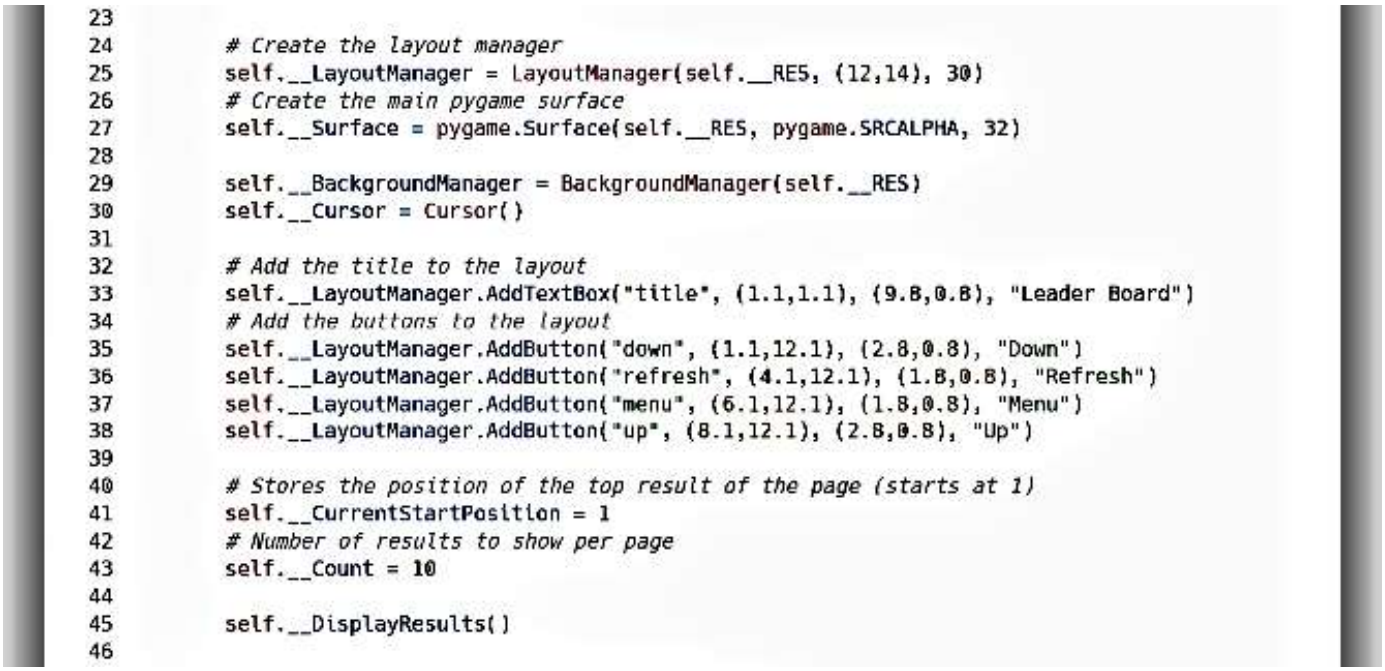
Coding leaderboard.py

22nd October 2022: The leader board is an important part of the program as it is what will make the game competitive and make user's want to play the game to try and beat their friends or their high score.



```python
# file: leaderboard.py

import pygame
import json

from src.modules.layoutmanager  import LayoutManager
from src.modules.cursor         import Cursor
from src.modules.background     import BackgroundManager
from src.modules.connection     import Connection

# Load constants from file
with open("data\\app\\consts.json") as file:
    CONSTS = json.load(file)

class LeaderBoard:
    def __init__(self, RES:tuple, CONNECTION:Connection, UserID:str):
        """Initialise the leader board.
        """

        self.__RES = RES
        self.__CONNECTION = CONNECTION
        _ = UserID

```

It starts out the same as all the other sections by important the needed modules and creating the LeaderBoard class.



```python

        # Create the layout manager
        self.__LayoutManager = LayoutManager(self.__RES, (12,14), 30)
        # Create the main pygame surface
        self.__Surface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)

        self.__BackgroundManager = BackgroundManager(self.__RES)
        self.__Cursor = Cursor()

        # Add the title to the layout
        self.__LayoutManager.AddTextBox("title", (1.1,1.1), (9.8,0.8), "Leader Board")
        # Add the buttons to the layout
        self.__LayoutManager.AddButton("down", (1.1,12.1), (2.8,0.8), "Down")
        self.__LayoutManager.AddButton("refresh", (4.1,12.1), (1.8,0.8), "Refresh")
        self.__LayoutManager.AddButton("menu", (6.1,12.1), (1.8,0.8), "Menu")
        self.__LayoutManager.AddButton("up", (8.1,12.1), (2.8,0.8), "Up")

        # Stores the position of the top result of the page (starts at 1)
        self.__CurrentStartPosition = 1
        # Number of results to show per page
        self.__Count = 10

        self.__DisplayResults()

```

The constructor method then goes on to create the Layout Manger, surface, Background Manager, and Cursor objects. It then adds the title of Leader Board and the navigation buttons to the layout manager. The Current Start Position stores the ordinal position of the result that will be displayed at the top of the screen. This starts as 1 and if they move down a page, it will become 11. The Count attribute stores how many scores will be displayed per page. Lastly it calls the Display Results method.

```
46
47      def __LoadResults(self, StartPosition:int):
48          """Load the results from the server and database.
49
50          Args:
51              StartPosition (int): Position of the first result to show.
52
53          Returns:
54              Results (list): List containing the formatted results.
55          """
56
57          # Send request for the reseults to the server
58          self.__CONNECTION.Send(f"105##{StartPosition}##{self.__Count}")
59          # Receive results from the server
60          TempResults = self.__CONNECTION.Receive()
61          # Convert the raw response into a list of results
62          TempResults = self.__ConvertStringToScores(TempResults)
63
64          Results = []
65          # Convert the raw results into a list of strings to display
66          for i, Result in enumerate(TempResults):
67              string = ""
68              string += str(i + StartPosition) + "  -  "
69              string += self.__CovertNanosecondsToString(int(Result[1])) + "  -  "
70              string += Result[0] + " - " + Result[2] + " " + Result[3]
71              Results.append(string)
72          return Results
73
```

The Load Results method is used to get the scores from the server and database. It first sends a request to the server using the Query ID of 105 giving it the parameters of the Start Position and the number of results to get. Next it receives these scores and uses the Convert String To Scores method to get a list of the scores. It then needs to convert the score data into a string that can be inserted into a textbox. It does this by manipulating a string by appending to the end of it the next piece of data. It also must convert the score, which is stored in nanoseconds, into a string of minutes, seconds, and milliseconds.

```
73
74      def __ConvertStringToScores(self, String:str):
75          """Convert the raw response from the server into a list of results.
76
77          Args:
78              String (str): The raw string data.
79
80          Returns:
81              Results (list): List containing the data.
82          """
83
84          # Splits the data by the seperator "&&"
85          SplitString = String.split("&&")
86          Results = []
87          try:
88              while len(SplitString) != 0:
89                  TempScore = []
90
91                  # Group the items into groups of 2 items
92                  for i in range(4):
93                      TempScore.append(SplitString[i])
94
95                  # Delete the grouped items
96                  for i in range(4):
97                      del SplitString[0]
98                  # Add the list of grouped items to the results list
99                  Results.append(TempScore)
100         # If an error occurs...
101         except:
102             pass
103         return Results
104
```

The Convert String To Scores method works similarly to the Convert String To Questions method as in the game.py file but groups the items into groups of size 4 instead of 6.

```
104
105     def __CovertNanosecondsToString(self, Time):
106         """Converts the time into minutes seconds and milliseconds.
107
108         Returns:
109             string: the time like this 'minutes:seconds.milliseconds'
110         """
111
112         # Convert time to milliseconds
113         Time = Time/1000000
114         # Calculate minutes, seconds and milliseconds
115         Minutes = Time // 60000
116         remainder = Time % 60000
117         Seconds = remainder // 1000
118         Milliseconds = remainder % 1000
119
120         # Round everything to an integer
121         # Make each number take up 2 characters
122         Minutes = str(round(Minutes)).zfill(2)
123         Seconds = str(round(Seconds)).zfill(2)
124         Milliseconds = str(round(Milliseconds))[:2].zfill(2)
125         return f"{Minutes}:{Seconds}.{Milliseconds}"
126
```

The convert nanoseconds to string method is the same as the method used in the timer but isn't inherited as it may want to be modified to output a more accurate score as it isn't restricted to the size of the timer like it is in the game.

```
126
127     def __DisplayResults(self):
128         """Load and add the results to the layout manager.
129         """
130
131         # Load the results
132         Results = self.__LoadResults(self.__CurrentStartPosition)
133
134         # Remove current results from the layout
135         for TempNum in range(10):
136             self.__LayoutManager.RemoveTextBox(f"result-{TempNum}")
137
138         try:
139             # Add the new results
140             for TempNum in range(10):
141                 self.__LayoutManager.AddTextBox(f"result-{TempNum}", [1.1,2.1+TempNum],
    [9.8,0.8], Results[TempNum], "L")
142
143         # If there isn't enough results it will be skipped
144         except:
145             return False
146         return True
147
```

The Display Results method first loads the results and then removes all the current results using the ID of "result-0", "result-1", "result-2", … . Once it has done this, it then attempts to create a textbox for each score however this will not always be possible if their aren't enough scores to fill the page. When this happens it simply returns False so that the program knows there isn't another page.

```
147
148     def Run(self, WINDOW:pygame.Surface, CLOCK:pygame.time.Clock):
149         """Run the leader board loop
150
151         Returns:
152             NextState (str): The next state to switch to.
153         """
154
155         self.__NextState = "main-menu"
156         self.__LastPage = False
157         self.__Fading = False
158         self.__Run = True
159
160         # Start the loop
161         while self.__Run:
162
163             # Gets position of the mouse from pygame
164             MousePosition = pygame.mouse.get_pos()
165             # Get any keys pressed
166             KeysPressed = pygame.key.get_pressed()
167
168             # If user presses ESC...
169             if KeysPressed[pygame.K_ESCAPE]:
170                 # Return user to the main menu
171                 self.__NextState = "main-menu"
172                 self.__FadeMenu()
173
```

The Run method of the Leader Board starts like every other Run method by creating various attributes that will be needed later. The only one that is different is the Last Page attribute which is a Boolean value used to signify if the user is on the last page of results or not. It's used so that the user cannot go past the last page.

The while loop starts the same as all the others by getting the keys pressed and mouse position.

```
173
174             # If user presses up arrow...
175             if KeysPressed[pygame.K_UP]:
176                 if self.__CurrentStartPosition != 1:
177                     self.__CurrentStartPosition -= self.__Count
178                 self.__DisplayResults()
179                 self.__LastPage = False
180
181             # If user presses down arrow...
182             if KeysPressed[pygame.K_DOWN] and not self.__LastPage:
183                 self.__CurrentStartPosition += self.__Count
184                 if not self.__DisplayResults():
185                     self.__LastPage = True
186
187             for event in pygame.event.get():
188                 # If user clicks x to close window
189                 if event.type == pygame.QUIT:
190                     self.__SafeExit()
191
192                 # If user clicks a mouse button
193                 if event.type == pygame.MOUSEBUTTONUP:
194                     # Detect if they clicked a button
195                     TempButtonClicked = self.__LayoutManager.CheckMouseCollision(MousePosition)
196
197                     if TempButtonClicked == "menu":
198                         self.__NextState = "main-menu"
199                         self.__FadeMenu()
200
201                     elif TempButtonClicked == "down" and not self.__LastPage:
202                         self.__CurrentStartPosition += self.__Count
203                         if not self.__DisplayResults():
204                             self.__LastPage = True
205
206                     elif TempButtonClicked == "up":
207                         if self.__CurrentStartPosition != 1:
208                             self.__CurrentStartPosition -= self.__Count
209                         self.__DisplayResults()
210                         self.__LastPage = False
211
212                     elif TempButtonClicked == "refresh":
213                         self.__CurrentStartPosition = 1
214                         self.__DisplayResults()
215                         self.__LastPage = False
216
```

It then goes on to check what keys have been pressed. The user can use the arrow keys to move up and down pages but it also makes sure that it isn't the last page if the user is attempting to move down a page. If the user clicks, then it checks the mouse is colliding with any of the buttons which is done using the Layout Manager. If it detects a click then it triggers the corresponding if statement which will change the start position and call the Display Results method to get the new results to display.

```
216
217             # Add background to the surface
218             TempSurface = self.__BackgroundManager.Update(self.__Fading)
219             self.__Surface.blit(TempSurface, (0,0))
220
221             # Add the layout to the surface
222             TempSurface = self.__LayoutManager.GetSurface(MousePosition)
223             self.__Surface.blit(TempSurface, (0,0))
224
225             # Add the cursor to the surface
226             TempSurface = self.__Cursor.GetSurface()
227             TempPosition = self.__Cursor.GetPosition(MousePosition)
228             self.__Surface.blit(TempSurface, TempPosition)
229
```

It then goes on to add the background, textboxes and buttons, and the cursor to the main surface.

```
229
230             # If menu is fading to a different state...
231             if self.__Fading:
232                 # Overlay the screen with the fading surface
233                 # Increases fade level and sets FadeSurface to that level
234                 self.__FadeLevel += CONSTS["fade-amount"]
235                 if self.__FadeLevel > 255:
236                     self.__FadeLevel = 255
237                 # Fill the fade surface with current fadelevel as the alpha channel
238                 self.__FadeSurface.fill((0,0,0,self.__FadeLevel))
239                 # Adds the FadeSurface to the top of the main surface
240                 self.__Surface.blit(self.__FadeSurface, (0, 0))
241                 # If it's fully faded out...
242                 if self.__FadeLevel >= 255:
243                     # Hand control back to main.py
244                     return self.__NextState
245
246             # Adds the surface to the main window
247             WINDOW.blit(self.__Surface, (0,0))
248
249             CLOCK.tick(CONSTS["fps"])
250             pygame.display.update()
251         return self.__NextState
252
```

As with all the other sections, if the program is fading to a different section it uses this algorithm to slowly fade the screen to black before switching.

```
252
253     def __FadeMenu(self):
254         """Start fading the window to a different state.
255         """
256
257         self.__Fading = True
258         self.__FadeSurface = pygame.Surface(self.__RES, pygame.SRCALPHA, 32)
259         self.__FadeLevel = 0
260         self.__FadeSurface.fill((0, 0, 0, self.__FadeLevel))
261
262     def __SafeExit(self):
263         """Safely exit the program
264         """
265
266         exit()
267
```

These last methods are the same as in the other sections of the program.

Coding server.py

7th November 2022: The server is a vital part of the program as without it the program is not playable. Therefore, it needs to be able to manage errors wherever they arise. It will be the gateway for the client program to insert and get data from the database while also protecting the database from possible attacks.

```python
# file: server.py

import socket
import threading
from time import gmtime, strftime

import query_manager

def Log(Type:str, Description:str):
    """Log an event to the terminal.

    Args:
        Type (str): Type of log. E.g. 'SERVER', 'ERROR'
        Description (str): What the log is
    """

    # Get current time as a string
    time = strftime("%Y-%m-%d %H:%M:%S", gmtime())
    # Print the time and log
    print(f"{time} [{Type}] - {Description}")

```

The server starts by important the socket, threading and time modules as well as importing the Query Manager. A simple function called Log is then defined which is used to create a timestamped log of everything that happens on the server. This is very important for debugging as well making sure everything is working correctly. It uses the strftime() and gmtime() functions of the time module to get the current GMT time and format it into a custom format of:

$$"2022-12-28\ \ 18{:}57{:}25"$$

It will take also use a type and description to make the output easier to understand.

```
2023-03-07 20:18:56 [SERVER] - Server started with IP: 192.168.1.101 and PORT: 5050
2023-03-07 20:19:06 [SERVER] - New connection with IP: 192.168.1.102
```

```
2023-03-07 20:19:16 [REQUEST] - From: 192.168.1.102 Request: 106##henrywalker1
2023-03-07 20:19:16 [REQUEST HANDLER] - From IP: 192.168.1.102 - Checked username: henrywalker1
2023-03-07 20:19:16 [RESPONSE] - To: 192.168.1.102 Response: True
```

```
2023-03-07 20:19:30 [REQUEST] - From: 192.168.1.102 Request: 100##henrywalker1##Pass123##Henry##Walker
2023-03-07 20:19:31 [REQUEST HANDLER] - From IP: 192.168.1.102 - Added user. Username: henrywalker1
2023-03-07 20:19:31 [RESPONSE] - To: 192.168.1.102 Response: None
```

```
2023-03-07 20:19:31 [REQUEST] - From: 192.168.1.102 Request: 103##henrywalker1##Pass123
2023-03-07 20:19:31 [REQUEST HANDLER] - From IP: 192.168.1.102 - Checked login with username: henrywalker1
2023-03-07 20:19:31 [RESPONSE] - To: 192.168.1.102 Response: 8
```

Above are screenshots which shows the use of the Log function when the server is running. It shows a client connecting, creating an account, and then being logged in. As shown above, it can be split into blocks of groups of 3, the request, the processing of the request, and the response.

```
21
22 class Server:
23     def __init__(self, HEADERLENGTH:int):
24         """Initialise the server and create the socket.
25         """
26
27         self.__HEADERLENGTH = HEADERLENGTH
28         # Get the IP of the computer running the server
29         self.__IP = socket.gethostbyname(socket.gethostname())
30         self.__PORT = 5050
31         # Create a socket using IPv4 and TCP
32         self.__ServerSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
33         # This allows the socket to override any old sockets of the same address
34         self.__ServerSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
35         # Link the IP and PORT to the socket
36         self.__ServerSocket.bind((self.__IP, self.__PORT))
37         # Passively waits for a connection in the background
38         self.__ServerSocket.listen()
39         Log("SERVER", f"Server started with IP: {self.__IP} and PORT: {self.__PORT}")
40
```

When the server is created, it works similarly to how the Connection module works. What is different is that it gets the IP address of the device it is running on, and this is then bound to the socket. Then at the end it uses the listen method to start listening for new connections.

```
40
41     def __ReceiveData(self, Connection:socket.socket):
42         """Receives data from the socket.
43
44         Args:
45             Connection (socket): The socket to receive data from
46
47         Returns:
48             False: Connection has been closed
49             str: Data which has been received
50         """
51
52         try:
53             # Receive the header in binary
54             DataHeader = Connection.recv(self.__HEADERLENGTH)
55             # Convert data length in binary into decimal
56             DataLength = int(DataHeader.decode('utf-8').strip())
57             # Receive the data and convert to a string
58             Data = Connection.recv(DataLength).decode('utf-8')
59             return Data
60
61         # If cannot receieve data socket has been closed
62         except:
63             return False
64
```

The receive data method will be used to receive data from a connection, decode the data and return it as a string. This works exactly the same as the function in the Connection class.

```
64
65      def __SendData(self, Connection:socket.socket, Data:str):
66          """Send data over the socket.
67
68          Args:
69              Connection (socket): The socket to send the data over.
70              Data (str): The data to send.
71          """
72
73          try:
74              # Prepare data and header and send them
75              ByteData = Data.encode('utf-8')
76              # Generate the header and turn into binary
77              ByteHeader = f"{len(ByteData):<{self.__HEADERLENGTH}}".encode('utf-8')
78              # Send the header and data over the socket
79              Connection.send(ByteHeader + ByteData)
80
81          # If an error occurs
82          except:
83              Log("ERROR", "In SendData")
84
```

Much like the receive data method, the Send Data method is as well the same as the method in the Connection class with the slight difference that if there is an error, it logs it.

```
84
85      def __HandleClient(self, Connection:socket.socket, ClientAddress:str):
86          """A new instance of this will be created for each client.
87
88          Args:
89              Connection (socket): The socket between the server and the client.
90              ClientAddress (str): The IP address of the client.
91          """
92
93          Run = True
94          while Run:
95              # Wait until the client sends a request
96              Request = self.__ReceiveData(Connection)
97              Log("REQUEST", f"From: {ClientAddress} Request: {Request}")
98
99              # If the request is to close the socket...
100             if Request == "CloseSocket" or Request == False or Request == "False":
101                 Run = False
102                 Log("CLIENT", f"Socket closed with IP: {ClientAddress}")
103
104             # If it's not to close the socket...
105             if Run:
106                 # Handle the request and get the response
107                 Response = self.__HandleRequest(Request, ClientAddress)
108                 Log("RESPONSE", f"To: {ClientAddress} Response: {Response}")
109                 # Send the response back to the client
110                 self.__SendData(Connection, str(Response))
111
```

The Handle Client method will be executed for each client connected to the server. This means if there are 4 clients, this method will be running 4 times simultaneously. This will be explained below. It uses a while loop to wait for a request, and then when it receives a request, handle it. The handling of the request will produce a response either data or the string of “None”. This response is then sent back to the client and the loop waits for the next request from the client.

```
111
112     def __HandleRequest(self, Request:str, ClientAddress:str):
113         Response = None
114         # ClientAddress shortened to ca
115         ca = ClientAddress
116         try:
117             # Split request by the seperator "##" into a list
118             # Shortened to sr meaning SplitRequest
119             sr = Request.split("##")
120 
121             # Matches the request using the first item in the request list
122 
123             # Add user
124             if sr[0] == "100":
125                 query_manager.AddUser(sr[1], sr[2], sr[3], sr[4])
126                 Log("REQUEST HANDLER", f"From IP: {ca} - Added user. Username: {sr[1]}")
127 
128             # Add question
129             elif sr[0] == "101":
130                 query_manager.AddQuestion(sr[1], sr[2], sr[3], sr[4], sr[5])
131                 Log("REQUEST HANDLER", f"From IP: {ca} - Added question")
132 
133             # Add score
134             elif sr[0] == "102":
135                 query_manager.AddScore(sr[1], sr[2])
136                 Log("REQUEST HANDLER", f"From IP: {ca} - Added score. Username: {sr[1]} Score:
    {sr[2]}")
137 
```

The Handle Request function takes the Request and the Client Address as parameters. At the moment the request is a single string, so it first separates this into a list by splitting it using the ## separator explained in the Design section. The first item of the list, sr[0], is the Query ID. The program uses if statements to send the rest of the request's data to the correct function within the query manager. At each stage it logs the request using the Clients IP Address and what the request was for.

```
137 
138             # Check login
139             elif sr[0] == "103":
140                 Response = query_manager.CheckLogin(sr[1], sr[2])
141                 Log("REQUEST HANDLER", f"From IP: {ca} - Checked login with username: {sr[1]}")
142 
143             # Get random questions
144             elif sr[0] == "104":
145                 Response = query_manager.GetRandomQuestions(int(sr[1]))
146                 Log("REQUEST HANDLER", f"From IP: {ca} - Got questions")
147 
148             # Get scores
149             elif sr[0] == "105":
150                 Response = query_manager.GetScores(int(sr[1]), int(sr[2]))
151                 Log("REQUEST HANDLER", f"From IP: {ca} - Got scores")
152 
153             # Check username
154             elif sr[0] == "106":
155                 Response = query_manager.CheckUsername(sr[1])
156                 Log("REQUEST HANDLER", f"From IP: {ca} - Checked username: {sr[1]}")
157 
158             # Query not matched...
159             else:
160                 Log("REQUEST HANDLER ERROR", "Query not matched.")
161 
162         # If an error occurs...
163         except:
164             Response = "error"
165         return Response
166 
```

If at any point an error occurs, it will set the response to "error" which will then be sent back to the client where it can be processed if needed.

```
166
167     def Loop(self):
168         """Runs the main loop which handles new clients.
169         """
170
171         while True:
172             # When a new connection is created, accept it
173             Connection, ClientAddress = self.__ServerSocket.accept()
174             # Only the clients IP address is required
175             ClientAddress = ClientAddress[0]
176             Log("SERVER", f"New connection with IP: {ClientAddress}")
177
178             # Start a new thread for the new client
179             ClientThread = threading.Thread(target=self.__HandleClient,
    args=(Connection,ClientAddress,))
180             # Start the new thread
181             ClientThread.start()
182
183 HeaderLength = 10
184 # Create the server
185 myServer = Server(HeaderLength)
186 # Start running the server loop
187 myServer.Loop()
188
```

The Loop method of the Server is what keeps the server running. This method is a loop which when it detects a new connection, it accepts the connection and stores the Connection object as well as the IP address of the client.

So that the server can handle multiple clients at a time, it will make use of threading. This is where the program can run a function multiple times simultaneously without them affecting each other. This is essential for the server to work otherwise it would get stuck until the client sends a request. Within the server, when a new connection starts, it creates a new thread for that client, and it runs the Handle Client function with the connection and IP address as parameters. It then starts the thread once it has been created and then goes back to waiting for a new connection.

The very bottom of the file is where the server is instantiated, and the Loop method is started.

Creating the Database

Before creating the database, I first researched what software I should use for it. The sources I read said the best options are MySQL, SQLite, Microsoft SQL, and PostgreSQL. Out of all of these, I found that MySQL would be most suited to my needs as it is free and has a low impact on the device's performance.

After I installed and setup the software, I created the database using the following code from within Python. Line 5 to 9 connects to the MySQL server running on the same computer. Line 11 creates a cursor object which is used to execute SQL commands and retrieve data from the MySQL database. Line 14 is used to create the database.

```
1 # file: create_database.py
2
3 import mysql.connector
4
5 Connection = mysql.connector.connect(
6   host="localhost",
7   user="root",
8   password="root",
9   database="CosmicQuest")
10
11 Cursor = Connection.cursor()
12
13 # Create Database
14 Cursor.execute("CREATE DATABASE CosmicQuest")
15
```

Once the database was created, I then created each of the tables using the code below. I used PRIMARY KEY to identify what the primary key is in each table. AUTO_INCREMENT is used to generate the UserID when a new record is added to the table. Line 48 is needed to run the commands and create the tables.

```
15
16 # Create User table
17 Cursor.execute("""CREATE TABLE User (
18   UserID INT PRIMARY KEY AUTO_INCREMENT,
19   Username VARCHAR(1024) NOT NULL,
20   PasswordHash VARCHAR(1024) NOT NULL,
21   FirstName VARCHAR(1024) NOT NULL,
22   LastName VARCHAR(1024) NOT NULL
23   );""")
24
25 # Create Score table
26 Cursor.execute("""CREATE TABLE Score (
27   ScoreID INT PRIMARY KEY AUTO_INCREMENT,
28   UserID INT NOT NULL,
29   Score INT NOT NULL
30   );""")
31
32 # Create Question table
33 Cursor.execute("""CREATE TABLE Question (
34   QuestionID INT PRIMARY KEY AUTO_INCREMENT,
35   QuestionText VARCHAR(2048) NOT NULL,
36   LevelID INT NOT NULL
37   );""")
38
39 # Create Answer table
40 Cursor.execute("""CREATE TABLE Answer (
41   AnswerID INT PRIMARY KEY AUTO_INCREMENT,
42   QuestionID INT NOT NULL,
43   AnswerText VARCHAR(2048) NOT NULL,
44   IsCorrect TINYINT NOT NULL
45   );""")
46
47 # Execute all sql commands to the database
48 Connection.commit()
49
```

Coding query_manager.py

29th November 2022: The Query Manager is not a file that runs at all like the others, instead it stores all the functions which are used to insert and request data from the database.

```
1 # file: query_manager.py
2
3 import mysql.connector
4 import random
5 import bcrypt
6
7 # Connect to the database
8 Connection = mysql.connector.connect(
9     host="localhost",
10    user="root",
11    password="root",
12    database="CosmicQuest")
13
14 Cursor = Connection.cursor()
15
```

As the database is created using MySQL, I used the MySQL Python module to be able to send queries to the database from Python. The random module is needed when the client requests random questions. The bcrypt module is what I used to hash the passwords. The reason I didn't code my own hashing algorithm is because this would be very insecure whereas the bcrypt module has been rigorously tested to an industry-standard.

At the start of the file, it connects to the database which is stored on the same device as the server. It then creates a cursor in the Connection which will be used to send requests and get responses from the database.

```
15
16 def ConvertToString(List:list):
17     """Convert a list of lists into a string.
18
19     Args:
20         List (list): The list to convert.
21
22     Returns:
23         str: Containing all the data in the list seperated by '&&'
24     """
25
26     # Create an empty string
27     string = ""
28     # For each list within the main list...
29     for List2 in List:
30         # For each item within the current list2...
31         for Item in List2:
32             # Add the item to the string and seperate it by &&
33             string += str(Item) + "&&"
34     # Remove the last && from the string
35     string = string[:-2]
36     return string
37
```

The Convert To String function will be used extensively throughout all the queries as it converts the lists of data into a single string separated by the && separator. It does this by looping through each list and then the list inside the original list and adding each item to a string. At the end, it removes the last 2 characters from the string as it will always end with an extra && which should not be there.

```
37
38 def HashPassword(Password:str):
39     """Hash a password.
40
41     Args:
42         Password (str): The password as a string
43
44     Returns:
45         Hash (str): The hash of the password
46     """
47
48     # Hash the password using bcrypt
49     Hash = bcrypt.hashpw(Password.encode("utf-8"), bcrypt.gensalt())
50     # Return the hashed password as a string
51     return Hash.decode("utf-8")
52
```

The Hash Password function will be used to convert a password in string format into a hash of the password which can then be stored in the database. It uses the hash password method of the bcrypt module. This method takes in the password (which must be converted to binary) as well as a salt. The salt is used to create randomised noise within the hash so that lookup tables or dictionaries cannot be used to "reverse" the hash. The Hash is then converted from binary into a string.

```
52
53 def CheckPassword(Password:str, Hash:str):
54     """Check a password against a hash.
55
56     Args:
57         Password (str): The password to check
58         Hash (str): The hash to check against
59
60     Returns:
61         True: If they match
62         False: If they don't match
63     """
64
65     # Check if the password matches the stored hash using bcrypt
66     return bcrypt.checkpw(Password.encode("utf-8"), Hash.encode("utf-8"))
67
```

The Check Password function works similarly to the Hash Password but is used to compare if an entered password matches the stored password. Again, this uses the bcrypt method of checkpw() which takes in the password and the hash in binary. The function will then return True or False depending on if the passwords match or not.

```
67
68 # 100 - To add a new user
69 def AddUser(Username:str, Password:str, FirstName:str, LastName:str):
70     """Add a new user.
71
72     Args:
73         Username (str): The users username
74         Password (str): The users password in raw format
75         FirstName (str): Users first name
76         LastName (str): Users last name
77     """
78
79     # Hash the password
80     Hash = HashPassword(Password)
81     Query = "INSERT INTO user (Username, PasswordHash, FirstName, LastName) VALUES (%s, %s, %s,
   %s);"
82     # Send the query to the database with the data
83     Cursor.execute(Query, (Username, Hash, FirstName, LastName))
84     # Executes the query on the database
85     Connection.commit()
86
```

The Add User function is used to add a new user to the database once they have signed up using the program. First it hashes their password, and then creates the query to insert the Username, Hash, and Name into the User table of the database. On line 81, instead of the values there are %s 's. This is done as it allows the query to be in a generic form and the values to be passed in later. On line 83, you can see how the execute function takes in the generic query and the 2nd parameter is a tuple which contains the values to insert. Once the query has been sent to the database using the execute function, the commit function is used to run the query and save the data.

```
86
87 # 101 - To add a new question
88 def AddQuestion(QuestionText:str, CorrectAnswer:str, WrongAnswer1:str, WrongAnswer2:str,
   WrongAnswer3:str):
89     """Add a question to the database
90
91     Args:
92         For each parameter, new lines are represented using "$$"
93         QuestionText (str): The text of the question
94         CorrectAnswer (str): The text of the correct answer
95         WrongAnswer1 (str): The text of the 1st wrong answer
96         WrongAnswer2 (str): The text of the 2nd wrong answer
97         WrongAnswer3 (str): The text of the 3rd wrong answer
98     """
99
100    # Get Current questionID
101    Query = "SELECT COUNT(QuestionID) FROM question"
102    # Send the query to the database with the data
103    Cursor.execute(Query)
104    # Calculate the next question ID
105    NextQuestionID = int(Cursor.fetchall()[0][0])+1
106
107    Query = "INSERT INTO question (QuestionText) VALUES (%s);"
108    # Add the new question to the database
109    Cursor.execute(Query, (QuestionText,))
110    # Executes the query on the database
111    Connection.commit()
112
113    Query = "INSERT INTO answer (QuestionID, AnswerText, IsCorrect) VALUES (%s, %s, %s)"
114    # Insert Correct Answer
115    Cursor.execute(Query, (NextQuestionID, CorrectAnswer, 1))
116    # Insert Wrong answers
117    Cursor.execute(Query, (NextQuestionID, WrongAnswer1, 0))
118    Cursor.execute(Query, (NextQuestionID, WrongAnswer2, 0))
119    Cursor.execute(Query, (NextQuestionID, WrongAnswer3, 0))
120    # Executes the query on the database
121    Connection.commit()
122
```

To add a question to the database, the question, correct answer, and 3 wrong answers must all be provided. To start, the function must work out how many questions are already in the database so it knows what the next Question ID should be. As shown on line 105, the fetchall function is used to get data back from the database and then the indexes of 0 and 0 are used to get the value that's returned. It then adds 1 to this value to get the next question ID.

Now it moves onto inserting the question into the Question table. An important detail is that on line 109, a comma is used after QuestionText because without it, Python does not treat it as a tuple and instead a string. This causes an error because the execute function can only take in a tuple.

Once the question has been inserted, it moves onto inserting the answers into the Answer table. First, it inserts the correct answer and sets the IsCorrect variable to 1 so that it can be identified as the correct answer. It then inserts the 3 wrong answers but with the IsCorrect set to 0.

```
122
123 # 102 - To add a score
124 def AddScore(UserID:str, Score:str):
125     """Add a score.
126
127     Args:
128         UserID (str): The Users UserID
129         Score (str): The score in nanoseconds
130     """
131
132     Query = "INSERT INTO score (UserID, Score) VALUES (%s, %s);"
133     # Add the new score to the database
134     Cursor.execute(Query, (int(UserID), int(Score)))
135     # Execute the query on the database
136     Connection.commit()
137
```

When the user has completed the game, the program will send their score to the server which will then call this function. It takes in the UserID and their Score. It simply converts their UserID and Score into integers before inserting them into the Score table.

```
137
138 # 103 - Check login
139 def CheckLogin(Username:str, Password:str):
140     """Attempt to login
141
142     Args:
143         Username (str): Username of user
144         Password (str): Password attempted
145
146     Returns:
147         int: Successful login and UserID of user
148         'False': Incorrect login details
149     """
150
151     Query = "SELECT PasswordHash FROM user WHERE Username = %s;"
152     # Get the password hash of the username
153     Cursor.execute(Query, (Username,))
154     # Attempt to receive the passwordhash
155     try:
156         PasswordHash = Cursor.fetchall()[0][0]
157
```

To check the login details when someone tries to login, the details it sends to the server is the username and the password. It starts by attempting to get the password hash for the username specified.

```
157
158     # If no user with that username exists...
159     except:
160         return "False"
161
162     # If received data is None or blank...
163     if PasswordHash == "None" or PasswordHash == "":
164         return "False"
165
166     # check the passwords match...
167     if CheckPassword(Password, PasswordHash):
168         # Get UserID of the user logged in
169         Query = "SELECT UserID FROM user WHERE Username = %s;"
170         Cursor.execute(Query, (Username,))
171         UserID = Cursor.fetchall()[0][0]
172         return UserID
173     return "False"
174
```

However, if this fails it means that username doesn't exist so returns False. If the password hash isn't None and isn't empty it then uses the Check Password function to check if the password matches the hashed password.

If the passwords match, then it means they have logged in. Next it queries the database for the User ID of the specified username and then returns the User ID to signify they've logged in as well as for the program to identify who is has logged in as.

```
174
175 # 104 - To get random questions
176 def GetRandomQuestions(NumberOfRandomQuestions:int):
177     """Get x number of random questions.
178
179     Args:
180         NumberOfRandomQuestions (int): Number of questions to get
181
182     Returns:
183         str: String containing the questions
184     """
185
186     # Get number of total questions in database
187     Query = "SELECT COUNT(QuestionID) FROM question"
188     Cursor.execute(Query)
189     TotalNumberOfQuestions = int(Cursor.fetchall()[0][0])
190
191     # If there isn't enough questions in the database...
192     if TotalNumberOfQuestions < NumberOfRandomQuestions:
193         # Get every question from the database
194         NumberOfRandomQuestions = TotalNumberOfQuestions
195
196     # Generate x number of random QuestionIDs
197     RandomNumbers = []
198     while len(RandomNumbers) != NumberOfRandomQuestions:
199         RandomNumber = random.randint(1,TotalNumberOfQuestions)
200
201         # If the random QuestionID isn't already being used...
202         if not (RandomNumber in RandomNumbers):
203             RandomNumbers.append(RandomNumber)
204
```

This query is used to get random questions from the database when the user starts to play the game. It takes in an integer which represents the number of questions to get from the database. First it queries the database to get the total number of questions stored in the database. It does so by using SQL's count function. The number of total questions is required so that it can generate random numbers to select random questions. In case the client has requested more questions than exist, it will get all the questions from the database.

It then goes on to generate random numbers which will be used to select the questions. Using a loop which runs until there are enough random numbers, it generates a random number, checks if it is already in the list and if it isn't then adds it to the list.

```
205
206     Questions = []
207     QueryQuestion = "SELECT QuestionText FROM question WHERE QuestionID = %s;"
208     QueryAnswer = "SELECT AnswerText FROM answer WHERE QuestionID = %s AND IsCorrect = %s;"
209
210     # For each random QuestionID...
211     for RandomQuestionID in RandomNumbers:
212         TempQuestion = ["QuestionText", "CorrectIndex", None, None, None, None]
213
214         # Get the question text
215         Cursor.execute(QueryQuestion, (RandomQuestionID,))
216         Question = Cursor.fetchall()[0][0]
217         TempQuestion[0] = Question
218
219         # Get the correct answer
220         Cursor.execute(QueryAnswer, (RandomQuestionID, 1))
221         CorrectAnswer = Cursor.fetchall()[0][0]
222
```

Now it has all the random numbers, it loops through each random number. The loop starts by creating a Temporary Question which stores placeholders for the questions data. It then queries the database to get the text of the question using the Question ID, next it inserts this text into the TempQuestion list.

Secondly, it gets the correct answer by querying the database for the QuestionID and IsCorrect = 1.

```
222
223         # Randomise the position of the correct answer
224         PositionOfCorrectAnswer = random.randint(0,3)
225         TempQuestion[1] = PositionOfCorrectAnswer
226         TempQuestion[2+PositionOfCorrectAnswer] = CorrectAnswer
227
228         # Get the wrong answers
229         Cursor.execute(QueryAnswer, (RandomQuestionID, 0))
230         WrongAnswers = Cursor.fetchall()
231         # Shuffle the wrong answers
232         random.shuffle(WrongAnswers)
233
234         # Put wrong answers into the Temp Question List
235         for Index in range(4):
236             # Fill in the gaps in the list with the wrong answers
237             if TempQuestion[2+Index] == None:
238                 TempQuestion[2+Index] = WrongAnswers[0][0]
239                 del WrongAnswers[0]
240         # Add the Question to the list of questions
241         Questions.append(TempQuestion)
242
243     # Convert the lsit of questions into a single string
244     Questions = ConvertToString(Questions)
245     return Questions
246
```

Now it has the correct answer it generates a random number which is used to randomise the position of the answer in the list. It then inserts the correct answer into the temp question list.

Next it moves onto getting the wrong answers from the database. It then uses Python's shuffle function to put the wrong answers into a random order. Next it loops through each possible answer slot in the temp question list. For each slot it checks if it is empty and if it is then it puts the next wrong answer into the slot. If the slot isn't empty it means the correct answer is there and so it skips it and moves onto the next slot. At the end it then adds the Temp Question list to the list of Questions.

Before returning the questions, it uses the convert to string function to turn it into a string that can be sent over the Socket connection.

```
246
247 # 105 - Get Scores
248 def GetScores(StartPosition:int, Count:int):
249     """Get Count number of scores starting from StartPosition.
250
251     Args:
252         StartPosition (int): Position of first score to get (starts from 1)
253         Count (int): The number of scores to get
254
255     Returns:
256         str: Containing the specified scores
257     """
258
259     Query = "SELECT Username, Score, FirstName, LastName FROM user, score WHERE score.UserID =
    user.UserID ORDER BY Score ASC LIMIT %s OFFSET %s;"
260     # Get the scores from the database
261     Cursor.execute(Query, (Count, StartPosition-1))
262     Scores = Cursor.fetchall()
263     # convert the scores into a single string
264     Scores = ConvertToString(Scores)
265     return Scores
266
```

When the user goes to the leader board, changes to a new page, or clicks the refresh button, it will send a request to the server to get the new scores to be displayed. It takes in the Start Position and the number of scores to get as parameters. It simply queries the database using the parameter values but first subtracts 1 from the start position as the indexing within the database starts at 0. It then gets the results, converts it to a string and then returns the string containing the scores.

```
266
267 # 106 - Check username
268 def CheckUsername(Username):
269     """Check a username doesn't already exist in the database
270
271     Args:
272         Username (str): The username to check
273
274     Returns:
275         True: If username doesn't already exist
276         False: If username DOES already exist
277     """
278
279     Query = "SELECT UserID FROM user WHERE Username = %s;"
280     Cursor.execute(Query, (Username,))
281     Response = Cursor.fetchall()
282     if Response == []:
283         return True
284
285     return False
286
```

When a user is attempting to sign up at the start of the program and they've entered a username the program checks to see if that username already exists because if it does the user will be prompted to choose a different username. It queries the database for the User ID which has that username and if the response is an empty list this means no users have that username. If it makes it past checking the response is empty, then it means the username is taken so returns False.

The Constants File

Throughout the program there are many variables that won't change while the program is running but might need to be altered to make the game look better or more playable. By storing these constants in a single file, it means they only need to be changed once for it to affect many parts of the program.

The file is a JSON file which stores everything in a dictionary. Each item has a key which is a string used to identify the constant, as well as the value which is accessed whenever the key is referenced in the program. I designed the keys to follow a pattern so that the constants are easy to understand what they do. Each key starts with a code which represents which part of the program it relates to.

```
1
2 {
3     "text-colour": [255,255,255],
4     "textbox-outline-colour": [170,170,255],
5     "button-hover-outline-colour": [255,255,255],
6     "textbox-outline-width": 6,
7     "button-outline-width": 8,
8     "textbox-radius": 20,
9
10    "font-max-width": 2000,
11    "fade-amount": 8,
12
```

The first section of constants is relating to buttons and textboxes. The constants store the colours of the textboxes and buttons as well as the width of the outlines and how round the corners are.

The second section stores the maximum width of the font, this is needed when rendering the text as well as the fade amount which is used when fading between sections.

```
12
13      "fps": 30,
14      "mm-bg-colour": [10,10,10],
15      "mm-stars-ratio": 0.4,
16      "mm-stars-radius": 1,
17      "mm-stars-vel-range": [-4,-1],
18      "mm-stars-colour": [90,90,90],
19      "mm-cursor-size": 50,
20      "mm-btn-width": 350,
21      "mm-btn-height": 70,
22      "mm-font-size": 50,
23      "mm-mute-btn-width": 100,
24      "mm-mute-btn-height": 40,
25      "mm-mute-font-size": 30,
26
```

The next section of constants relate to the main menu as well as the background. It stores things such as the colour of the background, size of the stars, and size of the cursor.

```
26
27      "gm-rocket-width": 38,
28      "gm-rocket-height": 169,
29      "gm-rocket-angle-acceleration": -0.007,
30      "gm-rocket-v": 5,
31      "gm-rocket-max-angle-velocity": 0.06,
32      "gm-rocket-acceleration": 0.8,
33      "gm-friction-constant": 0.98,
34      "wrong-answer-time-penalty": 3,
35
36      "gm-planet-radius-range": [100,200],
37      "number-of-questions": 16,
38
```

The next section stores constants relating to the game. The size of the rocket, accelerations, velocities, and size of planets are stored here.

```
38
39      "pth-mm-theme": "assets\\sfx\\theme.wav",
40      "pth-mm-cursor": "assets\\vfx\\cursor.png",
41      "pth-font-regular": "assets\\txt\\font-regular.otf",
42      "pth-htp-music": "assets\\sfx\\theme.wav",
43      "pth-img-planet": "assets\\vfx\\planet.png",
44      "pth-img-rocket-off": "assets\\vfx\\rocket-off.png",
45      "pth-img-rocket-on": "assets\\vfx\\rocket-on.png"
46  }
47
```

The last part of the constants file stores the file paths to the different assets in the program. This is so that the assets can be changed without needing to edit the actual code.

The Final Program

The Client-Side Program

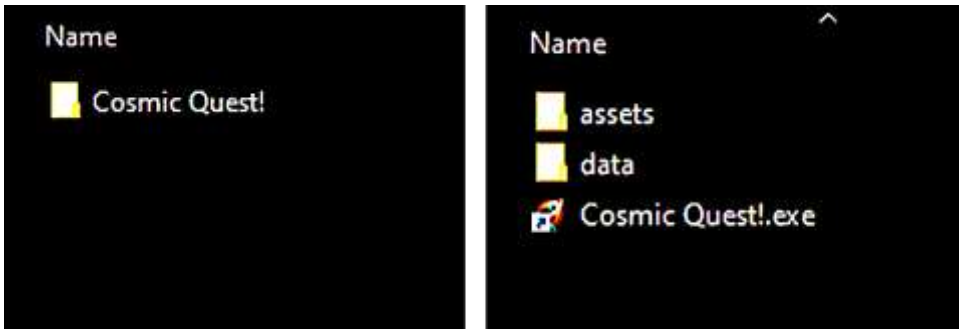
Converting the code to an EXE

Now that the program is fully programmed, I need to make it into something that can be easily shared and installed on many computers. To do this, I will use 2Python modules called auto-py-to-exe and pyinstaller which both work together to turn Python code into a single .EXE file. This is very important for my project as it will close-source the code meaning it cannot be modified but it was also allow it to be easily used. When it is an EXE file, the user will not need to install anything else except for having the .EXE file and the assets folders to store the images, sounds, and constants file.

To turn the program into an EXE file, I used auto-py-to-exe to generate the following command which then uses pyinstaller to bundle the program.

```
pyinstaller --noconfirm --onefile --console
    --add-data "./data/app/consts.json"
    --add-data "./assets/sfx/theme.wav"
    --add-data "./assets/txt/font-regular.otf"
    --add-data "./assets/vfx/cursor.png"
    --add-data "./assets/vfx/planet.png"
    --add-data "./assets/vfx/rocket-off.png"
    --add-data "./assets/vfx/rocket-on.png"
    "./program.py"
```

By running the above code, it outputs an EXE file. By combining this with the assets and other required folders, the program that the user will see is shown below:



To use the program all they will need to do is open the EXE file.

Customising the game

Although the program will be fully functional, users will be able to customise the game to change how it looks but not how it functions. This is important as if they can change the functionality, they may be able to get a very quick score which wouldn't normally be possible.

They can customise the game by editing the consts.json file which is stored within the data folder. Within this file, they can change the colour of buttons and textboxes, as well as change the density of the stars or the size of them.

```
{
    "text-colour": [255,255,255],
    "textbox-outline-colour": [170,170,255],
    "button-hover-outline-colour": [255,255,255],
    "textbox-outline-width": 6,
    "button-outline-width": 8,
    "textbox-radius": 20,

    "fade-amount": 8,

    "fps": 30,

    "mm-bg-colour": [10,10,10],
    "mm-stars-ratio": 0.4,
    "mm-stars-radius": 1,
    "mm-stars-vel-range": [-4,-1],
    "mm-stars-colour": [90,90,90],
    "mm-cursor-size": 50,

    "pth-mm-theme": "assets\\sfx\\theme.wav",
    "pth-mm-cursor": "assets\\vfx\\cursor.png",
    "pth-font-regular": "assets\\txt\\font-regular.otf",
    "pth-htp-music": "assets\\sfx\\theme.wav",
    "pth-img-planet": "assets\\vfx\\planet.png",
    "pth-img-rocket-off": "assets\\vfx\\rocket-off.png",
    "pth-img-rocket-on": "assets\\vfx\\rocket-on.png"
}
```

The above screenshot shows all the constants which can be changed to change the look of the program.

The Server-Side Program

Usage

Unlike the main program, the server will not be converted to an EXE file as it will not be able to access the database if it was an EXE. Therefore, the server will be kept as a Python file. The way the server works, the database needs to be on the same device as that which is running the server. This is done so that it is more secure as the database can only be accessed via the server and no other way. The server would not need to be installed as it would be run by myself due to it not needing to be specific to the client.

Adding More Questions

If more questions need to be added to the database, this will be done directly on the device which has the database. This is so that the database cannot be accessed from anywhere else except the physical device, and through the server where only specific requests can be made.

How the program can be used in a lesson

As this program is designed for use in schools, it is essential that it will be able to be used in a classroom environment. Therefore, I have planned how the program can be used within a lesson.

Before it can be used, the program will need to be distributed to each device. This can be done in the same way any software is installed on all the devices.

Within the lesson, students would first create an account or login, then they would play the game several times. While the class is playing the game, the teacher would show the leader board on the projector in the class room. This is so that the students can look at the leader board and see who is doing well and who they need to try and beat.

This could be used for a whole lesson, or it could be part of a lesson where students get to choose how they revise. As mentioned in the analysis, the best way to revise is to use lots of different techniques not just one.

Websites I Used

Throughout the making of the program, there were several times where I did not know how to use a specific function or needed to know more about something to implement it correctly. The table below shows the websites I used and what for:

Link	Description
https://www.geeksforgeeks.org/python-display-images-with-pygame/	How to load an image into a Pygame surface.
https://pythonprogramming.net/adding-sounds-music-pygame/	How to play music using Pygame.
https://stackoverflow.com/questions/2612802/how-do-i-clone-a-list-so-that-it-doesnt-change-unexpectedly-after-assignment	How to make a duplicate of a list so that editing the copied list doesn't edit the original list.

Testing

Recorded Testing

So that you can see how the program works and that it is entirely functional, I have recorded various videos to help with the testing section. I recorded 3 videos, the most important one is the “Client & Server Testing Video” which shows the client-side program and the server running while I explain the different parts of the program that are being tested. The other 2 videos show the server and the program running but without a commentary.

To access the videos, I have made a website which has buttons to the 3 videos which have been uploaded to YouTube. Either scan the QR code below to access the website or type in the URL which is:

https://henryjwalker.github.io/

If the above website doesn’t work, below is the URL of the most important testing YouTube video:

https://youtu.be/o1uMGxvTHoE

Testing While Programming

While programming, I tested the section of code I was working on at that time. This meant I fixed the errors as they appeared and would mean there would be less errors to deal with once it was finished. Compared to the testing carried out once the program was finished, this testing is much more detailed as the problems were much more specific.

Questions Getting Skipped

The Problem

While coding the first version of the game, I had a class called Questions. This was similar to how the QuestionManager functions in the final program. While testing the game and checking that questions appear and the correct answer moves the player on, I noticed that occasionally a question wouldn’t appear, and it would ‘skip’ the question. After debugging the program thoroughly, I found the error in this code:

```
if mouse_clicked:
    index_selected = questions.check_click(mouse_pos)

if index_selected == questions.get_correct_index():
    questions.next_question()
```

```
def check_mouse_click(self, c_pos):
    for i, button in enumerate(self.buttons):
        if button.detect_hover(c_pos):
            return i
    else:
        return False
```

The problem is that the function I created, check_click(), can return False or the index of the button the mouse is hovering over. In the 2nd 'if' statement, I intended for it to check if index_selected (the integer) is the same as the index of the correct answer, and if it is, then move onto the next question. The problem arises when index_selected is False and not an integer. This means the 'if' statement is evaluating the following:

$$if\ False == CorrectAnswerIndex:$$

I believed that this would evaluate to False which in most cases it would, but the error arises when the correct answer index is 0. This gives the following condition:

$$False == 0$$

In Python, this evaluates to True. This meant that when the user doesn't click a button AND the correct answer index is 0, it will think the user clicked the correct answer.

When a button hasn't been clicked index_selected is set to False. Then when doing the if statement, if the correct index is 0, this would return True because in Python False == 0.

How I Fixed It

To stop this from happening, instead of the function returning False when nothing was clicked, I changed it to return None. As well as this, I then created the Question Manager which then checked the condition by matching the buttons ID with the correct answer index. This can be seen within the Check Click Method of the Question Manager.

Rocket not showing flames when moving forwards.

The Problem

When coding the rocket into the game, I encountered a problem with it switching images when moving and not moving. What is meant to happen is that when the user is pressing the 'w' key to move the rocket, it displays flames coming out from the bottom of the rocket and then when 'w' isn't being pressed, it doesn't show any flames. The way I coded this was by having 2 images, 1 with the flames and 1 without, and then the program did all the rendering twice so that the correct image could be displayed when needed.

However, after implementing this using the code below, the rocket was not changing image when it was meant to.

```
self.original_surface_off = pygame.image.load(self.consts["pth-img-rocket-off"])
self.original_surface_on = pygame.image.load(self.consts["pth-img-rocket-on"])

self.original_surface_off = pygame.transform.smoothscale(self.original_surface_off, self.size)
self.original_surface_on = pygame.transform.smoothscale(self.original_surface_on, self.size)

self.surface_off = self.original_surface_off
self.surface_on = self.original_surface_on
```

This is because of the bottom 2 lines of code in the screenshot above. What I intended for this code to do was to create a copy of both surfaces. The problem is that Pygame surfaces are objects, and within Python, when you do what I did in the bottom 2 lines, it doesn't make a copy of the object, instead it creates a new pointer to the SAME object. This meant that there was no copy and only 1 surface object. Therefore, when it came to rendering the different states, it wasn't using the original surface and was overriding the last state. This meant the state never changed as there was no on state.

How I Fixed It

After learning that it is because Pygame surfaces are objects, and objects can't be copied how I thought they could, I learnt that I needed to use the .copy() method.

This meant the code looked like this:

```
# Create the Current surfaces
self.__CurrentSurfaceOff = self.__OriginalSurfaceOff.copy()
self.__CurrentSurfaceOn = self.__OriginalSurfaceOn.copy()
```

Now with that changed it worked as intended and changed to display the correct image when the 'w' key is pressed.

Rotational Friction

The Problem

After playing the game several times, I realised it was very easy to get to the other planet. This was because you just had to line up the rocket with the other planet and go forwards. When the player stops turning the rocket, it stops rotating instantly. This isn't how a rocket would behave in space and it makes it too easy to play the game.

How I Fixed It

Therefore, to make it harder, I added decided to add rotational friction, this means that the rocket will continue rotating after the user stops pressing 'a' or 'd'.

By adding rotational momentum to the game, it will make it more challenging and engaging for the player because it introduces an additional element of control that they need to consider. With rotational momentum, the rocket will continue to rotate which can make it more difficult to navigate to the other planet. This will also add a sense of realism to the game, as it mimics the way that real rockets and other objects behave in space.

To add this to the game, I added the below attributes to the rocket:

```
# Create rockets angle variables
self.__Angle = 0
self.__AngleVelocity = 0
self.__AngleAcceleration = CONSTS["gm-rocket-angle-acceleration"]
```

Then when the user presses 'a' or 'd', it adds the acceleration to the velocity. Then each frame it adds the velocity to the angle so that it continues spinning. To make it not too difficult to play, it does slow down, and this is implemented by multiplying the velocity by 0.95 each frame.

```
# Increase angle by the current angle velocity
self.__Angle += self.__AngleVelocity
# Scale angle velocity by frictional constant
self.__AngleVelocity *= CONSTS["gm-friction-constant"]
```

After implementing this, I asked several people, which had played it before, to play it again. They all agreed that this made it harder to control the rocket but that it was much more fun as it made it more of a challenge.

Connection to server unexpectedly closing.

The Problem

When testing the game and leader board after finishing the program and just after adding the new questions to the database, I discovered that the connection would close after receiving the questions from the server. What got my attention was when playing the game, the last question was missing 2 answers and then when I went to the leader board, nothing appeared due the connection being closed.

As you can see from the screenshot, the leader board still functions apart from displaying the results.

I found that the problem was that when the server sends the questions to the client, it takes up too many characters which it then can't send over the connection. The socket connection has a limit on the amount of data it can send in one go and all the questions together goes over this limit. When it goes over this limit the connection crashes which causes it to disconnect from the server.

How I Fixed It

To fix this I changed the number of questions sent in one go from the server to the client. Originally, it was 50 questions but this was too many. I changed it to 30 which is below the limit but is still enough for the game to work properly.

Complex Button Class

The Problem

The original Button class I created was very customisable which allowed every part of it to be changed. This was very useful however it meant that whenever I wanted to create a button, it would require a lot of variables and parameters. This took up a lot of space on the screen and was very hard to understand. Below shows all the code just to create a single button. This would be on 1 line, but is so long it wraps onto 4 lines:

```
self.buttons.append(Button(self.consts, self.res, "play", self.res[0]/2,
(self.res[1]/2) - 2*self.consts["mm-btn-height"] - 40, "play",
self.consts["mm-font-size"], self.consts["mm-btn-width"],
self.consts["mm-btn-height"]))
```

How I Fixed It

Instead of compromising on customisability, I decided to abstract the program further as well as provide better encapsulation for the buttons. I did this by creating the Layout Manager class. The Layout Manager class creates the button from within the class and to create a button, only takes in the required parameters. With this implemented it meant to create a button, this is all the code that is needed:

```
self.__Layout.AddButton("play", (3.1,3.1), (0.8,0.8), "Play")
```

This made it much easier to understand the code just from looking at it and it also made it a lot quicker to code.

Specific testing for each objective

#	Objective	How was it tested?	Was it met?
1	The code will be closed source.	This doesn't need testing as once it is converted to an exe file, nothing can be accessed, thus making it closed source.	Yes
2	The program runs on the Windows operating system.	By running the program on windows 10 and 11. You can see in the testing video of this working.	Yes
3	The program properly runs on the most common sizes of screens.	By running the program on various different screen sizes. You can see it working in the testing video.	Yes
4	The questions will cover all the GCSE Physics content.	When I made the questions I made sure they cover every topic.	Yes
5	All questions should be multiple choice and require no calculations to get the answer.	Again, when I made the questions, I made sure that they all have multiple choice answers.	Yes
6	Nothing will need to be installed (even Python) to run the program (they'll only need the files).	In the testing video you can see all the files that are required, once you have the files, the testing video shows how you run the program.	Yes
7	Users need to create an account or login before accessing the program.	In the testing video you can see that without logging in, you can't get to the main program.	Yes
8	User accounts will be stored on a remote database.	In the testing video you can see that when an account is created it is sent to the server where it is then stored in the database.	Yes
9	All stored passwords are hashed before being stored.	In the code for the query manager, you can see that only hashed passwords are put into the database.	Yes
10	The program will have a main menu which will allow the user to choose what they want to do.	This can be seen in the testing video, and you can see the buttons working correctly as well.	Yes
11	Music will only play while the user is on the main menu, and not anywhere else in the program.	This is shown in the testing video where the music stops and starts when leaving or entering the main menu.	Yes
12	Before playing the game, clear instructions will explain how to play the game and what the aim is. This can be skipped if the user desires.	The instructions can be seen in the testing video where they give an overview of the game, then the controls, and finally the aims of the game.	Yes
13	The timer will only start once the user starts moving the rocket.	This is shown off in the testing video.	Yes
14	The questions and answers will be stored in a remote database.	In the testing video you can see the server send the questions to the client after it has retrieved them from the database.	Yes

15	Only 2 planets will be shown on the screen while playing the game.	This can be seen in the testing video.	Yes
16	The user will have to correctly answer 8 questions for the timer to stop and the game to end.	As you can see from the testing video, the timer only stops once 8 questions have been answered correctly.	Yes
17	If a question is answered incorrectly, the user is given another question and they get a time penalty.	This is shown off in the testing video.	Yes
18	When the user completes the game, their score will be stored within a remote database.	In the testing video you can see the server receive the request to store the users score in the database.	Yes
19	The program will have a leader board where users can see everyone's scores. The scores will be accessed from a remote database.	This can be seen in the testing video.	Yes
20	A server will be used to receive requests, process them, and form a response for the client.	This can be seen in the testing video as it receives the requests, processes them, and then sends the response back to the client.	Yes
21	The way the user interacts with the program should be simple, minimal, and quick and the rocket should be controlled using W, A, S, and D.	In the testing video you can see that everything is intuitive, and everything is explained beforehand.	Yes
22	At least 50% of GCSE Physics students say they would use the game to revise.		

Thorough Testing Once Program Completed

Once I had finished programming the program, I thoroughly tested it to make sure it did exactly as it was intended to do. I tested every part of the program to make sure there is nothing that will cause the user to become confused or the program to crash.

#	Description	Test Data	Expected Result	Pass/Fail	References
1	Account can be created and gets added to the database.	1, henryw, 123456, Henry, Walker	After values are entered, it goes to the main menu.	Pass	SS1, SS3
2	Account can be logged into after creating.	2, henryw1, 123456	After entering values, it goes to the main menu.	Pass	SS2, SS3
3	Entering no data when creating an account or logging in.	No input to username	Error message shown.	Pass	SS13, SS14
4	Entering extreme data when creating an account or logging in.	20,000 'a's	Server will detect extreme data and close connection.	Pass	SS15
5	Entering separator characters when creating an account	Henry##	Error message shown and returned to menu.	Pass	SS16
6	Buttons on the main menu change colour when mouse is hovering over it	{Move mouse over a button}	Button changes the outline colour when hovered	Pass	SS4

7	Button works and moves to the correct section.	{Play button clicked}	Moves to the correct section when button clicked.	Pass	SS5
8	The music plays on the main menu and stops when switching to a different section.	2, henryw1, 123456, {clicks Play / Leader board}	Music plays when main menu appears, stops when a button is pressed to play the game.	Pass	N/A
9	The music stops and restarts when the mute button is clicked twice.	{Click mute button}, wait a few seconds, {click mute button}	Music stops on first click, then starts again on the second click	Pass	N/A
10	Instructions are displayed when user goes to play the game.	{click play button}	After clicking play button it displays instructions on how to play the game.	Pass	SS5
11	On the last instruction page, the next button changes to a start button.	{Click next button twice}	The next button changes to Start	Pass	SS6
12	When start button clicked planets and rocket are displayed.	{Start button clicked}	Planets and rocket are displayed.	Pass	SS7
13	Timer does NOT start straight away and only starts when rocket starts moving.	{Use W, A, S, and D to control the rocket and move it}	Timer starts as soon as a key is pressed.	Pass	SS7, SS8
14	Correct questions counter starts at zero and is displayed	{Start the game}	Counter displayed at top of screen and says 0/8	Pass	SS7
15	Rocket collision with planet is detected and displays a question.	{Move rocket to the second planet}	Question appears when rocket collides with 2nd planet.	Pass	SS9
16	Question disappears and displays rocket and new planets when correct answer clicked.	{correct answer clicked}	Question disappears, rocket and planets appear.	Pass	SS9, SS10
17	Correct questions counter increases when correct answer clicked.	{Click correct answer}	Counter increases to 1/8	Pass	SS10
18	Once 8 questions have been answered correctly the timer stops and displays buttons.	{Answer 8 questions correctly}	Timer stops, displays time, and buttons displayed.	Pass	SS11
19	When Leader Board button clicked from main menu and from end of game it displays the leader board.	{Click leader board button}	Leader Board displays	Pass	SS12
20	Leader board scores are displayed in correct order and shows the correct info.	{Click leader board button}	Scores are displayed in order of quickest to slowest. Each score has the time, username, and full name.	Pass	SS12
21	The Down button moves to the next page of results.	{Down button clicked}	Next page is displayed, and correct results are displayed.	Pass	N/A
22	The Up button doesn't do anything when on the top page and Down button	{Down and up buttons clicked on first and last pages}	The page stays the same and doesn't move.	Pass	N/A

	doesn't move down on the last page.				
23	Server can add users to the database.	AddUser() from query manager used.	User inserted into the database.	Pass	SS17
24	Server can add questions to the database.	AddQuestion() from query manager used.	Question and the 4 answers are added to the database.	Pass	SS18, SS19
25	Server can add score to the database.	AddScore() from query manager used.	UserID and score are added to the database.	Pass	SS20
26	Server can query database for questions and answers.	GetRandomQuestions() from query manager used.	Correct number of questions, and in random order are returned.	Pass	SS9
27	Server can query database for scores.	GetScores() from query manager used.	Correct number of scores, and in descending order returned.	Pass	SS12
28	The program can run on: • Windows 11 • Windows 10	Run program, login, play game.	The program should run correctly and be playable.	Pass	SS21, SS22
29	The program runs on the most common screen sizes.	N/A	The program runs and everything is in the correct positions. SS23: 16:9, SS24: 4:3, SS25: 2:1.	Pass – The program works correctly down to a resolution of 700x700	SS23, SS24, SS25

Typical, Erroneous, and Extreme Tests

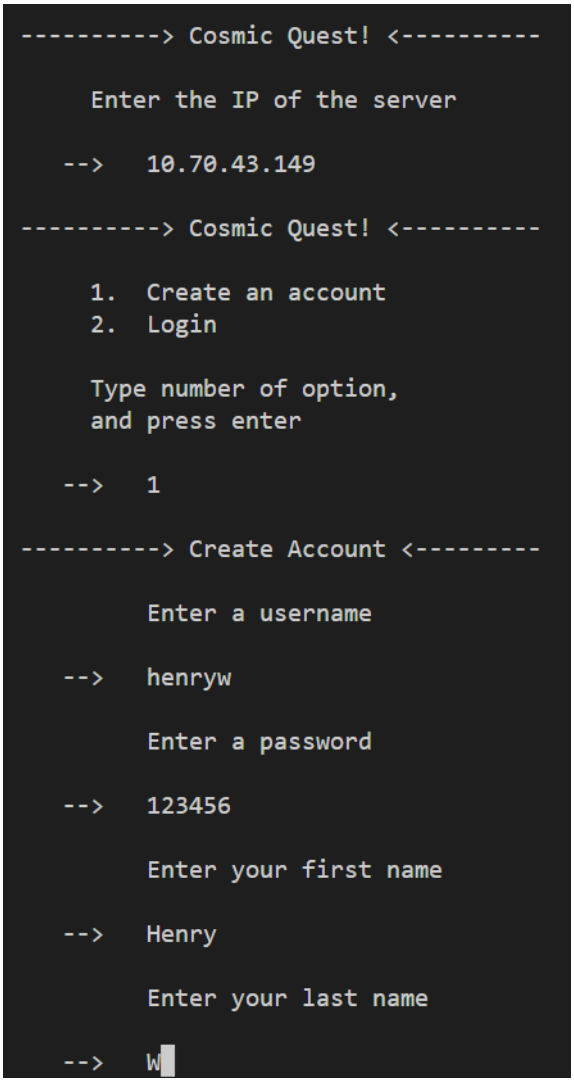
To make sure my program is robust, it is standard to test inputs by inputting typical data (data that is expected), erroneous data (types of data that is intended to cause errors), and extreme data (data that tests the boundaries).

When I designed my program, I did so in a way that meant the user would require very little input into the program to use it. This means the only place that they can input data into the program is from the create account and login sections.

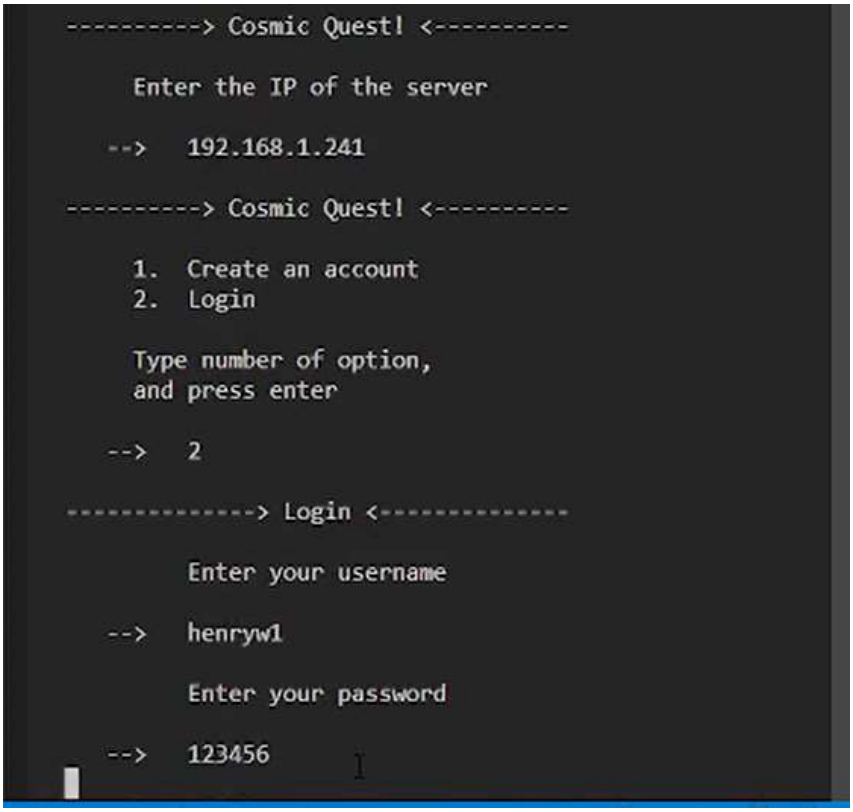
I tested these inputs thoroughly by using all 3 types of tests on every input into the program. These tests have been summarized in the above table in tests 3, 4, and 5.

References

SS1



SS2


----------> Cosmic Quest! <----------
Enter the IP of the server
--> 192.168.1.241
----------> Cosmic Quest! <----------
1. Create an account
2. Login
Type number of option,
and press enter
--> 2
--------------> Login <--------------
Enter your username
--> henryw1
Enter your password
--> 123456

SS3


Cosmic Quest!
Play
Leader Board
Exit
Mute

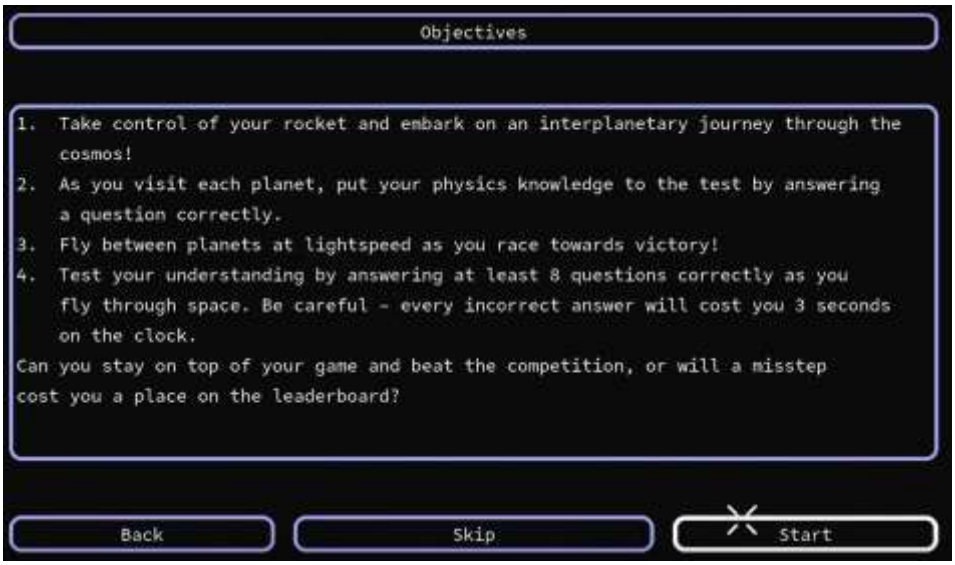
SS4


Play
Leader Board
Exit
Mute

SS5


Welcome to Cosmic Quest!
Welcome to Cosmic Quest! In this exciting 2D game, you'll take control of a rocket as you navigate through the vast expanse of space, visiting different planets and answering challenging physics questions along the way. Cosmic Quest is the perfect game for students looking to test their knowledge and have a blast while doing it. As you travel from planet to planet, you'll be faced with a variety of physics concepts. Each correct answer will bring you closer to your ultimate goal: completing the Cosmic Quest and earning a high score on the leaderboard. But beware - the clock is ticking! You'll need to use all of your knowledge and quick reflexes to beat the clock and emerge as the ultimate physics champion.
Back
Skip
Next

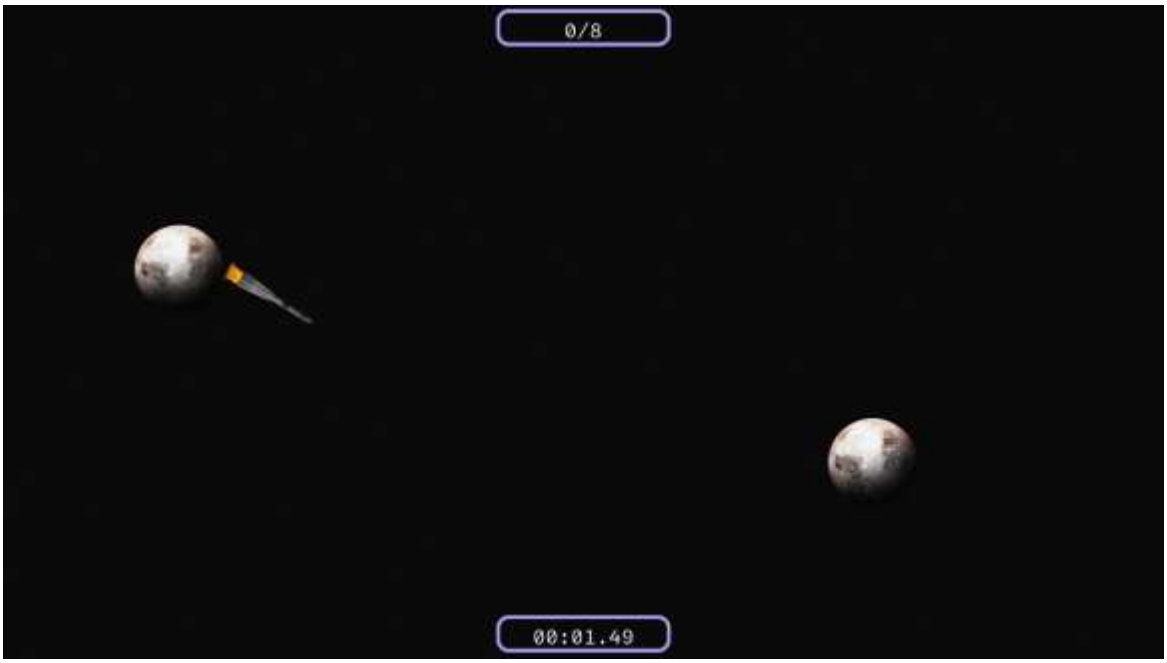
SS6


Objectives
1. Take control of your rocket and embark on an interplanetary journey through the cosmos!
2. As you visit each planet, put your physics knowledge to the test by answering a question correctly.
3. Fly between planets at lightspeed as you race towards victory!
4. Test your understanding by answering at least 8 questions correctly as you fly through space. Be careful - every incorrect answer will cost you 3 seconds on the clock.
Can you stay on top of your game and beat the competition, or will a misstep cost you a place on the leaderboard?
Back
Skip
Start

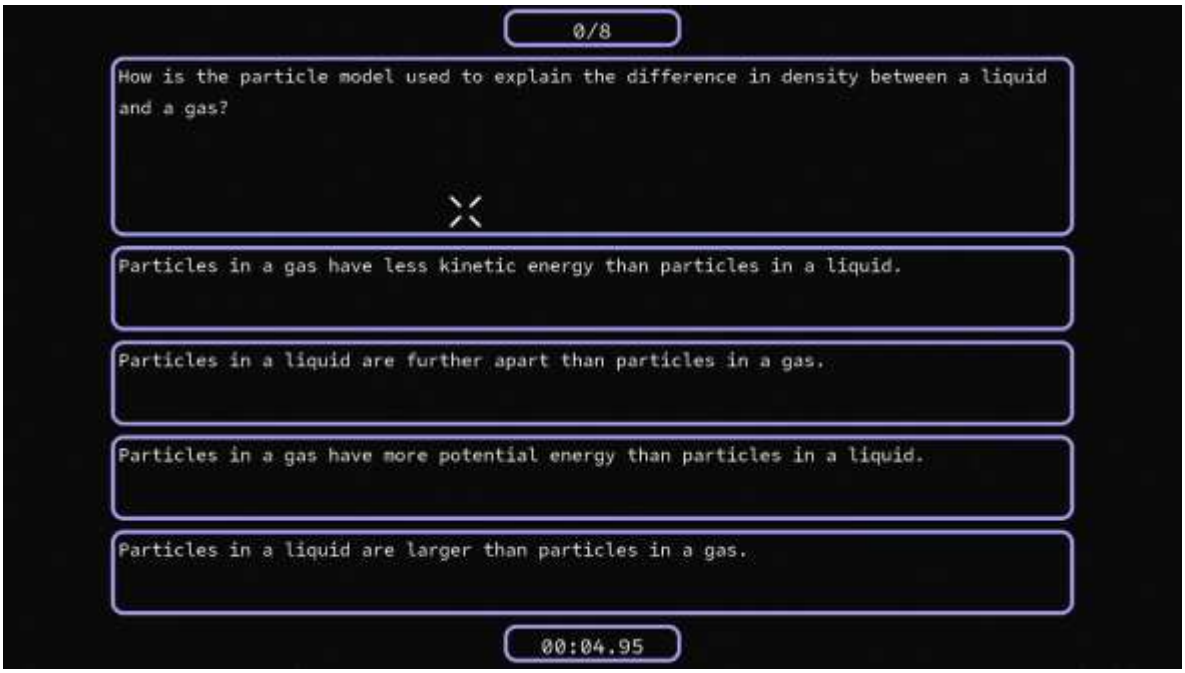
SS7


0/8
00:00.00

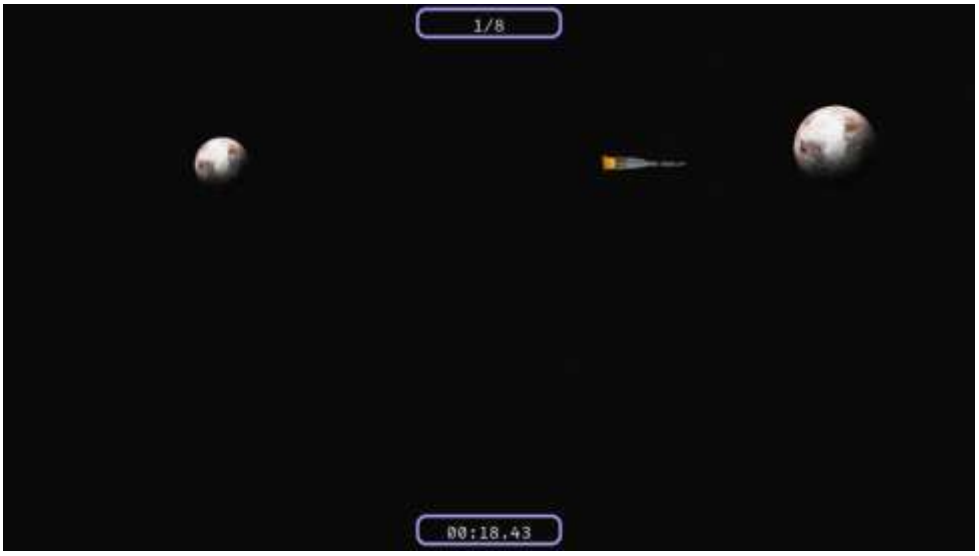
SS8


0/8
00:01.49

SS9


0/8
How is the particle model used to explain the difference in density between a liquid and a gas?
Particles in a gas have less kinetic energy than particles in a liquid.
Particles in a liquid are further apart than particles in a gas.
Particles in a gas have more potential energy than particles in a liquid.
Particles in a liquid are larger than particles in a gas.
00:04.95

SS10

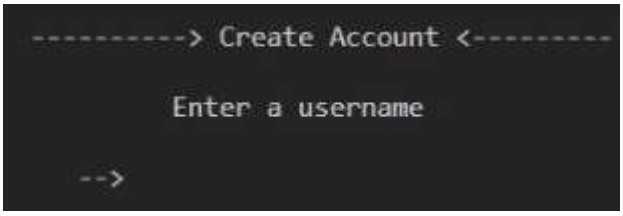

1/8
00:18.43

SS11


Your time is: 01:11.78
Menu
Leaderboard
Play Again
Exit

SS12

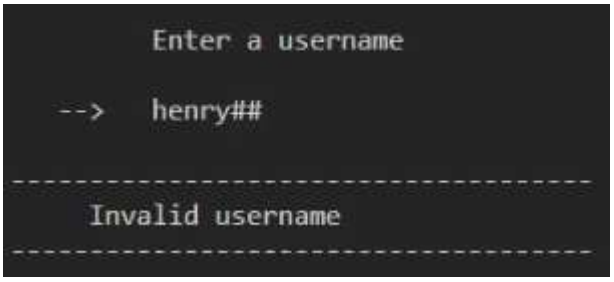
SS13



SS14

SS15

SS16



SS17

UserID	Username	PasswordHash	FirstName	LastName
1	hjw	$2b$12$g4ihg4EloXsIgPRcm...	Henry	Walker
2	henry	$2b$12$AeMI5hhL2DTdSBw...	Henry	Walker
3	henry 2	$2b$12$Ftou9xJbGFFJ3j/a...	henry	W

SS18

QuestionID	QuestionText
1	Light bulbs are labelled with a power input.$$What does power input mean?
2	How is the particle model used to explain the difference in density between a liquid$$and a gas?
3	What name is given to the total kinetic energy and potential energy of all the$$particles of helium ...
4	What term is used to describe an object or group of objects in physics?
5	What term is used to mean everything outside of a system?
6	What change happens to the energy store of the system when a ball is projected$$upwards?
7	What energy change happens to the system when a vehicle brakes and slows down?
8	What term is used to describe energy that has spread out into a less useful form?

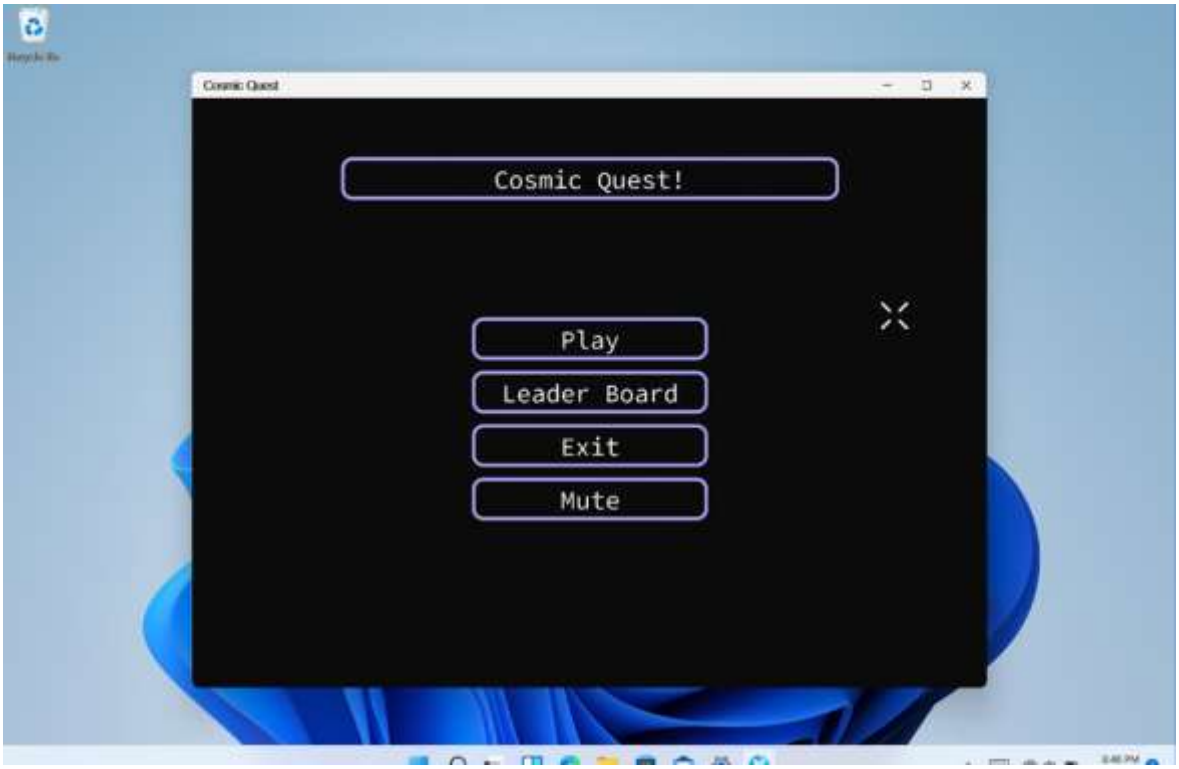
SS19

AnswerID	QuestionID	AnswerText	IsCorrect
1	1	The energy transferred each second to the bulb	1
2	1	The charge transferrred each second by the bulb	0
3	1	The current through the bulb	0
4	1	The potential difference across the bulb	0
5	2	Particles in a gas have more potential energy than...	1
6	2	Particles in a gas have less kinetic energy than par...	0

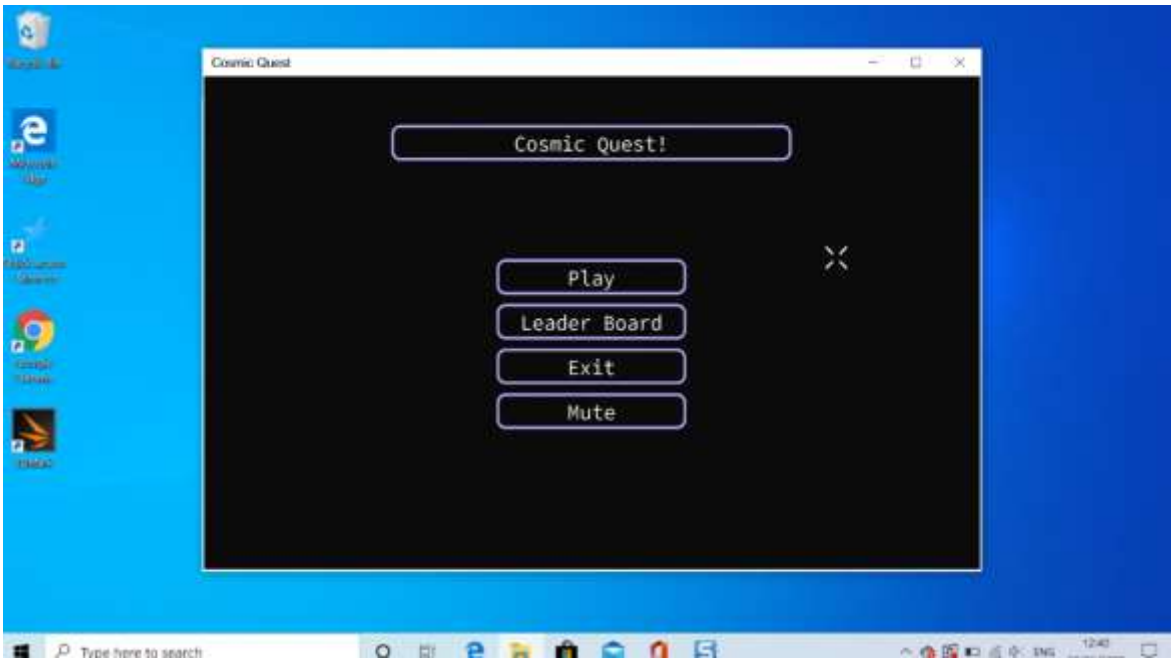
SS20

ScoreID	UserID	Score
1	1	57632195100
2	1	56169998000
3	1	63128243800
4	2	101702823900
5	2	79811222100

SS21


Cosmic Quest
Cosmic Quest!
Play
Leader Board
Exit
Mute

SS22


Cosmic Quest
Cosmic Quest!
Play
Leader Board
Exit
Mute

SS23


Cosmic Quest!
Play
Leader Board
Exit
Mute

SS24


Cosmic Quest!
Play
Leader Board
Exit
Mute

SS25

Evaluation

Now that the program has been coded and tested, it's time to assess whether it has successfully met the original objectives. This step is an important part of the project, as it helps to identify areas that require improvement for future projects.

To evaluate the project, I will begin by reviewing the original objectives to ensure that they have been met. Once I have done this, I will get feedback from both the client and the students who would be using the program. Their feedback is important as it provides an insight into the program's usability, functionality, and overall effectiveness.

After collecting feedback, I will analyse it thoroughly to give an overall evaluation of the project. This analysis will involve looking at both positive and negative feedback, identifying common themes, and determining areas for improvement. By doing so, I can use the information gained to make improvements for future projects, ultimately helping to ensure their success.

Objectives Met

The first part of the evaluation is to check if the project has met the objectives that were set in the beginning. To do this, I will use the tests that were performed during the testing phase. These objectives are important because they are based on the requirements of the client. It's important to ensure that the project meets these objectives as they determine the success of the project. By meeting the objectives, we can ensure that the client's needs are met, and the project is completed as the client desired. Testing plays a crucial role in evaluating the project as it helps to identify any areas where the objectives have not been met, allowing for improvements to be made. Therefore, the first step in the evaluation process involves reviewing the original objectives and testing the project to ensure that these objectives have been met. By doing so, I can ensure that the project meets the client's requirements, and any improvements can be commented on.

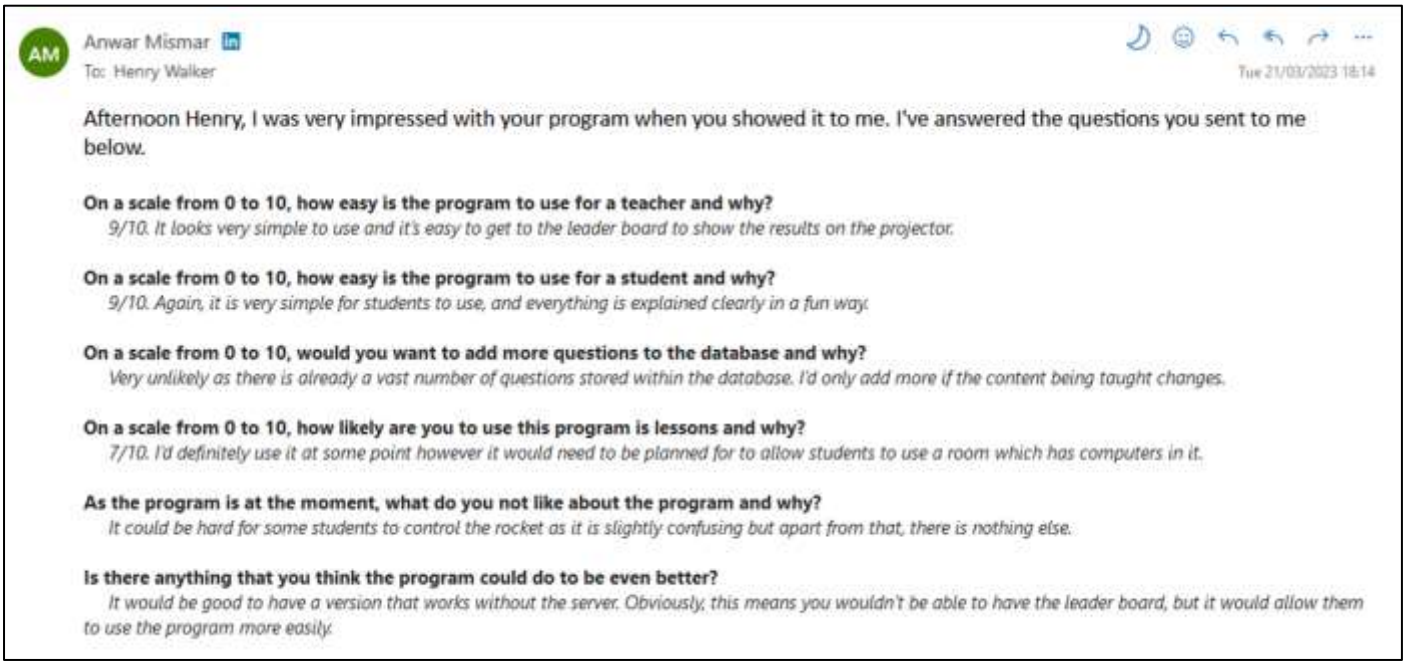
#	Objective	Met?	How well was it met?
1	The code will be closed source.	Yes	9/10. The code is turned into an executable file which cannot be edited. The only part of the program that can be edited is the constants file which allows the user to customise the game.
2	The program runs on the Windows operating system.	Yes	During the testing, I tested that it runs on many different devices. It ran on Windows 11, Windows 10, and various devices with various specs.
3	The program properly runs on the most common sizes of screens.	Yes	7/10. During testing, I tested that the program runs on many different screen sizes which is does. It runs on screen sizes down to a width and height of 700x700.
4	The questions will cover all the GCSE Physics content.	Yes	8/10. The questions I made cover every topic from GCSE Physics and more can be added by the teacher if they desire.
5	All questions should be multiple choice and require no calculations to get the answer.	Yes	Every question can be worked out without the use of a calculator and without needing to write anything down.
6	Nothing will need to be installed (even Python) to run the program (they'll only need the files).	Yes	Everything that is needed is stored within a single folder and nothing needs to be installed.
7	Users need to create an account or login before accessing the program.	Yes	Before the main menu is displayed the user has to create an account or login. This is checked during the testing stage.
8	User accounts will be stored on a remote database.	Yes	When an account is created the account details get sent to the server where they are then stored within the database.

9	All stored passwords are hashed before being stored.	Yes	Before the passwords are stored within the database they are hashed. No passwords are stored un-hashed.
10	The program will have a main menu which will allow the user to choose what they want to do.	Yes	Once the user has logged in they are shown the main menu where they then have the options to: Play the game, see the leader board, mute the music, or exit the game.
11	Music will only play while the user is on the main menu, and not anywhere else in the program.	Yes	As shown during testing the music stops as soon as the user moves off of the main menu and plays when they return to the main menu.
12	Before playing the game, clear instructions will explain how to play the game and what the aim is. This can be skipped if the user desires.	Yes	8/10. Using the feedback from the client and other users, the instructions are clear enough to be able to play the game while also being concise.
13	The timer will only start once the user starts moving the rocket.	Yes	This is shown working in the testing video where the timer doesn't start until the rocket starts to move.
14	The questions and answers will be stored in a remote database.	Yes	The questions are stored in the database and the program connects to the server using a socket connection where it then retrieves the questions and their answers.
15	Only 2 planets will be shown on the screen while playing the game.	Yes	As shown in the testing stage of the project, only 2 planets are shown at a time. When I got people to play the game, they agreed that it was clear where they needed to get to.
16	The user will have to correctly answer 8 questions for the timer to stop and the game to end.	Yes	This is shown in the testing video that you need to answer 8 questions correctly for the game to end.
17	If a question is answered incorrectly, the user is given another question and they get a time penalty.	Yes	This is shown in the testing video.
18	When the user completes the game, their score will be stored within a remote database.	Yes	Once the user completes the game their score and their user ID is sent to the server where it then stores them in the database.
19	The program will have a leader board where users can see everyone's scores. The scores will be accessed from a remote database.	Yes	This can be accessed either from the main menu or once the user has completed the game. This is shown working in the testing video.
20	A server will be used to receive requests, process them, and form a response for the client.	Yes	This is shown working in the testing video where you can see it receiving a request, processing it, then sending back a response.
21	The way the user interacts with the program should be simple, minimal, and quick and the rocket should be controlled using W, A, S, and D.	Yes	8/10. I got various people to test the program and they all agreed that the instructions clearly explained how to play the game and that the controls were easy to use.
22	At least 50% of GCSE Physics students say they would use the game to revise.	Yes	Below shows an online survey I did which I used to get feedback on the project from a large number of students. The results clearly show that the vast majority of students would use the program to revise.

Now that I have reviewed how well the project meets the original objectives, I can conclude that the project fulfils everything it was meant to and more.

Feedback from client

Since the program is designed for use in schools, I presented it to my client, Mr Mismar, a Physics teacher who teaches GCSE and A level physics, to get his feedback. I explained how the program could be used during lessons (explained at the end of the technical log) and then later email him and asked him some questions to find out how well it meets his needs.



Anwar Mismar

To: Henry Walker

Tue 21/03/2023 18:14

Afternoon Henry, I was very impressed with your program when you showed it to me. I've answered the questions you sent to me below.

On a scale from 0 to 10, how easy is the program to use for a teacher and why?
9/10. It looks very simple to use and it's easy to get to the leader board to show the results on the projector.

On a scale from 0 to 10, how easy is the program to use for a student and why?
9/10. Again, it is very simple for students to use, and everything is explained clearly in a fun way.

On a scale from 0 to 10, would you want to add more questions to the database and why?
Very unlikely as there is already a vast number of questions stored within the database. I'd only add more if the content being taught changes.

On a scale from 0 to 10, how likely are you to use this program is lessons and why?
7/10. I'd definitely use it at some point however it would need to be planned for to allow students to use a room which has computers in it.

As the program is at the moment, what do you not like about the program and why?
It could be hard for some students to control the rocket as it is slightly confusing but apart from that, there is nothing else.

Is there anything that you think the program could do to be even better?
It would be good to have a version that works without the server. Obviously, this means you wouldn't be able to have the leader board, but it would allow them to use the program more easily.

Mr Mismar's feedback is important because he has experience in teaching Physics, which means he can provide valuable feedback into how the program could be used as well as how it can be improved. By asking him specific questions, I can gain a better understanding of the program's strengths and weaknesses, and what could be improved to make it more effective in a classroom setting.

Question	Response
On a scale from 0 to 10, how easy is the program to use for a teacher and why?	9/10. It looks very simple to use and it's easy to get to the leader board to show the results on the projector.
On a scale from 0 to 10, how easy is the program to use for a student and why?	9/10. Again, it is very simple for students to use, and everything is explained clearly in a fun way.
On a scale from 0 to 10, would you want to add more questions to the database and why?	Very unlikely as there is already a vast number of questions stored within the database. I'd only add more if the content being taught changes.
On a scale from 0 to 10, how likely are you to use this program is lessons and why?	7/10. I'd definitely use it at some point however it would need to be planned for to allow students to use a room which has computers in it.
As the program is at the moment, what do you not like about the program and why?	It could be hard for some students to control the rocket as it is slightly confusing but apart from that, there is nothing else.
Is there anything that you think the program could do to be even better?	It would be good to have a version that works without the server. Obviously, this means you wouldn't be able to have the leader board, but it would allow them to use the program more easily.

From asking these questions, and analysing his responses, I got a good understanding of how well it met what he wanted and expected.

From his responses, I can conclude that:

- As a teacher, he finds the program easy to use.

- For a student, the program is easy to use.
- He is likely to use the program in a lesson to help students revise.
- He believes some students may find it difficult to control the rocket.
- He believes it would be better if it could also be used offline.

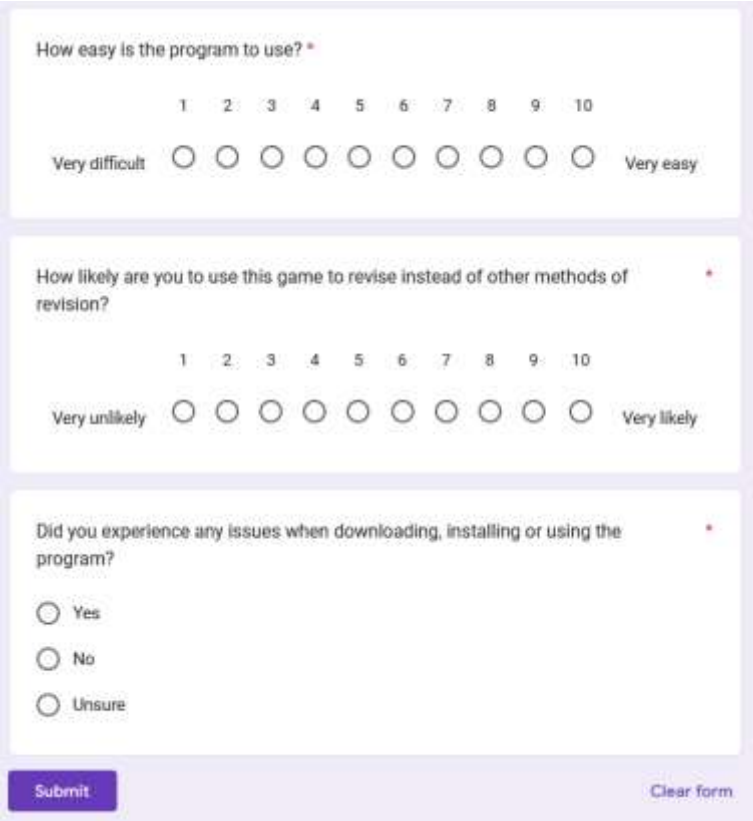
From these points above, I agree that the program is easy to use, and this is reinforced in the online survey below. I do partly agree with his belief that some students may find it difficult to control the rocket, however I don't believe this is enough of a problem that it would need to be changed. This is because from the online survey below, almost everyone who responded agreed the program was very easy to use. Therefore, I agree that some students may find it difficult to control the rocket, but this needs to be part of the game as without it, the game would be too easy and thus it wouldn't be as competitive therefore making it less effective.

Overall, the program does everything he wanted it to and expected it to. This is because he wanted a program which students can use to revise and based off his responses; they would be able to do so. He did also bring my attention to the fact that it would be good if the program could be used offline. I will talk more about this later in the further improvements section.

Feedback from online survey

In addition to seeking feedback from a teacher, I think it's essential to also get feedback from the students who will be using the program in their lessons. Therefore, I created an online survey that asked for their feedback on the program. Before taking the survey, students downloaded and tested the program to gain first-hand experience with it.

Collecting feedback from students is important as they are the ones who will be using it the most. By providing them with a chance to test the program, I can identify any issues or areas for improvement that may not have been apparent during the development or testing stages because of my knowledge of the program. Furthermore, students' feedback can provide valuable insights into the program's effectiveness, ease of use, and overall impact on their revision.

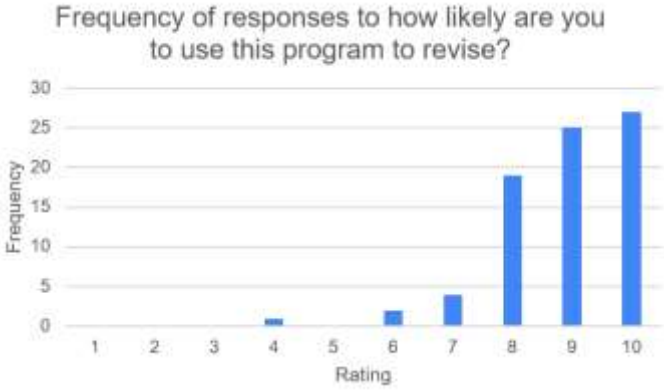


Above shows the questions asked on the survey which I sent out to a large number of students who study GCSE Physics. The survey was filled in by 78 people and below shows a sample of the responses:

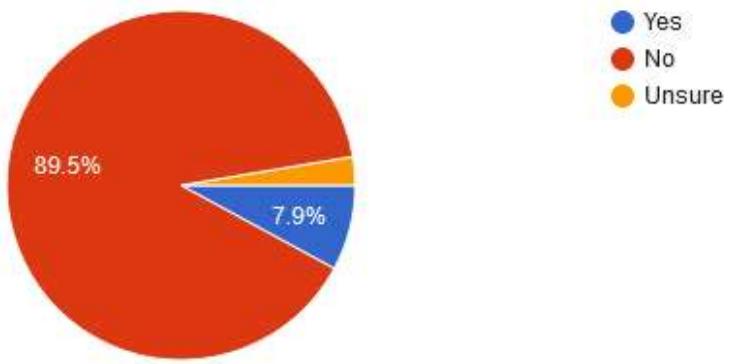
	B	C	D
1	How easy is the program to use?	How likely are you to use this game to revise instead of other methods of revision?	Did you experience any issues when downloading, installing or using the program?
2	9	10	No
3	8	9	No
4	7	7	Yes
5	9	10	No
6	10	8	No
7	9	9	No
8	5	7	No
9	6	6	No
10	9	8	No
11	8	7	No
12	10	9	No

Rather than presenting all the results, I have taken the time to analyse the data collected and produced a series of diagrams that illustrate the key results. By doing this, I intend to provide a clearer summary of the feedback and allow for the results to be understood more easily. By presenting the data in this way, I can highlight the strengths and weaknesses of the program in a visually appealing way which also makes it easier to form an evaluation by using the results.

The above diagram shows the ratings to how easy the program is to use. As you can see, the majority of users think the program is very easy to use and only a very small minority think it was more difficult.



Above, the diagram shows the how likely the respondents are to use the program to revise. The diagram clearly shows that almost everyone would use the program to revise.



The pie chart above was in response to “Did you have any problems when using the program?”, and as you can see the vast majority of respondents didn’t have any problems. By talking to those who took part in the survey, I found that the users who had problems using the program was due to the files corrupting when downloading. This was an issue with the file host I used and not related to the program.

From the survey and the data received, I am able to conclude that almost all students would use the program to revise, and that they also believe it is easy to use.

Overall Conclusion

After analysing the project's objectives thoroughly, I can confidently say that the project has been a success. It has met each objective and done each of them very well. Not only did the objectives set for the project get accomplished, but Mr Mismar's independent feedback also confirms that the program is effective in helping with students' revision. Mr Mismar's feedback also highlighted a few areas where improvements could be made, and these are talked about in greater detail in the next section.

Moreover, the feedback from end-user students was an essential part that also reinforced my conclusion. Their feedback was particularly valuable as it was used to meet one of my objectives. The students agreed they all are “very likely” to use the program to revise and found it very user-friendly, which was a crucial aspect that the project aimed to achieve.

In conclusion, the project has exceeded mine and my client’s expectations and has fulfilled every objective. The positive feedback from Mr Mismar and end-user students reinforces that the project is successful and has effectively achieved its intended purpose.

Possible improvements

Although the project went very well and did everything it was meant to do, there is always rooms for improvement. Therefore, by analysing the feedback, I have been able to conclude what could be improved.

Firstly, the program runs perfectly on any windows device and with many different screen sizes. This is great as schools very commonly use windows on their computers. However, from talking to my client, Mr Mismar, I realised that for my program to be used, teachers would need to book a classroom with computers. As this isn’t always possible, it means my program wouldn’t always be useable when it is needed. Therefore, if I were to have more time to improve the project, I would look to re-make the program so that it can be used on mobile phones. By doing this, it would mean teachers do not need a computer classroom to use the program, thus making it more easily accessible.

Secondly, the other feedback I got from Mr Mismar is that although the whole basis of the program is that users scores are on the leader board to make it competitive, this requires them to connect to the server. He suggested that it if I were to have more time, it would be good for the program to still work without the server. I then went on to think about how this could work and decided the best way would be that the same program can work online or offline. This means it will store the questions client-side which means when it is online, it will be able to update the questions with new questions, edited questions, or removed questions. This allows for the user to benefit from up-

to-date questions while also benefiting from not needing to connect to the server every time they want to use the program.

In conclusion, the improvements I would make are to:

- Amend the program so that it works on mobile devices.
- Amend the program so that it still functions without connecting to the server.

Implementing the improvements

If I were to implement these improvements to the program it is important that I understand how it would be done.

To make the program useable on mobile devices, I would need to be able to convert the python code into an app that can run on mobile devices. Android devices use the file type .APK for apps and IOS devices use .IPA file types. By doing some initial research, I found that the best Python module which would work with these files types is Kivy. Kivy allows the Python program to be converted to a Windows program, MacOS application, Android app, or IOS app. This would be very beneficial to the users of the program as they would then be able to use it on many different devices. Kivy also manages the user inputs, which for mobile phones, would be touch screen. Due to the input methods being very different, I would change the controls for the rocket to be a joystick which the user moves on the screen. All other inputs into the program would be the same as they will still be able to click on the buttons.

For the second improvement, to make it work without a server, the first change would be to make the program download the questions from the server. Once it has the questions stored, every time it connects to the server it will check for any changes to the questions on the server and if there are any, it will update them. It will do this by storing the last time it updated the questions, and comparing this to a log which contains all changes to the questions. Similarly to the questions, the users scores will work in the reverse way. The users scores will be stored on the device, and then when it connects to the server it will send any saved scores to the server. This means the users scores still get put into the leader board making sure the competitive side of the program still exists.

Endnotes

[1] Star in a Box - https://starinabox.lco.global/
[2] N-body Simulations - https://en.wikipedia.org/wiki/N-body_simulation
[3] Space Engine – https://spaceengine.org
[4] Online Survey – The survey can be viewed here: https://forms.gle/bJqe2srGgaBsi67K6 and the results can be viewed here: https://docs.google.com/spreadsheets/d/1A-vrbUYuZAR2RY367JzTB-7kTza9f4OLPR4tHdY1dAE/edit?usp=sharing
[5] Sussex University, Revision methods - https://www.sussex.ac.uk/skills-hub/revision-and-exams
[6] Spacing Learning Over Time by Will Thalheimer - https://www.worklearning.com/wp-content/uploads/2017/10/Spacing_Learning_Over_Time__March2009v1_.pdf
[7] LinkedIn Password Storage - https://www.linkedin.com/pulse/how-store-passwords-database-arvind-kumar
[8] PyWeek (Page 13 and 14) - https://pyweek.readthedocs.io/_/downloads/en/latest/pdf/