|  |  |  |
| --- | --- | --- |
| SDD Assessment Task Project | Aaron Takizad | |
|  | |  |

# Defining and Understanding the problem

This project requires the design and development of a game intended to encourage students of John Purchase Public to learn their multiplication and addition. As per the requirements, the game must be suitable for students in grade 4 and provide motivational feedback. In order to create a suitable application which would allow this, additional research must be invested in determining the interests of the target audience (i.e. students aged 10-12). This research will impact the graphical interface of the program, how students are motivated, and the game difficulty.

The specification requires that if the player achieves a score above 80%, they are presented with a personalised certificate including their name and a game summary. Answers should also be checked upon submission, allowing the user to view their progress through a dynamically updating cumulative tally. In order to provide an element of challenge and competition, the program must contain three proposed levels for both addition and multiplication. These include: easy (1 to 10), medium (1 to 25) and hard (1 to 100). Additionally, a scoreboard featuring the top 3 players for each level must also be included. It must be ensured that numbers are randomly each time the program is run to enable long-term playability of the game by the user, preventing ‘rote learning’ of the answers.

In order to implement this solution, the hardware and software specifications of computers at John Purchase Public must be known in order to ensure the end product is compatible. This includes the operating system, CPU architecture, language frameworks, and programming language runtime packages and their versions. Failure to research this prior to development may result in incompatibility and require recompilation or reprogramming the solution. The software development approach must also be considered. These include agile, prototyping, waterfall, and rapid application development. Due to time constraints and lack of funding, only methodologies that emphasise swift production may be considered for the final project.

Issues of inclusivity must also be considered for gender, economic status, accessibility, and localisation issues. The interface should not be heavily geared towards either gender. Rather, it should be aesthetically appealing to both male and female users. This involves research to determine the interests of both genders of the target age group in order to find similarities. In terms of economic inclusivity, as the students have no financial independence due to their age, the game should be available at no cost to the individual students. Rather, any costs negotiated should be provided by the school. The program should also be accessible to students with disabilities, and thus should not be designed in a manner which conflicts with screen-reader or magnifying functionality. As such accessibility features are already available in all major operating systems, they will not need to be integrated separately. The interface will also be impacted as it must cater to those with hearing impairments. Hence, any sounds implemented should only be used for effect and not required to play the game. Although English is the primary language taught at John Purchase Public, the MySchool demographic report indicates that 59% of students come from a language background other than English, thus text and vocabulary should be kept to a minimum.

# Planning and Designing Software Solutions

My software solution, titled “Maths on Mars” will feature a sci-fi theme through the use of monospaced fonts, typical of Sci-Fi films and games, as well as imagery of spacecraft and the planet Mars placed on a space background. Sound effects and decorations implemented, namely a spaceship-style heads up display, will also portray this theme. I have chosen a sci-fi theme as it is not specifically targeted to either gender, and will engage the identified target market of grade 4 students through animations and effects, as well as being appropriate for older students if required.

## Software Development Approach:

To develop this project, I will be using the Agile software development methodology. This is an appropriate approach due to its focus on rapid delivery and allowing for changes to be made in any stage of the program. The Agile approach also places emphasis on sustainable development, or the ability to maintain a constant pace and taking the simplest approach. This provides the basis by which to determine the approach taken to create the solution, and how to choose an appropriate method of implementing features I am unfamiliar with. Although Agile encourages regular communication with the end user, this may not be required due to the clarity of the specifications outlined by John Purchase Public School. To further simplify the development process, I will make use of the top-down abstraction technique, creating a number of forms as ‘modules’ which will allow me to break down the problem and solve one part at a time. Ideally, the following Gantt chart will be adhered to, allowing a full day before the due date of the project for final changes.

## Choice of Language:

As computers in John Purchase Public School use Microsoft Windows, I will be using the Visual Studio with the Visual Basic.NET language to create the program. This is due to the native platform support and built-in form creation tools available with VB.NET. Additionally, as the language is notoriously easy to learn, and heavily documented on Microsoft Developer Network (MSDN), I will have no issues discovering how to solve any issues I encounter or determining how to implement features I am uncertain of. Namely, I will need to determine how to create unique random numbers, how to store and retrieve top scores, calculate the user’s cumulative score, create multiple forms, and print documents. For backwards compatibility reasons, I only using features available in .NET Framework 3.5 and compiling the program as x86 architecture, ensuring compatibility with older 32-bit and 64-bit machines. Knowledge of XML will also be required, as the specifications explicitly state that high scores must be stored and retrieved. Rather than using a Microsoft SQL Database, which will require a centralised server to be running, I will create a unique XML file for each game configuration. This is preferable to the plain text method as it provides a semantic structure since XML files are standardised. As all player scores are saved and only the top 3 will be displayed, the XML files may easily be imported into a number of other applications which can be used to determine trends in students’ learning providing functionality above what is required.

## Student Motivation:

As motivation is a core focus of the application, I will ensure the user interface is engaging and animated, and prominently features past high scores. Adding a further element of competition, my program will feature a timer – which will also be saved and displayed amongst the high scores – encouraging students to improve in speed as well as accuracy. Upon answering the question, the user will receive positive feedback in prominent text if they answered correctly, and be shown the correct answer otherwise. The program will also enable users whom have achieved 80% and over to print a certificate in recognition of their achievement. These two elements of motivation provide a social incentive to improve by topping the scoreboard, as well as a means to improve through the certificate. Users will also have the flexibility to set a number of questions they are comfortable with attempting. This ensures the player is not discouraged from attempting due to the game’s duration, as they control the length of the game.

## Development Environment, Incremental Backups & Record of Work:

In order to develop the program in Visual Basic, I will use Visual Studio 2015 Community Edition, with MS.NET Framework 3.5 installed. Additionally, as the Agile approach may result in major changes being made in the course of development, I will be using Github as a platform to back up my code and save incrementally. Using Github allows me to restore my code to any “commits”, or checkpoints I define. By leaving “commit messages”, I will detail the changes made between every incremental “commit”, thus utilising my Github repository as a journal log. Visiting the “commits” link will list all the messages written during the course of development, and the ability to download the latest and all previous versions of Maths on Mars. <https://github.com/AaronTakizad/maths_on_mars> The repository also lists acknowledgements for all visual components retrieved from external sources under the creative commons license.

## Screen Layout Diagrams:

|  |  |  |
| --- | --- | --- |
| C:\Users\aaron\AppData\Local\Microsoft\Windows\INetCache\Content.Word\splash.jpg | The game will initiate with a splash screen featuring a blue background image, likely of stars or planets, with the text Maths on Mars. The spacecraft seen in the image will be displayed, and is the same image presented to the user in the main game. The splash screen will automatically disappear after a number of seconds, revealing the main start screen. The choice of font reflect the sci-fi theme. | |
| C:\Users\aaron\AppData\Local\Microsoft\Windows\INetCache\Content.Word\main_display.png | The main start screen presents a form to be filled by the user. The data collected includes the full name, the difficulty level of easy, medium or hard (unlabelled due to canvas size), and a game type of either addition or multiplication. The number of questions will also be asked. A “help” button will also be available. The progress bar at the bottom reflects whether the form is complete, and will alert the user if invalid input is entered through text display (i.e. letters in the “number of questions” field. | |
|  | Should the user choose to click the “help” button, a child window will open with simple instructions on how to fill the form, and the implications of choosing the difficulty levels. The help screen will also direct users to a user manual for how to play the full game. This will feature annotated game screens explaining the function of each element, as well as a written description on how the game is played. The colour red is intentionally used to contrast from the start screen. | |
|  | | Once the user has provided their information and clicked “Let’s go!” the start form will disappear and the appropriate game window will load. The user’s name and difficulty/game type choices will be displayed in a label, as well as a red high scores section, contrasted from the user’s game summary in the bottom right corner. I have chosen to display high scores on the game itself to encourage users to outrank others or their own personal best. A live updating time counter provides incentive for students to work faster. For input, the users will be displayed two random numbers which they must add/multiply. | |
| C:\Users\aaron\AppData\Local\Microsoft\Windows\INetCache\Content.Word\game_over.png | | The summary updates accordingly, and if the user is correct, they are presented with a “Great job!” message, and the spaceship moves closer to mars – proportionally to the number of questions the user chose to answer. Once remaining questions reaches 0, a “Game Over!” message flashes for several seconds. After this, the game automatically closes, revealing a game summary. The background image for these forms will be of a red hue, as to avoid clashing with the red planet and spacecraft. | |
| C:\Users\aaron\AppData\Local\Microsoft\Windows\INetCache\Content.Word\certificate.png | | The game summary displays a congratulatory message in green in front of a blue background, matching that of the start screen. If the user receives over 80%, they receive the message depicted in the image. Otherwise, they will be shown the percentage they did achieve, with the message “Unfortunately, you don’t qualify for a certificate this time” The user will be alerted to click a button should they wish to print out their certificate, which will launch a system dialog box. Should printing functionality be implemented in the final program, a note will also be placed, reminding students that they must ask for permission from their teacher before printing. | |

## Algorithm:

BEGIN start\_screen

Display splash

Wait 1 second

Hide splash

Get player\_name from user

Get question\_count from user

Get help\_required from user

Get game\_difficulty from user

Get game\_type from user

IF help\_required = true

Call help\_module

END IF

CASEWHERE game\_difficulty:

easy:

IF game\_type = addition

Call easy\_addition

ELSE

Call easy\_multiplication

END IF

medium:

IF game\_type = addition

Call easy\_addition

ELSE

Call easy\_multiplication

END IF

hard:

IF game\_type = addition

Call easy\_addition

ELSE

Call easy\_multiplication

END IF

ENDCASE

END

BEGIN help\_module

Display help\_text

END

BEGIN easy\_addition

Let rnd\_num\_one = random integer between 1 and 10

Let rnd\_num\_two = random integer between 1 and 10

Let correct\_counter = 0

Let incorrect\_counter = 0

Let questions\_answered = 0

Get player\_name from start\_screen

Get question\_count from start\_screen

Let questions\_remaining = question\_count

Let timer\_status = true

Let time\_passed = 0

Let correct\_answer = 0

Open easy\_addition\_highscores for read

Sort easy\_addition\_highscores descending

Close easy\_addition\_highscores

Let i = 1

REPEAT

Display easy\_addition\_highscores(i)

i = i + 1

UNTIIL i > 4

WHILE questions\_remaining > 0

Display player\_name

Display correct\_counter

Display incorrect\_counter

Display questions\_answered

Display questions\_remaining

Display time\_passed

IF timer\_status = true

wait 1 second

time\_passed += 1

END IF

correct\_answer = rnd\_num\_one + rnd\_num\_two

Get answer from user

IF answer = correct\_answer

Display "Correct!"

correct\_counter = correct\_counter + 1

ELSE

Display "Incorrect!"

Display correct\_answer

incorrect\_counter = incorrect\_counter + 1

END IF

questions\_remining = questions\_remaining - 1

questions\_answered = total\_questions\_answered + 1

rnd\_num\_one = random integer between 1 and 10

rnd\_num\_two = random integer between 1 and 10

END WHILE

Display "Game over!"

Open easy\_addition\_highscores for append

IF correct\_counter > easy\_additon\_highscores(3)

Write easy\_addition\_highscores from player\_name, correct\_counter, time\_passed

END IF

Close easy\_addition\_highscores

Call game\_summary

END

BEGIN medium\_addition

Let rnd\_num\_one = random integer between 1 and 25

Let rnd\_num\_two = random integer between 1 and 25

Let correct\_counter = 0

Let incorrect\_counter = 0

Let questions\_answered = 0

Get player\_name from start\_screen

Get question\_count from start\_screen

Let questions\_remaining = question\_count

Let timer\_status = true

Let time\_passed = 0

Let correct\_answer = 0

Open medium\_addition\_highscores for read

Sort medium\_addition\_highscores descending

Close medium\_addition\_highscores

Let i = 1

REPEAT

Display medium\_addition\_highscores(i)

i = i + 1

UNTIIL i > 4

WHILE questions\_remaining > 0

Display player\_name

Display correct\_counter

Display incorrect\_counter

Display questions\_answered

Display questions\_remaining

Display time\_passed

IF timer\_status = true

wait 1 second

time\_passed += 1

END IF

correct\_answer = rnd\_num\_one + rnd\_num\_two

Get answer from user

IF answer = correct\_answer

Display "Correct!"

correct\_counter = correct\_counter + 1

ELSE

Display "Incorrect!"

Display correct\_answer

incorrect\_counter = incorrect\_counter + 1

END IF

questions\_remining = questions\_remaining - 1

questions\_answered = total\_questions\_answered + 1

rnd\_num\_one = random integer between 1 and 25

rnd\_num\_two = random integer between 1 and 25

END WHILE

Display "Game over!"

Open medium\_addition\_highscores for append

IF correct\_counter > medium\_additon\_highscores(3)

Write medium\_addition\_highscores from player\_name, correct\_counter, time\_passed

END IF

Close medium\_addition\_highscores

Call game\_summary

END

BEGIN hard\_addition

Let rnd\_num\_one = random integer between 1 and 100

Let rnd\_num\_two = random integer between 1 and 100

Let correct\_counter = 0

Let incorrect\_counter = 0

Let questions\_answered = 0

Get player\_name from start\_screen

Get question\_count from start\_screen

Let questions\_remaining = question\_count

Let timer\_status = true

Let time\_passed = 0

Let correct\_answer = 0

Open hard\_addition\_highscores for read

Sort hard\_addition\_highscores descending

Close hard\_addition\_highscores

Let i = 1

REPEAT

Display hard\_addition\_highscores(i)

i = i + 1

UNTIIL i > 4

WHILE questions\_remaining > 0

Display player\_name

Display correct\_counter

Display incorrect\_counter

Display questions\_answered

Display questions\_remaining

Display time\_passed

IF timer\_status = true

wait 1 second

time\_passed += 1

END IF

correct\_answer = rnd\_num\_one + rnd\_num\_two

Get answer from user

IF answer = correct\_answer

Display "Correct!"

correct\_counter = correct\_counter + 1

ELSE

Display "Incorrect!"

Display correct\_answer

incorrect\_counter = incorrect\_counter + 1

END IF

questions\_remining = questions\_remaining - 1

questions\_answered = total\_questions\_answered + 1

rnd\_num\_one = random integer between 1 and 100

rnd\_num\_two = random integer between 1 and 100

END WHILE

Display "Game over!"

Open hard\_addition\_highscores for append

IF correct\_counter > hard\_additon\_highscores(3)

Write hard\_addition\_highscores from player\_name, correct\_counter, time\_passed

END IF

Close hard\_addition\_highscores

Call game\_summary

END

BEGIN easy\_multiplication

Let rnd\_num\_one = random integer between 1 and 10

Let rnd\_num\_two = random integer between 1 and 10

Let correct\_counter = 0

Let incorrect\_counter = 0

Let questions\_answered = 0

Get player\_name from start\_screen

Get question\_count from start\_screen

Let questions\_remaining = question\_count

Let timer\_status = true

Let time\_passed = 0

Let correct\_answer = 0

Open easy\_multiplication\_highscores for read

Sort easy\_multiplication\_highscores descending

Close easy\_multiplication\_highscores

Let i = 1

REPEAT

Display easy\_multiplication\_highscores(i)

i = i + 1

UNTIIL i > 4

WHILE questions\_remaining > 0

Display player\_name

Display correct\_counter

Display incorrect\_counter

Display questions\_answered

Display questions\_remaining

Display time\_passed

IF timer\_status = true

wait 1 second

time\_passed += 1

END IF

correct\_answer = rnd\_num\_one + rnd\_num\_two

Get answer from user

IF answer = correct\_answer

Display "Correct!"

correct\_counter = correct\_counter + 1

ELSE

Display "Incorrect!"

Display correct\_answer

incorrect\_counter = incorrect\_counter + 1

END IF

questions\_remining = questions\_remaining - 1

questions\_answered = total\_questions\_answered + 1

rnd\_num\_one = random integer between 1 and 10

rnd\_num\_two = random integer between 1 and 10

END WHILE

Display "Game over!"

Open easy\_multiplication\_highscores for append

IF correct\_counter > easy\_multiplication\_highscores(3)

Write easy\_multiplication\_highscores from player\_name, correct\_counter, time\_passed

END IF

Close easy\_multiplication\_highscores

Call game\_summary

END

BEGIN medium\_multiplication

Let rnd\_num\_one = random integer between 1 and 25

Let rnd\_num\_two = random integer between 1 and 25

Let correct\_counter = 0

Let incorrect\_counter = 0

Let questions\_answered = 0

Get player\_name from start\_screen

Get question\_count from start\_screen

Let questions\_remaining = question\_count

Let timer\_status = true

Let time\_passed = 0

Let correct\_answer = 0

Open medium\_multiplication\_highscores for read

Sort medium\_multiplication\_highscores descending

Close medium\_multiplication\_highscores

Let i = 1

REPEAT

Display medium\_multiplication\_highscores(i)

i = i + 1

UNTIIL i > 4

WHILE questions\_remaining > 0

Display player\_name

Display correct\_counter

Display incorrect\_counter

Display questions\_answered

Display questions\_remaining

Display time\_passed

IF timer\_status = true

wait 1 second

time\_passed += 1

END IF

correct\_answer = rnd\_num\_one + rnd\_num\_two

Get answer from user

IF answer = correct\_answer

Display "Correct!"

correct\_counter = correct\_counter + 1

ELSE

Display "Incorrect!"

Display correct\_answer

incorrect\_counter = incorrect\_counter + 1

END IF

questions\_remining = questions\_remaining - 1

questions\_answered = total\_questions\_answered + 1

rnd\_num\_one = random integer between 1 and 25

rnd\_num\_two = random integer between 1 and 25

END WHILE

Display "Game over!"

Open medium\_multiplication\_highscores for append

IF correct\_counter > medium\_multiplication\_highscores(3)

Write medium\_multiplication\_highscores from player\_name, correct\_counter, time\_passed

END IF

Close medium\_multiplication\_highscores

Call game\_summary

END

BEGIN hard\_multiplication

Let rnd\_num\_one = random integer between 1 and 100

Let rnd\_num\_two = random integer between 1 and 100

Let correct\_counter = 0

Let incorrect\_counter = 0

Let questions\_answered = 0

Get player\_name from start\_screen

Get question\_count from start\_screen

Let questions\_remaining = question\_count

Let timer\_status = true

Let time\_passed = 0

Let correct\_answer = 0

Open hard\_multiplication\_highscores for read

Sort hard\_multiplication\_highscores descending

Close hard\_multiplication\_highscores

Let i = 1

REPEAT

Display hard\_multiplication\_highscores(i)

i = i + 1

UNTIIL i > 4

WHILE questions\_remaining > 0

Display player\_name

Display correct\_counter

Display incorrect\_counter

Display questions\_answered

Display questions\_remaining

Display time\_passed

IF timer\_status = true

wait 1 second

time\_passed += 1

END IF

correct\_answer = rnd\_num\_one + rnd\_num\_two

Get answer from user

IF answer = correct\_answer

Display "Correct!"

correct\_counter = correct\_counter + 1

ELSE

Display "Incorrect!"

Display correct\_answer

incorrect\_counter = incorrect\_counter + 1

END IF

questions\_remining = questions\_remaining - 1

questions\_answered = total\_questions\_answered + 1

rnd\_num\_one = random integer between 1 and 100

rnd\_num\_two = random integer between 1 and 100

END WHILE

Display "Game over!"

Open hard\_multiplication\_highscores for append

IF correct\_counter > hard\_multiplication\_highscores(3)

Write hard\_multiplication\_highscores from player\_name, correct\_counter, time\_passed

END IF

Close hard\_multiplication\_highscores

Call game\_summary

END

BEGIN game\_summary

IF correct\_counter/question\_count x 100 >= 80

Display “You’re eligible for a certificate!”

ELSE

Display “Unfortunately, you don’t qualify for a certificate this time.”

END IF

END

## Justification of Algorithm Data Types & Data Structures:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Explanation** |
| player\_name | String | A string is the appropriate data type for storing text. This is a global variable, so other forms will access and use its contents. |
| question\_count | Integer | The number of questions a user chooses to answer can only be a whole number. |
| help\_required | Boolean | The user can *either* choose to display the help form, *or* not. |
| game\_difficulty | String | The values are stored as either “easy”, “medium” or “hard” |
| game\_type | String | The values are stored as either “addition” or “multiplication |
| rnd\_num\_one | Integer | A randomly generated integer. The program only intends to produce integers for the user to add/multiply. |
| rnd\_num\_two | Integer | Another randomly generated integer. The program only intends to produce integers for the user to add/multiply. |
| correct\_counter | Integer | The number of questions a user answers correctly can only be a whole number. |
| incorrect\_counter | Integer | The number of questions a user answers incorrectly can only be a whole number. |
| questions\_answered | Integer | The number of questions answered can only be an integer. |
| questions\_remaining | Integer | The number of questions remaining can only be an integer. |
| timer\_status | Boolean | The timer is *either* enabled, *or* disabled. |
| time\_passed | Integer | The timer increments in seconds, so the value can only be a whole number. |
| answer | Integer | As the two numbers can only be whole numbers, their sum or product can only be an integer. |
| correct\_answer | Integer | As the two numbers can only be whole numbers, their sum or product can only be an integer. |
| easy\_addition\_highscores | Data dictionary | The variable reads and stores values from an XML file containing the high scores as a data dictionary, associating the player’s name with their ranking and game duration. |

## Justification of Algorithm Conditional Statements (Control Flow):

### Pre-test Loop

Each game module in the algorithm utilises a while loop to repeat the process of asking questions, asking for user input, and checking the result until the condition is met (i.e. there are no questions remaining).

### Post-test Loop

All game modules made use of a REPEAT UNTIL or “post-test loop” in order to display the top three high scores retrieved from the XML file read by the easy\_addition\_highscores variable.

### Binary Selection

Binary selections were used in the algorithm to perform a certain action if a condition was true, and a different action otherwise. In the start\_screen module, a binary selection was used to determine whether or not the help screen was to be shown. Likewise, another IF statement was used in determining whether or not to increment one second to the timer, if a user answered a question correctly, and whether a user is presented with a certificate upon game completion.

### Multiway Selection

A multiway selection was used in the start\_screen module when determining which game module to call. This is because a user’s difficulty option as well as game type impacted the decision, leading to more than 2 possible outcomes.

## Algorithm Desk Check:

The following desk check should result in a user playing an easy addition game of 6 questions, entering their name as “Harry”, and achieving a score of 50%. Hence, this example will make use of all variables including incorrect\_counter. As the timer cannot be simulated, it will be assumed that the game takes a total of 45 seconds and that the high scores XML file is empty. The easy\_addition module was selected solely to provide easier inputs for the desk check and reduce the likelihood of human error. As the program follows a top-down abstraction approach, all the forms utilise the same logic, with minor differences in the XML file read, the range of the numbers, and whether the correct\_answer variable is determined by adding or multiplying the two random numbers. Hence, should the logic of one module work, the subsequent modules will also be functional. The test data chosen for the random numbers must be between the range of 1-10 inclusive. However, the initial value is 0, reflecting its value whilst the player is still on the start screen and its values have not yet been assigned to random numbers.

|  |  |
| --- | --- |
| player\_name | Harry |
| question\_count | 6 |
| help\_required | False |
| game\_difficulty | Easy |
| game\_type | Addition |
| rnd\_num\_one | ~~0~~ ~~1~~ ~~4~~ ~~2~~ ~~1~~ ~~9~~ 6 |
| rnd\_num\_two | ~~0~~ ~~2~~ ~~3~~ ~~6~~ ~~1~~ ~~1~~ 2 |
| correct\_counter | ~~0~~ ~~1~~ ~~1~~ ~~2~~ ~~3~~ ~~3~~ 3 |
| incorrect\_counter | ~~0~~ ~~0~~ ~~1~~ ~~1~~ ~~1~~ ~~2~~ 3 |
| questions\_answered | ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 |
| questions\_remaining | ~~6~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0 |
| timer\_status | ~~False~~ ~~True~~ False |
| time\_passed | ~~0~~ ~~10~~ ~~25~~ ~~30~~ ~~35~~ ~~40~~ 45 |
| answer | ~~0~~ ~~3~~ ~~5~~ ~~8~~ ~~2~~ ~~8~~  7 |
| correct\_answer | ~~0~~ ~~3~~ ~~7~~ ~~8~~ ~~2~~ ~~10~~ 8 |
| easy\_addition\_highscores | ~~None~~ “Harry, 3, 45” |
| OUTPUT | ~~“”~~ ~~“Correct!”~~ ~~“Incorrect!”~~ ~~“Correct!”~~ ~~“Correct!”~~ ~~“Incorrect!”~~ ~~“Incorrect!”~~ ~~“Game Over!”~~ “Unfortunately, you don’t qualify for a certificate this time.” |

## Testing and Evaluation:

I periodically submit my code to Github, detailing the changes made between these versions. This enabled me to add one new feature at a time and test it before moving it, whilst providing a safety net to restore my work if this were to create errors. To test my start screen, I entered different possible data types in the text fields. This revealed program crashes if a string was entered in the “Number of Questions” field. To resolve this, I wrote code to validate the field, alerting the user that a valid number must be entered. In testing my help and summary screens, I realised they appear to the upper-left corner of the screen. I resolved this through changing the relevant property to “centre screen”. Testing also revealed that a user may not submit an answer by pressing the enter key, so this functionality was also implemented. Committing my changes via Git also enabled me to log any issues I found and continue working on the program, enabling me to keep track of any issues that may arise. This was particularly helpful in keeping track of errors caused by my method of moving the spaceship proportionally to the number of questions requested.