**A Level computer Science**

Component 3

Space Game

Logo

Description automatically generated

By: Louis Pattern

For: A. Issa

**Stoke on Trent Sixth Form College**

Table of Contents

[1.1 Introduction 7](#_Toc129255890)

[1.2 Problem Identification 7](#_Toc129255891)

[1.3 Why the problem is suited to a computational solution 8](#_Toc129255892)

[1.4 Stakeholders analysis 8](#_Toc129255893)

[1.5 Research 9](#_Toc129255894)

[Defender 9](#_Toc129255895)

[Space Invaders 11](#_Toc129255896)

[1.6 Stakeholder Consultation 13](#_Toc129255897)

[1.7 Features of the proposed solution 14](#_Toc129255898)

[1.8 The requirements of the solution 15](#_Toc129255899)

[1.9 Success Criteria 16](#_Toc129255900)

[1.10 Limitations of the solution 17](#_Toc129255901)

[1.11 Hardware and software requirements 18](#_Toc129255902)

[2.1 Introduction 20](#_Toc129255903)

[2.2 Decomposition of the problem 21](#_Toc129255904)

[2.2.1 Decomposition Diagram 21](#_Toc129255905)

[2.2.2 Data Flow Diagram 22](#_Toc129255906)

[Login data flow diagram: 22](#_Toc129255907)

[Game data flow diagram: 23](#_Toc129255908)

[2.2.3 Input Process Output 23](#_Toc129255909)

[2.3 How All Solution Parts are Linked 25](#_Toc129255910)

[2.3.1 State Diagram of the different forms/parts 25](#_Toc129255911)

[For the login system: 25](#_Toc129255912)

[For the game: 25](#_Toc129255913)

[2.3.2 Classes 26](#_Toc129255914)

[Game sprites UML class diagram: 26](#_Toc129255915)

[2.4 Database Design 27](#_Toc129255916)

[2.4.1 Normalisation 27](#_Toc129255917)

[2.4.2 Data Dictionary 27](#_Toc129255918)

[2.4.3 Entity Relationship Diagram 28](#_Toc129255919)

[2.4.4 SQL Pseudocode 28](#_Toc129255920)

[2.5 Design of Main Parts of the Solution 29](#_Toc129255921)

[2.5.1 Part ONE - Login: 29](#_Toc129255922)

[2.5.1.1 Form Design and Layout 29](#_Toc129255923)

[Login window 29](#_Toc129255924)

[2.5.1.2 Validation rules 30](#_Toc129255925)

[2.5.1.3 Algorithms and Pseudocode for Login 30](#_Toc129255926)

[Cancel button: 31](#_Toc129255927)

[Login window class: 32](#_Toc129255928)

[2.5.1.4 Test Plan for PART ONE 32](#_Toc129255929)

[2.5.2 Part TWO – Admin Panel: 33](#_Toc129255930)

[2.5.2.1 Form Design and Layout 33](#_Toc129255931)

[2.5.2.2 Validation rules 33](#_Toc129255932)

[2.5.2.3 Pseudocode for Admin Panel 34](#_Toc129255933)

[2.5.2.4 Login and Admin UML class diagram 35](#_Toc129255934)

[2.5.2.5 Test Plan for PART TWO 35](#_Toc129255935)

[2.5.3 Part THREE – Game menus: 38](#_Toc129255936)

[2.5.3.1 Game Menus Design and Layout 38](#_Toc129255937)

[Game main menu 38](#_Toc129255938)

[Game settings menu 39](#_Toc129255939)

[Highscores Menu 40](#_Toc129255940)

[2.5.3.2 Justification of Validation rules 40](#_Toc129255941)

[2.5.3.3 Algorithms and Pseudocode for Menus 41](#_Toc129255942)

[Main menu flowchart 43](#_Toc129255943)

[2.5.3.4 Key Variables/Classes 44](#_Toc129255944)

[2.5.3.5 Test Plan for PART THREE 44](#_Toc129255945)

[2.5.4 Part FOUR – Design of Main Game: 45](#_Toc129255946)

[2.5.4.1 Window Design and Layout 45](#_Toc129255947)

[2.5.4.2 Algorithms for Main Game 46](#_Toc129255948)

[Main flowchart 46](#_Toc129255949)

[Player movement flowchart 46](#_Toc129255950)

[2.5.4.3 Key Classes for the game 50](#_Toc129255951)

[2.5.4.4 Test Plan for Main Game 50](#_Toc129255952)

[2.5.5 Part FIVE – Error Handling 50](#_Toc129255953)

[2.5.5.1 Form Design and Layout 50](#_Toc129255954)

[2.5.5.2 Pseudocode for Error handling 51](#_Toc129255955)

[Validation module pseudocode: 52](#_Toc129255956)

[2.5.5.3 Test Plan for Error handling 54](#_Toc129255957)

[2.6 Stakeholders involvement 56](#_Toc129255958)

[Part TWO 56](#_Toc129255959)

[Revised Admin Window 56](#_Toc129255960)

[Pseudocode 57](#_Toc129255961)

[Part FOUR 57](#_Toc129255962)

[Revised Game UI 57](#_Toc129255963)

[Key classes 58](#_Toc129255964)

[2.7 Testing plan to inform evaluation 58](#_Toc129255965)

[Integration testing: 58](#_Toc129255966)

[Scenario testing: 59](#_Toc129255967)

[3.1 Introduction 61](#_Toc129255968)

[Modules 61](#_Toc129255969)

[3.2.1 Stage 1: Building the Error Handling 62](#_Toc129255970)

[3.2.1.1 Building General Modules (Error Handling) 62](#_Toc129255971)

[Imported modules 63](#_Toc129255972)

[Version 1 63](#_Toc129255973)

[Version 1 Testing: 63](#_Toc129255974)

[improvement 64](#_Toc129255975)

[Version 2 Testing: 64](#_Toc129255976)

[Version 2 Results: 64](#_Toc129255977)

[Version 3 65](#_Toc129255978)

[Version 3 Testing: 65](#_Toc129255979)

[Version 3 Results: 66](#_Toc129255980)

[3.2.1.2 Feedback from Stakeholder 68](#_Toc129255981)

[3.2.2 Stage 2: Building the Database 68](#_Toc129255982)

[3.2.2.1 SQLITE + Python Code 68](#_Toc129255983)

[Imported modules 68](#_Toc129255984)

[Version 1 68](#_Toc129255985)

[Version 1 Testing: 69](#_Toc129255986)

[Version 2 69](#_Toc129255987)

[Version 2 Testing: 70](#_Toc129255988)

[Automatic Testing 72](#_Toc129255989)

[Version 2 Results: 73](#_Toc129255990)

[Version 3 Testing: 76](#_Toc129255991)

[Version 3 Results: 77](#_Toc129255992)

[3.2.2.2 Feedback from Stakeholder (if appropriate) 77](#_Toc129255993)

[3.2.2.3 Review 78](#_Toc129255994)

[Version 4 78](#_Toc129255995)

[Version 4 Results: 79](#_Toc129255996)

[3.2.3 Stage 3: Building Login 79](#_Toc129255997)

[3.2.3.1 Code for Login (including validation) 80](#_Toc129255998)

[Imported modules 80](#_Toc129255999)

[Version 1 80](#_Toc129256000)

[Version 1 Testing: 81](#_Toc129256001)

[Version 1 Results: 82](#_Toc129256002)

[Version 2 84](#_Toc129256003)

[Version 2 Testing: 86](#_Toc129256004)

[Version 2 Results: 86](#_Toc129256005)

[Version 3 87](#_Toc129256006)

[Version 3 Testing: 89](#_Toc129256007)

[Version 3 Results: 90](#_Toc129256008)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 93](#_Toc129256009)

[Version 4 94](#_Toc129256010)

[Version 4 Testing 96](#_Toc129256011)

[3.2.3 Stage 4: Building …….. PART FOUR …. 96](#_Toc129256012)

[3.2.3.1 Code For … part FOUR… (including validation) 96](#_Toc129256013)

[3.2.3.2 Testing …. Part FOUR ….including validation 97](#_Toc129256014)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 97](#_Toc129256015)

[3.2.3.4 Review 97](#_Toc129256016)

[3.2.4 Stage 2: Building …….. PART FIVE …. 97](#_Toc129256017)

[3.2.4.1 Code for … Part FIVE… including validation 97](#_Toc129256018)

[3.2.4.2 Testing …. Part FIVE ….including validation 97](#_Toc129256019)

[3.2.4.3 Feedback from Stakeholder (if appropriate) 97](#_Toc129256020)

[3.2.4.4 Review 98](#_Toc129256021)

[3.3 Final Review, Improvements and Corrective Actions 98](#_Toc129256022)

[4.1 Introduction 101](#_Toc129256023)

[4.2 Testing to inform evaluation 101](#_Toc129256024)

[4.2.1 Testing 101](#_Toc129256025)

[4.2.2 Feedback from Stakeholders 102](#_Toc129256026)

[4.3 Evaluation 103](#_Toc129256027)

[4.4 Evaluating usability features 103](#_Toc129256028)

[4.5 Limitations and Maintenance 103](#_Toc129256029)

Chapter One: Analysis of the problem

## 1.1 Introduction

Many students, both in my college and from other colleges or high schools suffer from stress. I have discussed this issue with my stakeholders (a number of students) and decided that I can reduce this issue by making a videogame for them.   
This game will help to solve the issue of stress with students because it will help give them some time to relax in their free time or during breaks rather than being under constant pressure from exams, homework and coursework. Even a simple 2D game like the one I aim to create can be a temporary but helpful relief from stress. Students would be able to play the game in their free time as a source of entertainment and escapism.   
Therefore, this project will help to reduce the problem of stress by giving them a recreational game that they can play in their free time.  
The main goal of the project is to be a source of entertainment for my stakeholders during their free periods or outside of school entirely. The game I will make will have the genre of 2D shooter and will be aimed at being suitable for college students and children and has the theme of science fiction and space.  
The game will have both options for a single player fighting the computer and multiplayer for one player versus another locally, allowing users to play the game how they prefer. The game will involve each player controlling a spaceship on each half of the screen and shooting enemies approaching them. There will be a number of levels in single player, increasing in difficulty as the player progresses.  
In addition, the game as well as the login system will be customisable and have a variety of different settings. This will allow my stakeholders to further enjoy the game by changing the setting to those which best suite them. I will further address my stakeholders in later sections, surveying them for what features they would like to see in the game.

## 1.2 Problem Identification

College or high school life can be stressful, especially when exams are approaching. This can lead to many students becoming stressed to the point where they find it hard to focus on work. During their free time they may be constantly worrying about coursework or revision. It is also important that students like this allow themselves to have a break occasionally, but many find that they having nothing that they want to do or would be entertained doing in their free time.

Therefore, boredom is also another problem that negatively impacts college students. Students that often find that they have nothing to do outside of school or college could feel drained or exhausted when they are in school. Thus, this could lead to these students being less motivated and performing worse in the classroom.

A frequent existing solution to this is boardgames. However, these kinds of games often require 2 or more players and are static, which could lead to players becoming frustrated and dissatisfied with this solution.

Videogames could become a better solution to this problem by helping relieve the stress students experience in their free time by providing a means of escapism from their school life. Stress is also a major problem for young adults, so my project would not only be targeting teenagers. Videogames have also been proven to have other benefits. For example, improving focus and reaction time.

Most retro games are fully single player experiences, with no way to interact directly with another human player in the game aside from competing for a spot on a scoreboard after game completion. My project aims to go against this convention by allowing players to directly compete against each over in real time with their scores being tracked and displayed clearly on screen.

Furthermore, a lot of 2D shooters feature either just a single large level, a limited number of levels or there is little variation between levels. If there is little change in enemies or combat between levels then the user will become bored due to lack of challenge or stimulation. However, if there is no visual variation between levels then the user can also become bored, leading to the problems of boredom and stress being unsolved by the attempted solution.

## **1.3 Why the problem is suited to a computational solution**

This problem is amenable by a computational approach because it is a videogame, thus has to involve the user interacting with a computer. My project being a videogame rather than a non-computational solution such as a board game has numerous advantages. Firstly, the computer can process the user’s inputs and perform tasks much faster than a human can interact with a non-computational game.  
Using a computer also allows for high accuracy in calculations that are done quickly.  
Additionally, animations and movement for a videogame can be easily displayed by a computer screen whereas in a normal game they cannot. Colours in videogames can also be adjusted, allowing the option for the user to customise the colours for to best suite them, for example enabling colourblind mode.   
Using computational technology allows for me to utilise a wide range of abilities and features in the game.  
Finally, most people with disabilities are still able to play because usual input methods such as mouse and keyboard do not require much movement.  
In addition, the problem is suited to a computational approach because it has been attempted many times before my project. The technology used for developing 2D games is already greatly understood and there already exists a large number of computer games that have been developed in the past and are similar to my project, therefore this means the format has been tried and tested.  
Hence, this would allow my project to be more accurate and efficient than a non-computational project solution that has not been attempted by other people before. It would also give me an opportunity to improve of these past solutions while still incorporating some features that I think are beneficial to my game.

## 1.4 Stakeholders analysis

My stakeholders for this project will be, college students at a variety of colleges, a high school student and two adults. I will give each of my stakeholders a rundown on the proposed features of the game and interview them for feedback and criticism. If any of the stakeholders suggest a way to improve the game or a new feature to add to the game, I will try to implement it.

The high school and college students I have selected are casual gamers, playing a variety of videogame genres. Some prefer to play mobile games on their phone whereas others prefer to play on their home consoles. The few that owned their own PC build at home would consider themselves PC gamers.

One of the adults is a casual gamer and has their own gaming PC which they occasionally use to play games on. The other adult has little interest in videogames and does not play them at all in their free time. They also have limited experience with computers.

## 1.5 Research

### Defender

One existing videogame of a similar format is **Defender,** an arcade game from 1981: <https://en.wikipedia.org/wiki/Defender_(1981_video_game)>  
This game is a side scrolling shooter where the player has the objective of shooting aliens on another planet. The player can move in all directions, with up and down moving the ship directly and left and right moving the terrain. However, because of technical limitations of the time, the background is extremely simplistic, with the ground consisting of a single zig-zagging line.



Fig. 1 - Defender Gameplay

The game is also single player only, something which I aim to improve upon in my game. Different enemy types feature in Defender, with each alien behaving differently and awarding a different number of points when the player destroys them. These enemies are also vastly different visually, allowing the player to easily differentiate between them. This visual difference also allows new players to easily learn the behaviour and mechanics of each enemy. Another gameplay element of Defender is that the player can rescue captured humans by shooting pods.   
While Defender was an arcade game rather than a computer game that be run on windows without the use of an emulator, I still think it has some interesting features that I could implement in my own game.

**Parts that I may apply to my solution:**

The player’s inputs being used to control a spaceship, both movement and shooting. The scoring system is another good feature of this game, with enemies that are harder to kill rewarding the player with more points. This incentivises the player to attack new, harder variants of aliens and save more pods rather than just killing the same basic enemies repeatedly.   
Another feature that I will incorporate into my solution is the fact that the player is given multiple lives, which are displayed in the UI at the top of the screen. This feature makes the game easier because instead of getting a game over after one attack hitting the player, the player can survive multiple hits until the number of lives runs out.  
Another feature that I will include is the use of a game over screen when the player reaches 0 lives. From this screen the player can choose to exit or play again.



Fig. 2 – Defender Scoring System

**Disadvantages:**

Defender is a single-player only game, meaning it doesn’t have a 2-player mode like I plan to feature in the final version of my game. The game also obviosly cannot run on modern PCs without the user having to use additional software such as an emulator.

### Space Invaders

Another existing solution to the problem, is **Space Invaders, originally an arcade game but has been remade many times to be playable on a large variety of different systems.**

<https://en.wikipedia.org/wiki/List_of_Space_Invaders_video_games>



*Fig. 3 – Orignal Space invaders game*

Space invaders is another well known example of a space shooting game that I have researched. The original game as well as its remakes features the player controlling a small turret at the bottom of the screen that can move left and right as well as shooting verically upwards. The main objective of the game is to shoot all of the aliens while they move left and right before they descend and reach the bottom of the screen.   
In addition to this, some aliens also attempt to shoot the player. To help with this, the player is given a number of lives which (similar to Defender) allow the player to take multiple hits before the player loses and a game over screen is displayed.  
However, the player is also given a number of shields (the four green barriers just above the player seen in Figure 3). These shields will absorb incoming bullets but will suffer damage and can be fully destroyed. This includes the players own bullets. The shield will be damaged from the top if hit by an alien bullet and damaged from the bottom if the player shoots under the shield.  
The player’s score is tracked at the top of the screen and the player is awared score for killing aliens, with the higher up aliens being worth more points. There is also a ‘boss’ alien ship that occassionally appears at the very top of the screen and awards the player with a large amount of points when killed, but moves rapidly and is hard to hit.

When the player shoots all of the aliens on one level, they progress to the next level. This cycle continues indefinetely until the player runs out of lives, with aliens becoming faster and levels becoming harder as the player progresses.

**Parts that I may apply to my solution:**

As discussed before, I will apply the feature of the player receiving multuple lives, with the number of lives remaining being clearly displayed in the game’s UI. This allows the player to make a small mistake without being instantly punished with a game over, making the game more enjoyable and less frustrating. For my game, having more lives will make it easier for the player to complete the entire game Another feature I will adapt to my own solution is the presence of a scoring system.   
The player recieves score from shooting aliens and their score is saved when the game ends and the highest score ever achieved is visible. This allows playing the game to be more fun as the player is given an end goal of breaking the highscore and setting a new record.

**Limitations:**

**One main limitation of the game is that the player can only move horizontally. This means that the player can only move in 2 directions, left and right. This means the player is limited in their options when dodging the alien’s bullets. In my solution, I will aim to resolve this by including vertical as well as horizontal movement. This would allow the player ship to move in 8 directions (including diagonals) rather than 2.  
Another disadvantage is that the player is very limited in how they can play the game. In the original Space Invaders, there is only one way to play the game – progress through the endlessly cycling single-player levels. In my game, I will give the user multiple options for what game mode they want to play and allow them to select this from a menu.  
The original game is also limited visually, with the background being constantly black and entities being made from only solid colours. This is expected due to the age of the game and the technical limitations at the time. This will be easily fixed in my solution and I may feature the background as dynamic, changing depending what level the user is on. This will make the user less likely to get bored playing the game.**

## 1.6 Stakeholder Consultation

I have interviewed **a range of my stakeholders**, asking them for feedback on the game towards the end of the interview. Their responses will be summarised and placed in quotes.

**Interview with College Student - Francis Kip**

* **Have you ever played a videogame?**“Yes”
* **Do you play games a lot in your free time?**“Yes, when I’m not focused on homework or revision.”
* **What kind of games do you like to play?**“I mainly play first person shooters and platformers but there aren’t really any genres I don’t like”
* **Do you think playing games can be beneficial?**“Of course, they can help me relax and take a break from schoolwork. Multiplayer games also allow me to socialise and have fun with my friends.”
* **(After having showed interviewee proposed features) What features of my proposed game did you like?**“I liked the ability to directly play against another player. I also liked the idea of a leader board with the best players at the top.”
* **What features did you think could be improved?**“I think the game should have the ability to go full screen. I mostly don’t play games windowed. A pause feature would also be good.”
* **What colours do you think should be used?**“Black for the background and overall, nothing too bright.”

**Interview with Adult – Kurk Milo**

* **Have you ever played a videogame?**“I played some arcade games when I was younger”
* **Do you play games a lot in your free time?**“No.”
* **Do you think playing games can be beneficial?**“Maybe as long as they are played in moderation”
* **(After having showed interviewee proposed features) What features of my proposed game did you like?**“I liked the retro arcade style.”
* **What features did you think could be improved?**“Add music to the game”
* **What colours do you think should be used?**“Blue for the player and red for enemies”

## 1.7 Features of the proposed solution

After consultation with stakeholders, the following are identified as **main features** of the solution (game):

* Single player mode where the player fights against enemies controlled by AI.
* Multiplayer ‘versus mode’ where one player fights another. This will give my project a unique feature compared to other scrolling shooter games. This would also allow for friendly competition that isn’t possible in only single player games. There is a range of projectiles depending on whether they are being fired by an enemy or a player.
* Log in screen where the user enters their username and password, a database it checked to make sure they are correct. If they are, the user will be logged in and taken to the game.
* High score leader board, featuring the player’s name followed by score. This information will be stored in a database and will be displayed when a player completes the game or they manually select ‘high scores’ from the main menu. Scores should be displayed in descending order and each difficulty level will have a separate leader board. The date when the score was obtained would also be stored in the database. Only the top 5 scores would be displayed, and these scores would be arranged in descending order with each player’s name clearly next to their corresponding score.
* Difficulty levels: easy, medium, and hard. For higher difficulties, enemies will have faster fire rate, being harder to dodge and more enemies will be present in each level. The player will also have more starting lives if they play on an easier difficulty.
* The limited number of lives the player has which will be correctly displayed at the top (or bottom) of the UI. The number of lives will depend on the difficulty.
* Some enemies also have multiple lives like the player, but they don’t receive invincibility frames like the player does.
* Invincibility frames: after a player takes damage and loses a life, they will be invulnerable and unable to lose more lives for a short period of time (≈ 0.5s). This feature will be accompanied by a flashing animation on the player’s ship. This is implemented because it prevents the player from losing a life for every frame they are in collision with a dangerous object. In other words, it prevents the user from losing multiple lives in very quick succession.
* The game has a ‘colourblind mode’ which can be selected from the settings menu. This will make the colours of the game more accessible and readable for people who suffer from colour-blindness.

## 1.8 The requirements of the solution

* Firstly, the user would need to log in with the log-in screen displayed when they launch the program. This would be done by entering their username and password into the boxes and hitting enter. This is done so their username can be saved to the high scores database (along with the score they get)
* The user can navigate the main menu with WASD or arrow keys and select an option by hitting the enter key or the spacebar.
* When in game, the player can move their ship normally with WASD, enable slower movement or ‘focus’ by holding shift while using WASD to move. Spacebar is used to shoot and can be hold down to shoot repeatedly. There is a maximum fire rate.

**Input requirements**

* The player is able to navigate the main menu and select the desired option: play, settings, versus, high scores or log off.
* When backspace is pressed, it takes them to the previous menu
* The player is able to move the ship using WASD on the keyboard. The ship cannot move outside the boundaries of the screen
* The player can shoot by pressing the spacebar. The key can be held down to shoot continuously.
* A second player is able to move a second ship with the arrow keys in versus mode

**Process requirements**

* The system must record the player’s score whenever they successfully hit an object or collect a score pickup.
* The system must detect whenever the player hits a dangerous object such as an alien and make the player take damage
* The player’s lives must be tracked. It will be decremented by one when the player is hit and incremented when the player picks up a health pickup.
* If the player’s lives reach 0 it takes them to the game over screen.
* The game has a timer that must be decremented every frame of gameplay. The timer is used for many important things such as when enemies appear.

**Output requirements**

* The main menu buttons must be displayed only on the main menu and have an animation for when the button that the player is currently selecting.
* The player’s lives must be displayed during gameplay. The number of hearts corresponds to the number of lives the player has left.
* The game has a timer that must be decremented every frame of gameplay. The timer is used for many important things such as when enemies appear and when the player wins.
* When the player selects the ‘highscores’ button from the main menu, the top five scores will be displayed in order on screen.
* In two-player mode, whichever player wins should display a different win screen.

**Storage requirements**

* The player’s settings are saved to a text file so that if they exit and return the settings they selected will not change.
* Usernames and passwords of users are saved to a table in a .db file
* Usernames and passwords of admins are saved to a separate table
* High scores are saved to another table, storing the player’s username, the score they got and the date the score was achieved.

## 1.9 Success Criteria

|  |  |  |
| --- | --- | --- |
| **Requirement** | **What this success means** | **Evidence** |
| Login screen before the game is played | A login window where the user can enter their username and password. If they are correct, it will login the user to the game. The username and password will be stored in a database that is local to the system. The password will be hashed. | Video of the login window with a correct username and password being entered. |
| User can customise the login window | The user should be able to change the colour of the login window and this preferred option should be saved. | Video of login window being customised. |
| Intuitive UI | The user can utilise the window without needing prior knowledge or guidance. Buttons should be clearly labelled. | Questioning and obtaining feedback from stakeholders after they have used the UI |
| Simple design | Buttons and text should not be too small, the colour scheme should also be appropriate. | Screenshots of the login window and game menus |
| Main menu for the game | A number of options that can be selected with the keyboard alone.  It should be clear which option the user is selecting. | Screenshot of the main menu of the game |
| Settings screen for the game | A settings menu where the user can change the window resolution, difficulty, audio volume and colourblind mode.  The currently active settings should be easily visible. | Screenshot of the setting menu of the game |
| Window size is changeable | When the window width and height is changed from the settings menu, the game will restart. Sprites’ size and position should scale based on window width and height. | Video of window size being changed |
| Number of lives displayed during gameplay | The HUD (heads up display) should contain a number of hearts that indicate the number of lives the player has left. In 2-player mode, each player’s number of lives must be displayed separately. | Screenshots of single player mode and 2-player mode showing different numbers of lives. |
| High-scores leader board | The top 5 scores achieved must be shown in the high-scores screen next to the player who got each score. The text must be clear and readable. | Screenshot of high-scores screen |
| Controllable ships by the players | In 2-player mode, one ship should be controlled with the WASD keys and the other controlled with the arrow keys. | Video of player movement |
| The game should have minimal bugs or glitches. | The game should function completely as intended. Any unintentional effects that completely change the game should be patched. | Evidence of bugs being patched in the logs |
| The program must not crash but instead show an error message | Tkinter messagebox is used to display a warning message at the centre of the screen with an appropriate message for the error. | Screenshot of error messages |
| Game should run smoothly at a constant 60fps | The game caps frame-rate at 60 frames per second. The game is simple and 2D so it should not have performance issues even on lower end hardware. | Video of game running with external software measuring fps |

## 1.10 Limitations of the solution

* The main limitation with the solution is that it only works on one platform – PC. It is not possible to run the solution on a mobile for or another device. This is because, after development is finished, the game will be compiled as an executable file that can only be run on the specific OS it was compiled for. The game will not be ported to other devices because this not be an effective use of development time considering I intend for the game to be played by my stakeholders on PC anyway.
* One limitation of the solution is that there is no support for controllers, or joystick input in the game. This is acceptable because the solution still fully functions with mouse and keyboard.
* Another limitation is that the max frames per second of the game is capped at 60 fps. This is because 60 frames per second appears smooth enough to the human eye. In addition, almost all modern hardware would be able to run my simple 2D game at a stable 60 fps.
* The graphics for the game are limited as objects are all simple 2-dimensional images rather than 3-dimensional. However, this has the advantage of making the hardware requirements to run the game lower, making it more accessible. This is because not everyone can afford a good graphics card which is often required to run most modern 3D games. This therefore could lead to more people being able to experience and have fun playing the game. Making the game in two dimensions also allows for me to complete the project in the allocated time frame.
* Concerning how player’s high-scores in the game will be saved, a local database will be used. This is a limitation of the game because you will only be able to save and view scores that were made on a specific machine rather than being able to connect to an online cloud storage where all scores made across any device is saved. However, this is intentional because it will allow the user to track their own scores on their PC.

## 1.11 Hardware and software requirements

Hardware requirements:

* 1.5GHz or faster processor – this fast of a processor is needed because the game has many animations and moving objects. Many calculations are needed to be done within a short time for many aspects of the game such as collision detection and player movement. This processor speed is also above the minimum needed to run Windows 10 OS.
* Minimum 2GB RAM – this is needed to run windows 10 in addition to the RAM needed for the game. The game will use the system’s RAM in order to improve performance.
* 20GB free hard drive space – sufficient to store the OS and the game. The game will use secondary storage to store images for sprites (many images are needed for animations), in-game music and the database.
* Working keyboard and mouse – this is needed for the input to the login system and for the user to be able to interact with the game
* Working monitor – to display the game’s GUI to the user

Software requirements:

* Operating System: 64-bit Windows 10 version 21H2 or later – this is needed in order to run a PC and manage the files of the game.

Chapter Two: Design

## 2.1 Introduction

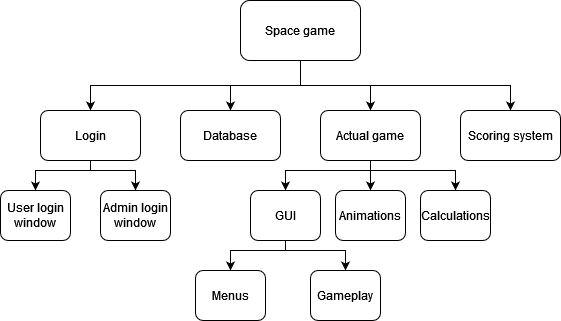
The design objectives for my game will be based upon the interviews conducted on the stakeholders and are similar to the requirements already specified. I will create a list of design requirements that will be implemented into the game. For the general design of the game, all of the stakeholders were fine with it being a 2D shooter where the player controls a spaceship and has the goal of shooting asteroids and other enemies. Most of my stakeholders liked the space theme. The game will be designed in **Python,** using the **pygame** module for the game and tkinter for the login window. I have decided to use python and pygame because it is suitable for my current level of programming skills and also contains a large number of libraries and modules that I can utilise in my project. Pygame in particular is quite often used as a learning tool to understand the basics of game development before moving up to the next level and using more advanced software.  
To demonstrate the interfaces to the users, I will design them digitally and show it to them. I will then collect feedback from this and implement the feedback if possible.



## 2.2 Decomposition of the problem

I have decided to split my project into four main sections, the login window, the login database, the main game screen, and the scoring system. I will later expand upon this main decomposition diagram by going into more depth with each of these sections and give them each a more detailed diagram.   
I have broken the problem down into a top-down design because it will allow me to develop each of the sections as separate modules or functions. This will make testing easier later on because each module can be tested separately, and then multiple modules can be tested together at once when testing the whole system.

### 2.2.1 Decomposition Diagram



All of these sections of the decomposition diagram can be developed separately but will link with each other in the complete project:

* The login section will encompass the entire of the login user interface, including the user and admin windows. It links with the database as user details will be stored there. When a user attempts to log in with the user login window, the data they entered must be compared to those in the database. If successful, this will take the user to the main game.
* For the main game section of my project, parts needed for the actual video game will be included. This will include a main menu, a settings menu, single player as well as multiplayer gameplay, and a highscores menu to view scores. This part will be done in pygame and will need GUI, animations and calculations in order to work.
* The scoring system in the game will be used to keep track of and display the player’s score in the game as well as save this score to the database once the game is completed.
* The database will store user details as well as highscores in a separate table. The database will be accessed by the login system in order to search for or create a new user and the scoring system will use the database to save and retrieve scores set by users.

### 2.2.2 Data Flow Diagram

#### Login data flow diagram:



#### Game data flow diagram:



### 2.2.3 Input Process Output

Login:

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Username entered once in a text box | If logging in, checks the username and corresponding password exist in the database by searching the users table linearly.  If creating a user, checks the username doesn’t already exist in the database and the username is valid (is between 3 and 20 characters in length and only contains alphanumeric characters and underscores). | An error message stating “invalid details” will be shown to the user if the details aren’t correct. The |
| Password entered in text box | When creating a new user, checks the password is valid (8 or more characters long and is a string). When logging in, checks the password matches user’s password. | Error or success message |
| Repeat password entered in text box | When creating a new user, checks if the repeat password is equal to the password. If they aren’t a new user will not be created. | “Passwords do not match” error message if the passwords don’t match. |

Actual game:

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| A choice from Main menu navigated and selected using keyboard. | When the user selects: If the current menu option is ‘play’ take the user to the main game function. If the current option is ‘settings’, take to the settings game state. If the current option is ‘highscores’ take the user to the viewing highscores state. | Change of current menu option selected (indicated by arrows). Screen changed when an option is selected. |
| Movement using player keyboard input | During gameplay, check if the player is not at the edge of the screen before moving in the corresponding direction. Update the ship’s position. | Draws the ship moving on screen. |
| Shooting | Check that the player has not already fired recently. This cooldown will depend on the difficulty. Create a new laser instance if shooting is successful. | Draws lasers fired by the player, displayed in window to player. |
| Settings | Update settings text file to the new values specified. | Current settings shown as highlighted in settings menu. |

## 2.3 How All Solution Parts are Linked

### 2.3.1 State Diagram of the different forms/parts

#### For the login system:



**Start**

**Takes to game**

This state diagram shows how a user would navigate the login window. The end part of this diagram would take a user to the game part of the project.

#### For the game:



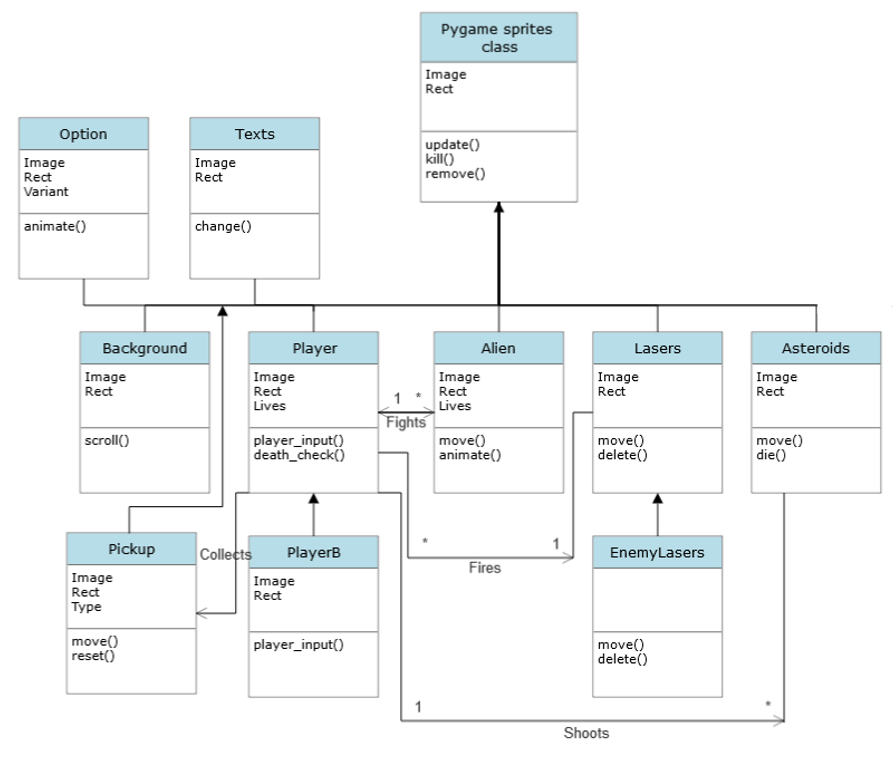
**Returns to login**

**Start**

This state diagram shows how a player would navigate the game section of the project. The end part of this diagram (followed by log off button) would take the user back to the login window section.

### 2.3.2 Classes

#### Game sprites UML class diagram:



I have decided to use classes in my project for a number of reasons. I found it to be extremely useful in the game section especially because there is often a large number of objects that need to be created and displayed on the screen at the same time. This is because a single class can be used as a template to create a large number of objects efficiently without needing to re

For example, with the lasers class, each individual laser will behave the same but there could be a large number created within quick succession.   
  
Using classes also allowed for inheritance. In the game, inheritance is mainly used to have the update(), kill() and remove() functions from the pygame.sprite.Sprite class inherit to each of the subclasses. This allows the code to be more compact because rather than writing an identical update() method for every sprite, they each inherit this from the pygame sprite class.

Classes can also interact with each other. For example, the player can shoot many lasers by creating many objects of that class. The lasers can than cause an alien object or asteroid object to take damage or die.

## 2.4 Database Design

I will use a database file to store users' usernames and passwords. I will allow an admin account to have the ability to create new users. Existing users should be able to log in by verifying whether their username and password match those in the database. I have decided to hash users’ passwords in order to improve the security of the system.

### 2.4.1 Normalisation

* The database is in first normal form because the data in each record is atomic so cannot be divided down further into multiple fields. Each record is also unique because of the use of the primary key fields in each table (username for the Users table and ID for the highscores table).
* The database is in second normal form because it is already in first normal form and there are no partial dependencies between part of composite key field and another field since there is no composite key field in any of the tables.
* It is also in third normal form because the database is already in second normal form. Additionally, there are no dependencies between non-primary key fields. This is because Name, Score and Date are all independent of each other

### 2.4.2 Data Dictionary

Users table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Data type | Length | Validation | Comment | Example |
| Username | Varchar | 255 | Primary key field, not null | Username must be unique for each user | Louis\_125 |
| Password | Binary |  | Not null | Hashed password stored | 01001001, … |

Highscores table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Data type** | **Length** | **Validation** | **Comment** | **Example** |
| ID | Int |  | Primary key field, autoincrement | Autoincremented to avoid the same ID appearing twice. | 5 |
| Name | Varchar | 255 | Not null | Foreign key field from the Users table. | Louis\_125 |
| Score | Int |  | Not null, not < 0, not > 9999999 | The player’s score stored as integer. It can’t be negative or too high. | 6500 |
| Date | Text | 16 | Not null | Date the score was achieved | 01/10/2022 |

### 2.4.3 Entity Relationship Diagram

∞

1

Highscore

User

The relationship between user and scores is one-to-many. This is because one user can set many different high scores but an individual score can only belong to one user. Therefore, one record in the Users table can match to many records in the Highscores table, with Username being a foreign key field in the Highscores table but a primary key field in the Users table.

### 2.4.4 SQL Pseudocode

// Creating the Users table that will store user details

CREATE TABLE IF NOT EXISTS Users(  
 Username VARCHAR(255) PRIMARY KEY NOT NULL,  
 Password TEXT NOT NULL,

// Creating Highscores table that will be used to store scores set in-game

CREATE TABLE IF NOT EXISTS Highscores(  
 ID INTEGER PRIMARY KEY AUTOINCREMENT,  
 Name VARCHAR(255) NOT NULL,  
 Score INT NOT NULL,  
 Date TEXT(16) NOT NULL,  
 FOREIGN KEY (Name) REFERENCES Users(Username));

## 2.5 Design of Main Parts of the Solution

### 2.5.1 Part ONE - Login:

I will use tkinter for the login system windows.

#### 2.5.1.1 Form Design and Layout

##### Login window



This checkbox will call a procedure that hides/shows the text in the password box.

Calls the cancel() procedure when clicked. This displays a confirmation message if the user wants to quit.

Enter button that calls log\_in() function when clicked.

#### 2.5.1.2 Validation rules

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Item** | **Data Type** | **Validation Rule** | **Justification** |
| Username | String | Must be between 3 and 20 characters long and contain only alphanumeric characters and underscores. | The length validation ensures that the username will not be too long or left blank making it suitable for the database. |
| Password | String | Must be greater 8 or more characters long. | This ensures the password cannot be left blank or be too short. |

#### 2.5.1.3 Algorithms and Pseudocode for Login

Function search(username, password, table):

// First must get data from correct table

IF table == “Admins”:

Records = execute SQL: SELECT \* FROM Admins

Else:

Records = execute SQL: SELECT \* FROM Users

ENDIF

FOR each row in records:

If row[0] == username and row[1] = password:  
 // Matching username AND password means the user is found

Return True

Next row

ENDFOR

// Entire table linearly searched without a match – not found

Return False

End function

Procedure log\_in(username, password):

IF search (username, password, “Users”):

// Here will be a function that closes login window and starts the game

Play game

ELIF search (username, password, “Admins”):

// Function to close login window and open admin window

Open admin window

ELSE:

// Error message displayed as popup box

Show error message

ENDIF

End procedure

##### Cancel button:



##### Login window class:



The attributes for this class will be public so they can be more easily viewed or changed by a function outside of the LoginWindow class.

* Hidden refers to whether the text in the password box is displayed as \*\*\* or not. This is true by default and toggles whenever the user clicks the show password checkbox.

#### 2.5.1.4 Test Plan for PART ONE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Type of data** | **Expected outcome** | **Justification** |
| 1.1 | Attempt to log in with correct user details | Username = “test\_user1”  Password = “testpassword01” | Normal | Successful log in message and logs in the user | The user should log in when correct details are entered |
| 1.2 | Attempt to log in with correct username but wrong password | Username = “test\_user1”  Password = “incorrectpass” | Invalid | Error message displayed | A password that doesn’t match the database should not log in the user. |
| 1.3 | Attempt to log in with wrong username but correct password | Username = “incorrectuser”  Password = “testpassword01” | Invalid | Error message displayed | A username not in the database should not log in the user. |
| 1.4 | Attempt to log in with both boxes empty | Username = “”  Password = “” | Erroneous | Error message displayed | To log in, username and password cannot be empty |
|  |  |  |  |  |  |

### 2.5.2 Part TWO – Admin Panel:

#### 2.5.2.1 Form Design and Layout



Passwords are checked to be equal when create\_user is called.

Calls the delete\_user function when clicked.

Button that calls the create\_user function when clicked.

#### 2.5.2.2 Validation rules

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Item** | **Data Type** | **Validation Rule** | **Justification** |
| Username | String | Between 3 and 20 characters long and contain only alphanumeric characters and underscores. The username must also not already exist in the table. | By checking the username doesn’t already exist in the table it ensured that each username will be unique. |
| Password | String | Must be greater 8 or more characters long. | This ensures the password cannot be left blank or be too short. |
| Re-entered password | String | Must be equal to the password before a new user is created. | This reduces human error when creating a new user by ensuring there is no mistakes when typing the password. |

#### 2.5.2.3 Pseudocode for Admin Panel

// This function will enter a username and corresponding password into the database if they are valid

Function create\_user(username, password, confirmed\_password):

If password == confirmed\_password:

If username is valid and password is valid:

execute SQL: insert into Users (Username, Password)

return “Entered user”

end if

Else:

return “Error – Passwords do not match”

end if

end function

// This function will delete a record from the database with the given username

Function delete\_user(name):

IF is\_existent\_user(name):

Try:  
 execute SQL: DELETE FROM Users WHERE Username=name

Return (“Successfully deleted”)

Catch:

Return (“Error deleting user”)

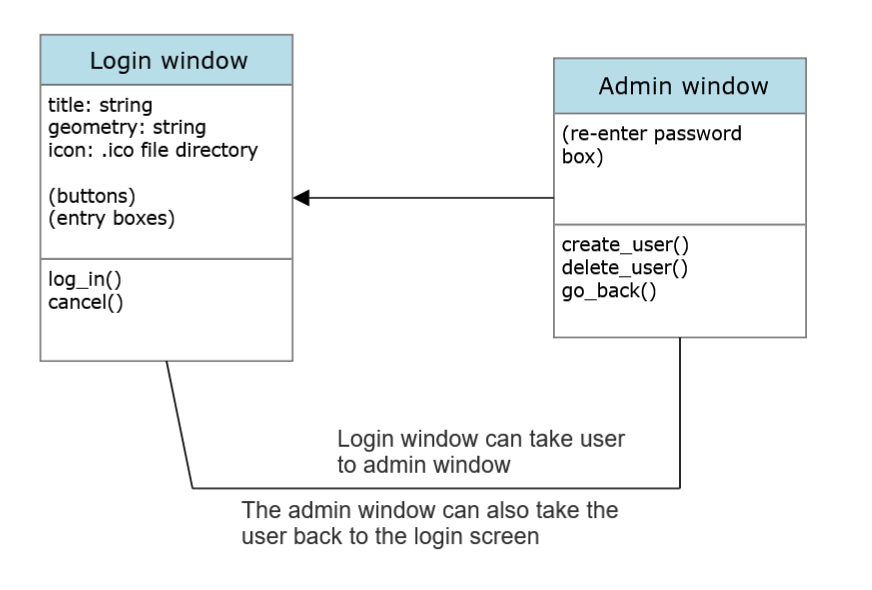
ELSE:

Return (“Error – user does not exist”)

ENDIF

End function

#### 2.5.2.4 Login and Admin UML class diagram



The Admin Window class will inherit from the previously discussed Login Window Class. If the user enters the correct details that correspond to an amin account, they will be taken to the admin window.

#### 2.5.2.5 Test Plan for PART TWO

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Type of data** | **Expected outcome** | **Justification** |
| 2.1 | Attempt to create a user with valid username and password. The re-entered password matches the password. | Username = “Entered\_user1”  Password = “testpassword01”  Confirmpass = “testpassword01” | Normal | User is successfully added to the Users table, message is displayed. | User information should be saved to the database when valid data is entered. |
| 2.2 | Attempt to create a user with valid username and password but re-entered password does not match. | Username = “Entered\_user2”  Password = “testpassword01”  Confirmpass = “notmatching” | Invalid | “Passwords do not match” error message. | Confirmation password must be the same as the original password. |
| 2.2b |  | Username = “Entered\_user2”  Password = “testpassword01”  Confirmpass = “Testpassword01” | Boundary (invalid) | “Passwords do not match” error message. | The passwords must be exactly the same (case sensitive) |
| 2.2c | Attempt to create a user with a valid username and password but the re-enter password is left empty | Username = “Entered\_user2” Password = “testpassword01” Confirmpass left blank | Invalid | “Passwords do not match” error message. | Re-entered password must be the same as the original password. |
| 2.3 | Attempt to create a user with all fields left blank. | Entry boxes are all empty. | Invalid | Error message | Username and password cannot be null. |
| 2.4 | Trying to create user with password left blank. | Username = “Entered\_user4” Password boxes left blank. | Invalid | “Invalid password” error message | A blank password is invalid – falls below character limit. |
| 2.5 | Trying to create user with username left blank. | Username entry box left blank. Password = “testpassword05”  Confirm | Invalid | “Invalid username” error message. | A blank username is invalid – cannot be null. |
| 2.6 | Attempt to create user that already exists in the table. | Username = “Entered\_user1” (Existent user)  Password = “testpassword06” (unique password)  Confirmpass = “testpassword06” | Invalid | “User already exists” error message | Username must be unique because it is the primary key. |
| 2.7 | Attempt to create an existent username with matching password. | Username = “Entered\_user1”  (Existent user)  Password = “testpassword01” (Correct password)  Confirmpass = “testpassword01” | Invalid | “User already exists” error message | The user already exists in the table. |
| 2.8 | Creating with a unique username but existent password. | Username = “Entered\_user8”  Password = “testpassword01”  (existent password)  Confirmpass = “testpassword01” | Valid | User is entered successfully. | The password does not have to be unique because it is not the primary key. |
|  |  |  |  |  |  |

### 2.5.3 Part THREE – Game menus:

#### 2.5.3.1 Game Menus Design and Layout

##### Game main menu

This menu is navigated with keyboard alone rather than mouse and keyboard as in the previous menus. Therefore, it is important to indicate which menu option is currently selected (indicated here by the bold line). A button is pressed when the user selects it and presses a confirmation key.



Starts the main game (single player)

Starts the 2-player game

Opens the settings menu

Opens the highscores menu

Currently selected option will have different animation

##### Game settings menu



Arrow indicates currently applied option

When any setting is selected the save\_setting() function is called. This updates the text file that stores the game’s settings.

When the resolution is changed the restart() function is called as the window must be recreated for resolution change to take effect.

##### Highscores Menu



get\_scores() function is used to read the scores from the table in descending order.

If there is less than 5 scores in the table then black spaces will be shown.

#### 2.5.3.2 Justification of Validation rules

Menu validation:

Validation is not necessary for the user’s inputs on the menus because there are only specific predetermined options that can be selected. The user does not enter text like they do in the login so this kind of validation is not needed for the game’s user input.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Item** | **Data Type** | **Validation rule** | **Justification** |
| Score | Integer | Must be positive integer and less than maximum value of 1,000,000 | The score cannot be below 0. If the score was a decimal or too large, the formatting of the high score screen would be ruined. |
| Name | String | Must already exist in the user table or “Guest” | The user must be registered in the database or if the player is playing without logging in, results will be saved under the “Guest” name. |

#### 2.5.3.3 Algorithms and Pseudocode for Menus

// Returns the scores as a list in descending order of score

Procedure get\_scores():

Set scores to empty list

execute SQL: SELECT \* FROM Highscores ORDER BY Score DESC

FOR each row in records:

Append to scores row[2]

ENDFOR

RETURN scores

End Procedure

// Returns the names as a list in descending order of score

Procedure get\_names():

Set scores to empty list

execute SQL: SELECT \* FROM Highscores ORDER BY Score DESC

FOR each row in records:

Append to scores row[1]

ENDFOR

RETURN scores

End Procedure

// Procedure used for initially creating the highscores table

Procedure create\_h\_table():

'''CREATE TABLE IF NOT EXISTS Highscores

(ID INTEGER PRIMARY KEY AUTOINCREMENT,

Name VARCHAR NOT NULL,

Score INT NOT NULL,

Date TEXT(16) NOT NULL,

FOREIGN KEY (Name) REFERENCES Users(Username));''')

//Main menu option class. This class will be used to generate all of the selectable ‘buttons’ on the main menu of the game.

Class Option(Sprite):

public cycle: Int  
 public timer: Int  
 public toggle: Bool  
 public type: Str  
 public image\_sprites: List // List of images used for animation

public procedure new(variant):  
 cycle = 0  
 timer = 0  
 toggle = True

if variant = “play”:  
 type = “play”  
 set image\_sprites to list of images for play

elif variant = “settings”:  
 type = “settings”  
 set image\_sprites to images for settings

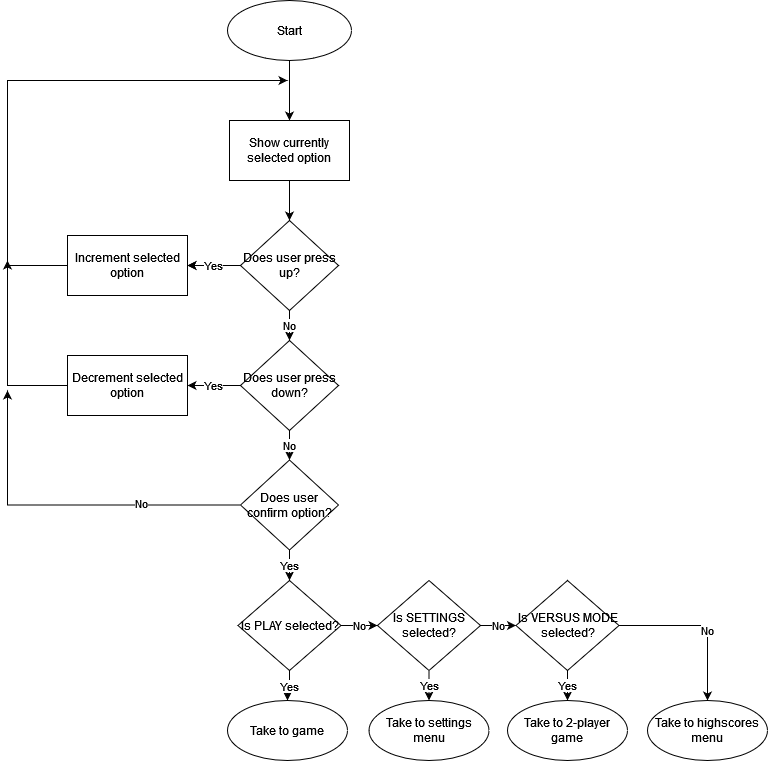
elif variant = “versus”:  
 type = “versus”  
 set image\_sprites to images for versus

elif variant = “highscores”:  
 type = “highscores”  
 set image\_sprites to images for highscores

END Procedure

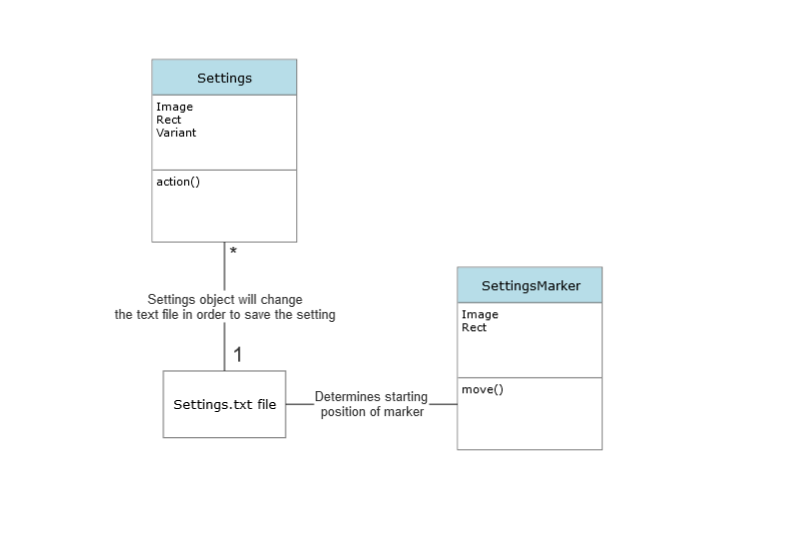
END Class

##### Main menu flowchart



#### 2.5.3.4 Key Variables/Classes

##### Settings Menu



In the settings menu, I will use two classes. The Settings class will be used to generate the options that the user can select in this menu and the SettingsMarker class will be used to generate arrows that show the option that is currently being applied.  
The Settings objects will have an action() method that will save a specific setting depending on the variant of the object. For example, if the variant of the object is difficulty and the user is currently selecting “easy” then this function will alter the settings text file to set the difficulty to “easy”.  
When the user initially opens the settings menu, the SettingsMarker objects will read from the settings text file and use the move function to go to the position of the currently selected settings using the move() method. These markers could then be moved if the user chooses to change a setting.

#### 2.5.3.5 Test Plan for PART THREE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Type of data** | **Expected outcome** | **Justification** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 2.5.4 Part FOUR – Design of Main Game:

The following section is the design of the main game section of my project. I plan to spend more time developing this section than other sections because is ultimately the purpose of my project – to be a game that users can play. The user will be taken to this window when they select “Play” on the previously designed main menu section on the game.

#### 2.5.4.1 Window Design and Layout

Counter that displays the player’s lives

Score counter, this will display the player’s score.



Player ship. Movement controlled with WASD.

Enemies that appear from the right of the screen.

The default aspect ratio of the window will be 16:9. However, the user will be able to change the size in settings.

Projectiles fired by the player with SPACE.

In this UI, the score is displayed at the top so that the user can keep track of their current score, representing how well they are doing in the game. This also allows the user to quickly find out which actions reward the player with more points.  
The lives counter is similarly positioned at the top to make it easier for the player to keep track of how many lives they have left. This would make the game more enjoyable for the player.

#### 2.5.4.2 Algorithms for Main Game

##### Main flowchart



This flowchart represents the main game loop of my project, where the player fights enemies while having a number of lives. The game will end if the player reaches 0 lives or the timer reaches 0. The values that the lives or timer start at could be adjusted.

##### Player movement flowchart

(This algorithm could be adapted to use different keys other than WASD, this is just an example)



The algorithm for moving the player must be this large for a number of reasons. If the player is holding opposing movement keys (such as W and S at the same time) then I must ensure that the program cancels this out and doesn’t move the player at all rather than taking the input of the first key press that is checked.  
In addition, if the player is inputting a diagonal movement (such as W and D) then this must be treated differently than simply adding one pixel movement onto the other because this would result in the player being able to move faster in diagonal directions.

To illustrate this point:

If the amount the player can move in one frame is distance **a**.  
Because the directions form a right-angled triangle with diagonals on the hypotenuse, the distance during a diagonal movement if the vertical movement is just added to the horizontal movement would be √2a2

This is due to Pythagoras’ Theorem and is shown below with the player being represented by a circle and movement of length **a** being represented by blue arrows.

Shape

Description automatically generated

The players movement should look more like this diagram, so that the player moves at a constant speed (pixels per frame) no matter which direction they are moving in.



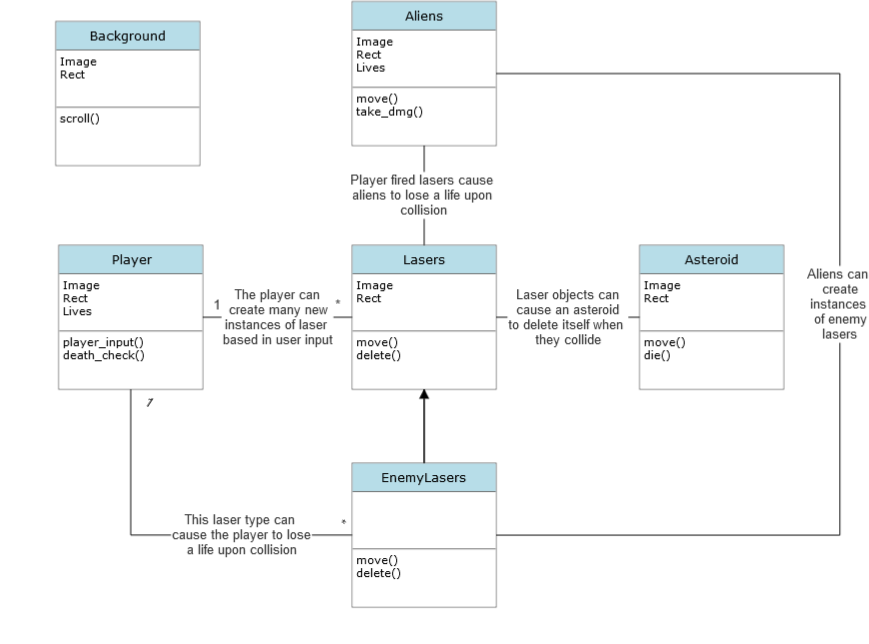
This would be achieved by considering the traingle that forms a hypotenuse of length **a**. If the distance that must be added to horizontal (and vertical) position is **x**, then:

A picture containing baseball, player

Description automatically generated

Therefore, this calculation will be used whenever the player moves diagonally.

#### 2.5.4.3 Key Classes for Main Game



This is the UML class diagram for the main single player section of the game. When the player inputs the key to shoot, this will create an additional instance of the Lasers object at the player’s ship’s position. The player lasers will move based on the move() method and will move from the left to right of the screen unless they collide with an object such as an Alien or Asteroid or reach the end of the screen.   
When a laser collides with an Asteroid, both objects will be destroyed instantly. However, when a laser collides with an Alien object, the lives of that Alien will be reduced by 1. When the lives of an Alien reach 0, it will be destroyed (and the user will be awarded score).  
This functions similar to how the EnemyLasers objects (which inherit attributes of the Lasers class) interact with the Player. Here, polymorphism is used so that the EnemyLasers have different move() and delete() methods that the parent class. EnemyLasers objects will move from the right of the screen to the left and will be deleted if they collide with the player or the end of the screen. When this laser type collides with the Player, the Player attribute Lives will decrease by one. If the lives of the player reaches 0, then the game will progress to the game over state and the user will see the game over screen.

#### 2.5.4.4 Test Plan for Main Game

Will do dev. and testing before this section

### 2.5.5 Part FIVE – Error Handling

The presence of message boxes and error handling within my project is important. This is because it allows me to display messages to the user that doesn’t rely on using the print statement in python. It also prevents the project rather than crashing, to display an error message as long as exceptions are utilised effectively in the code. These messages are useful in the login system, for example showing an ‘invalid login’ error message when the user enters an incorrect username and password.  
The question message box can be used to do a different function depending on whether the user selects the ‘yes’ or ‘no’ button.

#### 2.5.5.1 Form Design and Layout

I will use Tkinter to display messages.

Diagram

Description automatically generatedError message box:

Closes the message box window and returns true

Question message box:

Graphical user interface, diagram, application

Description automatically generated

Used to return False when the user clicks the button. The message window also closes.

Used to return true when the user clicks this button. The message window also closes.

#### 2.5.5.2 Pseudocode for Error handling

##### Messages module pseudocode:

Procedure show\_message(title, message, opt): // Option 1: info, 2: error, 3: warning, 4: question box  
 if opt == 1:

show info box (title, message)

elif opt == 2:

show error message (title, message)

elif opt == 3:

show warning message (title, message)

else:

return ask question message (title, message) // Returns true for yes, false for no

End if

End Procedure

This procedure includes the option parameter to make it more versatile and usable for displaying different kinds of messages. This is done in one function rather than multiple functions to make the code more maintainable and less complex.

##### Validation module pseudocode:

// This function will be used anytime length validation must be performed. Option 1 checks the length is equal, opt 2 checks >=, and opt 3 checks <=.

Function is\_length(data, length, opt)

TRY

IF opt == 1 THEN

IF length of data == length THEN

RETURN True

ELSE

RETURN False

ELSE IF opt == 2 THEN

IF length of data >= length THEN

RETURN True

ELSE

RETURN False

ELSE IF opt == 3 THEN

IF length of data <= length THEN

RETURN True

ELSE

RETURN False

END IF

EXCEPT Exception AS ex:

RETURN ex

END TRY

END FUNCTION

// This function takes three inputs: data, lo, and hi and checks whether the length of the input data is within the range specified by lo and hi (inclusive)

Function is\_inrange(data, lo, hi)

TRY

IF length of data >= lo AND length of data <= hi THEN

RETURN True

ELSE

RETURN False

ENDIF

EXCEPT ValueError

RETURN "Error 2"

ENDTRY

ENDFUNCTION

// This function takes the data to be validated, u, and the option for validation and returns a Boolean value. For username validation, the data must be a string from length 3 to 20 characters. Each character must also be a valid character for true to be returned. For password validation, the data must be a string and of length 8 to 255 characters for true to be returned.

Function is\_valid\_user(u, opt):

IF opt == "username" THEN

IF u is a string THEN

IF is\_inrange(u, 3, 20) THEN

SET u = u in uppercase

FOR n in range(length of u) DO

IF u[n] is not valid character THEN

SET valid = False

ENDIF

ENDFOR

ELSE

SET valid = False

ENDIF

ELSE

SET valid = False

IF u is a string AND is\_inrange(u, 8, 255) THEN

SET valid = True

ENDIF

ENDIF

RETURN valid

ENDFUNCTION

#### 2.5.5.3 Test Plan for Error handling

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Type of data** | **Expected outcome** | **Justification** |
| .1 | Attempt to display an info box (not error) message | Run show\_message() with title blank and message = “Test message”  opt = 1 | Normal | Info box with no title containing “Test message” displayed | Initial test to determine if the message can be displayed properly |
| .2 | OK button clicked on info box | Clicking on ‘OK’ | Normal | Message box window closes | The message box should close when the user clicks ‘OK’ |
| .3 | Attempt to display an error message | Run show\_message() with title blank and message = “Error message”  opt = 2 | Normal | Error message with no title containing “Error message” displayed | Initial test to determine if an error message can be displayed correctly |
| .4 | OK button clicked on error message box | Clicking on ‘OK’ | Normal | Message box window closes | The error message should close when the user clicks ‘OK’ |
| .5 | Attempt to display a question message box | Run show\_message() with title blank and message = “Question message”  opt = 4 | Normal | Question message with no title containing “Question message” displayed | Initial test to determine if a question message can be displayed correctly |
| .6 | ‘Yes’ button clicked on question message | Clicking on ‘Yes’ | Normal | Window closes and function returns true | A question box will be used whenever there will be different outcomes for clicking ‘yes’ and ‘no’. Therefore, Yes should return true and no should return false. |
| .7 | ‘No’ button clicked on question message | Clicking on ‘No’ | Normal | Window closes and function returns true |  |

## 2.6 Stakeholders involvement

In this section I will get feedback from my stakeholders on specific parts of the project, particularly the GUI in sections. Showing the backend parts of the project is now necessary, so I abstracted these details from the stakeholders.

### Part TWO

I interviewed my stakeholders in order to get feedback on the design for the login system. One of the stakeholders stated that it would be beneficial to include a back button on the admin window which takes them back to the user login screen. Another stakeholder recommended that I add a dropdown menu to the top of the admin window.

#### Revised Admin Window





Calls the reset\_scores() function when clicked

Goes back the login menu when clicked

Opens the dropdown menu when clicked

#### Pseudocode

Procedure reset\_scores():

execute SQL: DROP Table Highscores // The table is deleted and re-created in order to reset

create\_h\_table()

End Procedure

### Part FOUR

Additionally, when asking my stakeholders about the main game, they had suggested an improvement upon my design. One of these was to display the number of lives not as a counter, but rather as a health bar or a number of hearts. I can see why this would be more desirable because pretty much all modern games that feature a system where the player has a certain amount of “health” will use either bars or number of hearts to represent this. Using either one of these would make the user interface more intuitive and easier to read. Because of this, I have decided to change the lives counter to a certain number of hearts that will represent how many lives the player has left.

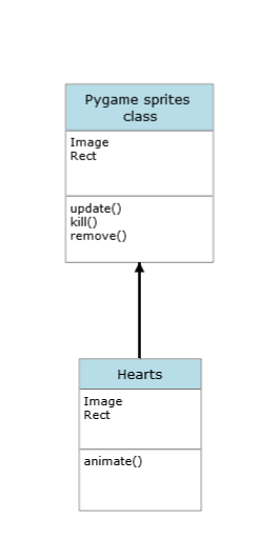
#### Revised Game UI



Number of lives displayed as heart counter in the top left. This will be updated based on the player’s current lives.

#### Key classes

The hearts as shown in the revised UI will be generated using a single class – Hearts.  
This, as usual will inherit and use methods from the pygame sprites class.

****

The animate method will be used to display an animation at the moment the player loses a life. This will be done to improve the quality of the game for the user and make it more visually appealing.

## 2.7 Testing plan to inform evaluation

I plan to carry out several levels of testing for the project. I first plan to test each of my functions and individually in order to check they each perform as intended using unit testing. I will then test each module of my project separately (e.g., testing only the login window). After this, I plan to test multiple modules together to see if there are bugs that arise when modules interact. I will fix any bugs before continuing on to test the next stages.  
This is done in order to eventually test that the whole system works as it is intended to.  
If a bug is found within a module of function level, I will attempt to fix the bug and then retest that module or function. If there is a bug found when testing the whole system, I will attempt to fix the bug and retest the system.

### Integration testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test Data** | **Expected Outcome** | **Justification** |
| .1 | Attempt to load the game | Run game.exe on Windows 10 OS | Game will load and run | The software should be compatible with windows 10. |
| .2 | Attempt to close the window by clicking the X in top right corner. | Clicking X | The game should close immediately | The software should be compatible with windows 10. |

### Scenario testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test Data** | **Expected Outcome** | **Justification** |
| .1 | Attempt an SQL injection | Inputting the string “User OR 1 =1” into both entry boxes | An invalid details error message should appear | The program should not log in with this test data. If it does, this means it is vulnerable to SQL injection attacks. |
| .2 | Attempt an SQL injection 2 | Inputting “DROP table Users” into username | An invalid details error message should appear | The table should not be deleted by this SQL injection attempt. |

Development and Testing

Chapter Three: Development and Testing

## 3.1 Introduction

When programming my project, I will use a modular approach. This means I will use the fact that I have decomposed the project into many smaller sections and I will first test each of these sections or modules separately. In the code, each module can import other modules, when necessary, in order to be able to run a function from another module. This approach allows for reusability of code and improves the readability and ease of editing the code.

I will use sqlite3 in order to build by database. This will mean the database is stored locally on one machine rather than on an online server. This makes it more suitable for a low number of users or even a single user. Because my project is single-player and is not an online game, it is not important that the database can only be accessed from one device.

I have decided to use Python 3.1, because it is the programming language that I feel the most comfortable with. There is also a wide range of libraries available, some of which will be used frequently in my game.

I will use Tkinter for the login part of the project as well as displaying popup messages. This allows me to create a user-friendly, visually appealing interface that the user will use to log in as well as displaying error messages. I am also comfortable with Tkinter as opposed to other libraries that can be used to generate UI.

For creating my game, I have decided to use Pygame. This will allow me to display the GUI for the space game as well as displaying the game’s menus such as the highscore menu. Pygame is suited for my project because it allows for creation of sprites that are flexible and can be displayed on screen when needed. It also allows the player to input and control their character in the game using the keyboard.

I will use the bcrypt library which will allow me to hash and salt the passwords of users that are stored in the database. This will improve the security of the system.

### Modules

|  |  |  |  |
| --- | --- | --- | --- |
| **Module name** | **Purpose** | **Relevant Design section** | **Development section** |
| Admincontrol.py | Provides the UI for an admin to create or delete users |  |  |
| colour\_changer.py | Changing the colour hue of an image |  |  |
| dates.py | Handles getting the dates needed for storing a score |  |  |
| game.py | Main game module, runs the game GUI |  |  |
| HighscoresData.py | Connects to the database in order to read or edit scores |  |  |
| login.py | Provides the UI for a user to log in |  |  |
| LoginData.py | Connects to the database in order to read or edit user details |  |  |
| main\_app.py (Top-level module) | Starts the initial login window |  |  |
| messages.py | Displays Tkinter popup messages for error handling |  |  |
| settings.py | Manages editing games settings that are saved to a text file |  |  |
| sprites.py | Contains all sprite classes that are used in game.py |  |  |
| testing.py | Runs automatic tests on each module |  |  |
| validation.py | Used anytime data must be validated |  |  |

## 3.2.1 Stage 1: Building the Error Handling

### 3.2.1.1 Building General Modules (Error Handling)

I have developed the Error handling section according to my design in 2.5.4 – Error Handling.  
I created a module called messages.py for displaying messages as well as a general validation module called validation.py. This contains all the code of this section and will be used for displaying error messages or validation.

#### Imported modules

1. **import** tkinter **as** tk
2. **from** tkinter **import** messagebox

I have used Tkinter Messagebox to display the windows in this section.

#### Version 1

Show\_message procedure:

1. **def** show\_message(title, message):
2. root = tk.Tk()
3. **messagebox.showinfo(title=title, message=message)**
4. root.destroy()

I first built this show\_message procedure, which displays a message window using Tkinter and creating a window called root. The Tkinter.messagebox module is then used to display an information box with the title as the value of the parameter ‘title’ and a message as the parameter ‘message’. When OK is clicked, the message box will be closed and the parent window will be destroyed with .destroy(). I will first test this version before continuing to develop this module.

##### Version 1 Testing:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** |
| 1.1 | Attempt to display a message | Run show\_message() procedure with title = “Test title” and message = “Test message” | Message box with title “Test title” and message “Test message” displayed |

Test 1.1 failed because, while the message was displayed correctly, the Tkinter parent window was still visible alongside the message box, see evidence below. This should not be the case.



#### Improvement

Show\_message procedure (fixed):

1. **def** show\_message(title, message, opt):
2. root = tk.Tk()
3. root.withdraw() *# Hides tk window immediately*
4. messagebox.showinfo(title=title, message=message)
5. **root.destroy()**

In order to fix the bug found in test 1.1, I have added code that withdraws the blank Tkinter window before the message box is displayed. This should prevent the window from appearing and now only the message box should be displayed.

##### Version 2 Testing:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** |
| 1.1 | Attempt to display a message | Run show\_message() procedure with title = “Test title” and message = “Test message” | Message box with title “Test title” and message “Test message” displayed |
| 1.2 | Ok button clicked in the message box | Clicking on ‘OK’ | Message box closes |

##### Version 2 Results:

Test 1.1 was successful as shown in this evidence:



Test 1.2 was successful as when the ‘OK’ button was clicked the message box window closed and the parent Tkinter window was never visible.

#### Version 3

Show\_message procedure (improved)

1. *# Options: 1-show info, 2-show error, 3-show warning, 4-ask y/n*
2. **def** show\_message(title, message, opt):
3. root = tk.Tk()
4. root.withdraw() *# Hides tk window immediately*
5. **if opt == 1:**
6. messagebox.showinfo(title=title, message=message)
7. **elif** opt == 2:
8. messagebox.showerror(title=title, message=message)
9. **elif** opt == 3:
10. **messagebox.showwarning(title=title, message=message)**
11. **else**:
12. **return** messagebox.askyesno(title=title, message=message)
13. root.destroy()

With the base show\_message() procedure now working as expected, I decided to add the feature to display different kinds of messages depending on option selected using the value of the ‘opt’ parameter.

##### Version 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 1.3 | Attempt to display a show info box. | Run show\_message() with title = “Info title” and message = “Info message”. opt = 1 | Info box displayed with title “Info title and message “Info message” | Success |
| 1.4 | Attempt to display an error box | Run show\_message() with title = “Error title” and message = “Error message”. opt = 2 | Error message displayed with title “Error title” and message “Error message” | Success |
| 1.5 | Attempt to display warning message box | Run show\_message() with title = “Warning title” and message = “Warning message”. opt = 3 | Warning message displayed with title “Warning title” and message “Warning message” | Success |
| 1.6 | Attempt to display question box | Run show\_message() with title = “Question title” and message = “Question message”. opt = 4 | Question box displayed with title “Question title” and message “Question message” | Success |
| 1.7 | Attempt to click “Yes” on question box | Clicking on the “Yes” button | Message box closes and returns True | Success |
| 1.8 | Attempt to click “No” on question box | Clicking on the “No” button | Message box closes and returns False | Success |

##### Version 3 Results:

Test 1.3:



Test 1.4:



Test 1.5:



Test 1.6:



Test 1.7:

Video 1.7

Test 1.8:

Video 1.8

### 3.2.1.2 Feedback from Stakeholder

I have showed version 3 of the message boxes to Kurk Milo. I went through each type of window (info, error, warning, question) as in testing. He agreed that the messages worked well and particularly liked the inclusion of different types of message boxes. He was satisfied with overall how the message boxes function and didn’t mention the addition of another feature.

After testing Version 3 and finding no bugs as well as getting positive feedback from my stakeholder, I can mark the development of the error handling section as complete and move onto developing the next module of my project, the database.

### 3.2.1.3 Building General Modules (Validation)

#### Imported modules

1. **from** datetime **import** \*

For the validation section of my project, the only imported module is datetime because this will be used for datetime validation.

#### Version 1

Is\_inrange() function:

1. *# range validation*
2. **def** is\_inrange(data, lo, hi):
3. **try**:
4. **if** (len(data) >= lo) **and** (len(data) <= hi):
5. **return True**
6. **return** False
7. **except** TypeError:
8. **return** "Error"

This function will be used to check if the length of the data is within range specified by the parameters ‘lo’ and ‘hi’. This will be used mainly to validate username and password length.

Is\_length function:

1. *# length validation*
2. *# parameters: data – string data that needs to be validated length(int) - length to compare data to*
4. **def** is\_length(data, length, opt):
5. **try:**
6. **if** opt == 1:
7. **if** len(data) == length:
8. **return** True
9. **return** False
10. **elif opt == 2:**
11. **if** len(data) >= length:
12. **return** True
13. **return** False
14. **elif** opt == 3:
15. **if len(data) <= length:**
16. **return** True
17. **return** False
18. **except** Exception **as** ex:
19. **return** ex

This function is used for validating length of the data against the value specified by the length parameter for the function. There are three different options for validation: exactly equal to, greater than or equal to, and less than or equal to.

##### Version 1 Testing:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** |
| 1.10 | Testing in\_range() with an integer value in the range specified by high and low | Data = “test” lo = 1 hi = 10 | True returned |
| 1.11 | Testing in\_range() with an integer outside the range | Data = “test” lo = 10 hi = 20 | False returned |
| 1.12 | Attempting to validate an integer value that lies on the boundary | Data = “1” lo = 1 hi = 10 | True returned because the range should be inclusive |
| 1.13 | Attempting to validate an incorrect data type (int) | Data = 1 lo = 1 hi = 5 | “Error” returned |
| 1.2.1 | Using is\_length() to check valid data is the correct length | Data = “test” length = 4 opt = 1 | True returned |
| 1.2.2 | Using is\_length() for valid data and >= | Data = “test” length = 1 opt = 2 | True returned |
| 1.2.3 | Using is\_length() for valid data and <= | Data = “test” length = 10 opt = 3 | True returned |
| 1.2.4 | Using is\_length() to check invalid data is the correct length | Data = “test” length = 5 opt = 1 | False returned |
| 1.2.5 | Using is\_length() for boundary invalid data and >= | Data = “test” length = 5 opt = 2 | False returned |
| 1.2.6 | Using is\_length() for invalid data and <= | Data = “test” length = 1 opt = 3 | False returned |
| 1.2.7 | Attempting to validate the wrong data type | Data = 1 length = 2 opt = 1 | Error message returned |

##### Automatic Testing

To run the tests shown in the previous table, I will use the unittest module in order to perform automatic testing. I will do this because it will allow for me to test this simple section of the project more efficiently and will be faster than using manual testing.

**Unittest (testing.py):**

**When performing any automatic tests when building this project I will use the class – TestMethods(). This class inherits from the unittest TestCase class, allowing for use of methods such as assertTrue() when testing. This is useful for comparing the value that functions (in this case is\_length() and is\_inrange()) return with the expected return value for the parameters given.  
For this test, the is\_inrange() function is tested first, followed by the is\_length() function.**

**TestMethods() class:**

1. **class** TestMethods(unittest.TestCase):
2. *# -----------------------------------------------------------------*
3. *# TESTING VALIDATION*
4. **def** test\_is\_inrange(self):
5. **data\_list = ["test", "1"]**
6. **for** data **in** data\_list:
7. self.assertTrue(is\_inrange(data, 1, 10))
8. self.assertFalse(is\_inrange("test", 10, 20))
9. self.assertEqual(is\_inrange(1, 1, 5), "Error")
11. **def** test\_is\_length(self):
12. length\_list = [4,1,10]
13. **for** opt **in** range(1,4):
14. self.assertTrue(is\_length("test", length\_list[opt-1], opt))
16. length\_list = [5, 5, 1]
17. **for** opt **in** range(1,4):
18. self.assertFalse(is\_length("test", length\_list[opt-1], opt))
20. **self.assertIsNot(is\_length(1, 2, 1), True)**
21. self.assertIsNot(is\_length(1, 2, 1), False)

##### Automatic Testing Results:



As, evident, all of the testing from the unittest passed, indicating that all of the tests 1.10 – 1.2.7 were successful and obtained the expected outcomes. Because of this, I will now develop the next functions of the validation section

#### Version 2

This function is used to validate whether the date given as the parameter is valid or not. It will be used later on in scoring section of the project, where each score in the database must have the date it was achieved stored in the same table.

Is\_valid\_date function:

1. *# Checks is a given date is valid*
2. **def** is\_valid\_date(given\_date): *# DD/MM/YYYY*
3. **try**:
4. datetime.strptime(given\_date, '%d/%m/%Y')
5. **return True**
6. **except** Exception as ex:
7. **return** False

Is\_valid\_user() function:

1. *# username and password validation*
2. *# u - data to be validated, opt - validation option: username or password*
3. **def** is\_valid\_user(u, opt):
4. **if** opt == "username":
5. **if isinstance(u, str):**
6. **if** is\_inrange(u, 3, 20):
7. u = u.upper()
8. characters = []
9. **for** i **in** range(65, 91):
10. **characters.append(chr(i))**
11. **for** i **in** range(48, 58):
12. characters.append(chr(i))
13. characters.append("\_")
14. valid = True
15. **for n in range(len(u)):**
16. **if** **not** u[n] **in** characters:
17. valid = False
18. **else**:
19. valid = False
20. **else:**
21. valid = False
22. **else**:
23. valid = False
24. **if** isinstance(u, str) **and** is\_inrange(u, 8, 255):
25. **valid = True**
26. **return** valid

##### Version 2 Testing:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** |
| 2.1 | Testing is\_valid\_date() with a correct date format | given\_date = “01/01/2001” | True returned |
| 2.2 | Testing is\_valid\_date() with an incorrect date | given\_date = “32/13/2001” | False returned |
| 2.2b | Testing is\_valid\_date() with an incorrect data type | given\_date = 2 | False returned |
| 2.3 | Testing is\_valid\_date() with extreme data for the given date | given\_date = “30/12/9999” | True returned |
| 2.4 | Attempting to use is\_valid\_user() with a valid username | u = “test\_user1” opt = “username” | True returned |
| 2.5 | Attempting to use is\_valid\_user() with an invalid string username | u = “username\_exceeding\_ chr\_limit” opt = “username” | False returned |
| 2.6 | Checking is\_valid\_user() with an invalid data type for username | u = 10 opt = “username” | False returned |
| 2.7 | Testing is\_valid\_user() with a valid string password | u = “test\_password1” opt = “password” | True returned |
| 2.8 | Testing is\_valid\_user() with an invalid password that is too short | u = “test” opt = “password” | False returned |
| 2.9 | Testing is\_valid\_user() with an invalid password that is incorrect data type | u = True opt = “password” | False returned |
| 2.10 | Attempting to run is\_valid\_user() with an valid password that is borderline (8 characters long) | u = “test1234” opt = “password” | True returned |

##### Automatic Testing

I have continued to use automatic testing for this stage, building the validation section, because the additional functions in this version are also simple and only return a Boolean value. This makes unittesting the functions effective. Unittesting is only possible because they are functions rather than complex procedure as will be seen later in the game section of the project.

TestMethods class:

1. **class** TestMethods(unittest.TestCase):
2. *# -----------------------------------------------------------------*
3. *# TESTING VALIDATION*
5. ***# testing dates dd/mm/yyyy format***
6. **def** test\_dates(self):
7. self.assertTrue(is\_valid\_date("01/01/2001")) *# Valid date*
8. self.assertFalse(is\_valid\_date("32/13/2001")) *# Invalid date*
9. self.assertFalse(is\_valid\_date(2)) *# Invalid date, wrong type*
10. **self.assertTrue(is\_valid\_date("30/12/9999")) *# Extreme (but valid)***
12. *# testing usernames*
13. **def** test\_users(self):
14. self.assertTrue(is\_valid\_user("test\_user1", "username")) *# Valid username*
15. **self.assertFalse(is\_valid\_user("username\_exceeding\_chr\_limit", "username")) *# Invalid username - too long***
16. self.assertFalse(is\_valid\_user(10, "username")) *# Invalid username - wrong type*
18. *# testing passwords*
19. **def** test\_passwords(self):
20. **self.assertTrue(is\_valid\_user("test\_password1", "password")) *# Valid password***
21. self.assertFalse(is\_valid\_user("test", "password")) *# Invalid password - too short*
22. self.assertFalse(is\_valid\_user(True, "password")) *# Invalid password - wrong type*
23. self.assertTrue(is\_valid\_user("test1234", "password")) *# Borderline valid*

##### Automatic Testing Results:



As evident, the automatic testing for Version 2 was successful, indicating that tests 2.1 – 2.10 returned the expected values. This section of the project is now fully functioning and tested, meaning I will now move on to developing and testing the next stage – the database.

## 3.2.2 Stage 2: Building the Database

### 3.2.2.1 SQLITE + Python Code

I have built the database according to my design in section 2.4 – Database Design. The name of the file that contains the code for this module is LoginData.py.

#### Imported modules

1. **import** sqlite3
2. **import** validation
3. **from** messages **import** \*

I have used sqlite3 to store my database. This means that the database will be stored locally on the machine. I will also use the messages module so an error message can be displayed in case there is an error that occurs with the database. I chose to use sqlite3 because having a local database is suitable for my game and having an online database was not one of the stakeholder requirements.

#### Version 1

Create\_table() function:

1. **def** create\_table():
2. **try**:
3. con = sqlite3.connect("LoginScores.db")
4. con.execute('''CREATE TABLE IF NOT EXISTS Users
5. **(Username VARCHAR PRIMARY KEY NOT NULL,**
6. Password TEXT NOT NULL);''')
8. con.execute('''CREATE TABLE IF NOT EXISTS Admins
9. (Username VARCHAR PRIMARY KEY NOT NULL,
10. **Password TEXT NOT NULL);''')**
11. con.commit()
12. con.close()
13. messages.show\_message("Success", "Database created successfully.", 1)
14. **except** Exception **as** ex:
15. **messages.show\_message("Error", ex, 2)**

This function connects to the database and creates the Users and Admins tables if they don’t already exist. The Username is the primary key field for both tables because each user must have a unique username.

##### Version 1 Testing:

I tested this code by running the function. As expected, the success message appeared on screen and the database file (LoginScores.db) was created.



#### Version 2

Enter\_user() function:

1. *# Function that inserts a user (username + password) into the database*
2. **def** enter\_user(u, p):
3. con = sqlite3.connect("LoginScores.db")
4. **try**:
5. **con.execute('''insert into Users (Username, Password) values (?, ?)''',**
6. (u, p))
7. con.commit()
8. con.close()
9. show\_message("Success", "Successfully added user: " + u, 1)
10. **return True**
11. except Exception as ex:
12. show\_message("Error creating user", ex, 2)
13. con.close()
14. **return** False

The enter\_user() function takes a username and password as parameters and inserts them into the Users table of the database. This function will be used by Admins to add new users.

Search() function:

1. **def** search(u, p, table):
2. con = sqlite3.connect("LoginScores.db")
3. cursor = con.cursor()
5. **if table == "Admins":**
6. cursor.execute("SELECT \* FROM Admins")
7. **else**:
8. cursor.execute("SELECT \* FROM Users")
9. records = cursor.fetchall()
10. **found = False**
11. **for** row **in** records:
12. **if** row[0] == u **and** row[1] == p:
13. found = True
14. cursor.close()
15. **con.close()**
16. **return** found

In the login section, I will need to check if the details a user enters matches those in the database. This will be done using this search() function. This takes the username, password, and the desired table to conduct the search and returns True if it finds a matching record in that table.

##### Version 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 2.1 | Attempt to create a user with valid details using enter\_user() function. | U = “test\_user1”  P = “testpassword01” | User is entered successfully to the Users table. | Success |
| 2.2 | Attempt to create a user with borderline details | U = “test2”  P = “Password2” | User is entered successfully to the Users table. | Success |
| 2.3 | Attempt to create a user with invalid details. | U = “t3”  P = “inval” | User should not be added to the database. An error message should appear. | Failed - |
| 2.3.2 |  | U = 10  P = 5 | Wrong data type, an error message should appear. | Failed |
| 2.4 | Attempt to search for an existent user with search() function. | U = “test\_user1”  P = “testpassword01” | True returned | Success |
| 2.5 | Attempt to search for correct username but incorrect password. | U = “test\_user1”  P = “incorrectpass” | False returned | Success |
| 2.6 | Attempt to search for non-existent user but existent password. | U = “incorrectuser”  P = “testpassword01” | False returned | Success |
| 2.7 | Attempt to search for non-existent user | U = “incorrectuser” P = “incorrectpass” | False returned | Success |
| 2.8 | Attempt to search with invalid details | U = “”  P = “” | False returned | Success |
| 2.8b |  | U = “test\_user1”  P = “” | False returned | Success |
| 2.8c |  | U = “”  P = “testpassword01” | False returned | Success |

Test 2.1:



Test 2.2:



Adding a new user using the enter\_user() function was successful, but using invalid details also added them to the table when they should be rejected. I will improve this function by adding validation before entering the data.

#### Automatic Testing

**Unittest (testing.py):**

**Explain ……………..**

1. **class** TestMethods(unittest.TestCase):
2. *# -----------------------------------------------------------------------------*
3. *# TESTING DB SEARCH*
4. **def** test\_valid\_search(self): *# Correct username and password*
5. **self.assertTrue(search("test\_user1", "testpassword01", "Users"))**
7. **def** test\_false\_search(self):
8. self.assertFalse(search("test\_user1", "incorrectpass", "Users")) *# Wrong password*
9. self.assertFalse(search("incorrectuser", "testpassword01", "Users")) *# Wrong username*
10. **self.assertFalse(search("incorrectuser", "incorrectpass", "Users")) *# Both wrong***
12. **def** test\_invalid\_search(self):
13. self.assertFalse(search(100, 50, "Users"))
14. self.assertFalse(search("test\_user1", 50, "Users"))
15. **self.assertFalse(search(100, "testpassword01", "Users"))**

#### Version 2 Results:

In order to test the search function, I used the unittest module because it allows for more efficient, automatic testing which allows for a range of data to be tested quickly. All of the tests passed as shown in this evidence:

Text

Description automatically generated

Functions from LoginData module:

Enter\_user() function (improved):

1. **def** enter\_user(u, p):
2. *# Validation performed on username and password before they are entered*
3. val\_u = validation.is\_valid\_user(u, "username")
4. val\_p = validation.is\_valid\_user(p, "password")
5. **if val\_u and val\_p:**
6. **try**:
7. con = sqlite3.connect("LoginScores.db")
8. con.execute('''insert into Users (Username, Password) values (?, ?)''',
9. (u, p))
10. **con.commit()**
11. con.close()
12. show\_message("Success", "Successfully added user: " + u, 1)
13. **return** True
14. **except** Exception **as** ex:
15. **show\_message("Error creating user", ex, 2)**
16. con.close()
17. **return** False
18. **else**:
19. show\_message("Error creating user", "Invalid", 2)
20. **return** False

This improved enter user function now validates the user details before inserting them into the database. The username must be between 3 and 20 characters and the password must be greater than or equal to 8 characters. The username must also only be composed of valid characters (upper and lowercase letters as well as underscores and numbers). Both username and password must be strings.

Delete\_user() procedure:

1. **def** delete\_user(u):
2. **try**:
3. con = sqlite3.connect("LoginScores.db")
4. cursor = con.cursor()
5. ***# Deleting single record***
6. sql = "DELETE FROM Users WHERE Username=?"
7. cursor.execute(sql, (u,))
8. con.commit()
9. cursor.close()
10. **con.close()**
11. show\_message("Success", "User deleted ", 1)
13. **except** sqlite3.Error **as** error:
14. *# Displays popup message*
15. **show\_message("Error", "Failed to delete record from sqlite table: " + str(error), 2)**

I have added a procedure to delete a user record from the table. This will be used later whenever an admin needs to remove a user from the database.

#### Version 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 2.8 | Attempt to create a user with invalid details with enter\_user() function | U = “t8”  P = “inval” | User should not be added to the database, error message. | Success |
| 2.9 | Attempt to create a user with invalid details. | U = 5  P = 10 | Error message | Success |
|  |  |  |  |  |

**Unit test (testing.py):**

1. **class** TestMethods(unittest.TestCase):
2. *# ---------------------------------------------------------------*
3. *# TESTING VALIDATION*
5. ***# testing usernames***
6. **def** test\_users(self):
7. valid\_users = ["test\_user1", "Bob12542", "testusername", "qwertyuiop\_asgdhjlh","te1"]
8. invalid\_users = ["l1", "qwertyuiop\_asdfghjklzxc", 10, ""]
10. **for username in valid\_users:**
11. self.assertEqual(is\_valid\_user(username, "username"), True)
13. **for** username **in** invalid\_users:
14. self.assertEqual(is\_valid\_user(username, "username"), False)
16. *# testing passwords*
17. **def** test\_passwords(self):
18. valid\_passwords = ["XQloP7\*jsalHp!", "Testpass"]
19. invalid\_passwords = [100000000, "12345", "Short", "Bound12" ""]
21. **for** password **in** valid\_passwords:
22. self.assertEqual(is\_valid\_user(password, "password"), True)
24. **for** password **in** invalid\_passwords:
25. **self.assertEqual(is\_valid\_user(password, "password"), False)**
26. *# -------------------------------------------------------------*

I have used unit testing here to test the is\_valid\_user() function. This allows me to test a range of different usernames and passwords at once.  
I will test valid and invalid usernames separately, including boundary data of length 3 when the usernames must be between 3 and 20 characters in length. Invalid usernames also included different data types than the expected (string).  
Passwords will also be tested this way. The boundary data includes a password of length 8 when only passwords of 8 characters or greater should be accepted. A password of length 7 is also testing as boundary under the invalid section.

#### Version 3 Results:

2.8, 2.9:



The unit test of the is\_valid\_user() function was successful, and all tests when as expected as shown in this evidence:



### 3.2.2.2 Feedback from Stakeholder (if appropriate)

I have not taken feedback from stakeholders at this stage in the development. This is because this section is the database itself, the back-end of the login section. Taking stakeholder feedback at this point is not appropriate. I will consult stakeholders after developing the front end of the login section.

### 3.2.2.3 Review

With the main functions for the data developed, I will review the database.



Using [sqlite viewer](https://inloop.github.io/sqlite-viewer/) I can see the usernames and passwords stored in the database. Here, the password visible in plain text and has not been hashed or salted. This is a problem and compromises the security of the login system as well as not fitting the specifications that the system must hash user’s passwords.  
This means I must fix my functions to incorporate the hashing of passwords in the database.

#### Version 4

Enter\_user function (fixed):

1. **def** enter\_user(u, p):
2. *# Validation performed on username and password before they are entered*
3. val\_u = validation.is\_valid\_user(u, "username")
4. **val\_p = validation.is\_valid\_user(p, "password")**
5. **if** val\_u **and** val\_p:
6. con = sqlite3.connect("LoginScores.db")
7. *# Convert password to bytes, hash and salt it*
8. p = bytes(p, encoding='utf-8')
9. **hash\_p = bcrypt.hashpw(p, bcrypt.gensalt())**
10. **try**:
11. con.execute('''insert into Users (Username, Password) values (?, ?)''',
12. (u, hash\_p))
13. con.commit()
14. **con.close()**
15. **return** True
16. **except** Exception **as** ex:
17. show\_message("Error creating user", ex, 2)
18. con.close()
19. **return False**
20. **else**:
21. show\_message("Error creating user", "Invalid", 2)
22. **return** False

Search function (fixed):

1. **def** search(u, p, table):
2. con = sqlite3.connect("LoginScores.db")
3. cursor = con.cursor()
4. p = bytes(p, encoding='utf-8')
6. **if** table == "Admins":
7. cursor.execute("SELECT \* FROM Admins")
8. **else**:
9. cursor.execute("SELECT \* FROM Users")
10. **records = cursor.fetchall()**
11. found = False
12. **for** row **in** records:
13. **if** row[0] == u **and** bcrypt.checkpw(p, row[1]):
14. found = True
15. **cursor.close()**
16. con.close()
17. **return** found

##### Version 4 Results:



The password is now successfully hashed in the database.

## 3.2.3 Stage 3: Building Login

### 3.2.3.1 Code for Login

#### Imported modules

1. **import** tkinter **as** tk
2. **from** tkinter **import** ttk
3. **from** messages **import** \*
4. **from** LoginData **import** search

For the imported modules of this section, I have decided to use tkinter because it can be used to easily display the simple, static GUI of the login windows. Tkinter also is the only framework built into the Python standard library that allows for creation of application interfaces.

#### Version 1

LoginWindow() class:

1. **class** LoginWindow(tk.Tk):
2. **def** \_\_init\_\_(self):
3. super().\_\_init\_\_()
5. **self.title("Login Window")**
6. self.geometry("600x350")
7. *# Window icon*
8. self.iconbitmap("graphics/saturn.ico")
9. *# title*
10. **self.label = ttk.Label(self, text='Welcome, Please Log In.', font=("Helvetica", 25, "bold"))**
11. self.label.pack()
13. *# text entry boxes*
14. self.entry1 = tk.Entry(self, bd=6, width=40)
15. **self.entry1.place(x=250, y=100)**
16. self.entry2 = tk.Entry(self, bd=6, width=40)
17. self.entry2.place(x=250, y=150)
19. *# login button*
20. **self.button = ttk.Button(self, text='Login')**
21. self.button.place(x=475, y=300)
23. *# exit button*
24. self.button2 = ttk.Button(self, text='Quit', width=10)
25. **self.button2.place(x=50, y=300)**
27. *# username and password text*
28. self.label = ttk.Label(self, text='User Name:', font=("Arial", 15))
29. self.label.place(x=75, y=100)
30. **self.label = ttk.Label(self, text='Password:', font=("Arial", 15))**
31. self.label.place(x=75, y=150)

create\_window function:

1. **def** create\_window():
2. login = LoginWindow()
3. login.mainloop()

For the login section I have decided to implement object-oriented programming. This is seen in this LoginWindow class which I have developed the attributes for (but not the functions).

##### Version 1 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 3.1 | Attempt to display the window | Running create\_window() | Window displays correctly, including text boxes and labels, without error | Success |
| 3.2 | Attempt to close the window | Clicking the X in the top right of window. | Window closes | Success |
| 3.3 | Attempt to resize the window | Running create\_window() and dragging the sides of the window. | Nothing should happen, the window should not be resizable. | Failed |
| 3.4 | Attempt to type in the text entry boxes | Clicking on entry boxes and inputting “Test text” into each one | Text entry boxes should display text correctly. | Success |
| 3.5 | Ensuring that the login window appears in the centre of the screen every time | Running create\_window multiple times | All windows should appear in the centre of the screen | Failed |

##### Version 1 Results:

3.1:



3.3:



As shown here, the proportions of parts of the window don’t work when the window is scaled to be much larger or smaller. I will focus on fixing this bug in the following iteration by making it so that the window size is fixed and the user will not be able to change it.

3.4



3.5:



Another issue that needs to be fixed is that the window appears in the top left of the screen, rather than being placed in the centre. This issue also makes the login window UI less convenient for the user, so it will be fixed in the next iteration.

#### Version 2

LoginWindow() class (fixed):

1. **class** LoginWindow(tk.Tk):
2. **def** \_\_init\_\_(self):
3. super().\_\_init\_\_()
5. **self.title("Login Window")**
6. self.geometry("600x350")
7. self.resizable(False, False) *# Window not resizable*
8. self.eval('tk::PlaceWindow . center') *# Appears in centre*
9. *# Window icon*
10. **self.iconbitmap("graphics/saturn.ico")**
11. *# title*
12. self.label = ttk.Label(self, text='Welcome, Please Log In.', font=("Helvetica", 25, "bold"))
13. self.label.pack()
15. ***# text entry boxes***
16. self.entry1 = tk.Entry(self, bd=6, width=40)
17. self.entry1.place(x=250, y=100)
18. self.entry2 = tk.Entry(self, bd=6, width=40)
19. self.entry2.place(x=250, y=150)
21. *# login button*
22. self.button = ttk.Button(self, text='Login')
23. self.button.place(x=475, y=300)
25. ***# exit button***
26. self.button2 = ttk.Button(self, text='Quit', width=10)
27. self.button2.place(x=50, y=300)
29. *# username and password text*
30. **self.label = ttk.Label(self, text='User Name:', font=("Arial", 15))**
31. self.label.place(x=75, y=100)
32. self.label = ttk.Label(self, text='Password:', font=("Arial", 15))
33. self.label.place(x=75, y=150)
35. **def log\_in(self):**
36. username = self.entry1.get()
37. password = self.entry2.get()
38. **if** search(str(username), str(password), "Users"):
39. show\_message("", "Welcome " + str(username), 1)
40. **LoginWindow.destroy(self)**
41. *# Closes login and takes the user to the game (implement later)*
42. **else**:
43. show\_message("", "Incorrect details", 1)
45. **def cancel(self):**
46. ans = show\_message("", "Exit?", 4)
47. **if** ans:
48. LoginWindow.destroy(self)
49. quit()

In addition to fixing the issues with Version 1, I have implemented the main functions of the LoginWindow() class.

The log\_in() procedure obtains the data put into the username and password and searches for this in the “Users” table of the database using the function in LoginData. If it finds a matching record, the game will be launched. If no matching username and password are found, an invalid details message is displayed and the user can attempt to log in again.

##### Version 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 3.3 | Attempt to resize the window | Dragging the sides of the window | The window should not be resizable | Success |
| 3.5 | Making sure the window is always created in the centre of the screen | Running create\_window() multiple times | Windows display correctly and is position in the centre of the screen | Success |
| 3.6a | Attempt to exit using quit button | Clicking “Quit” in login window then “Yes” on the messagebox | The window should close | Success |
| 3.6b | Attempt to cancel exit after using quit button | Clicking “Quit” in login window then “No” on the messagebox | The window should not close | Success |

##### Version 2 Results:

3.3:



As shown in video 3.3, the window can no longer be resized, but it is still possible to drag and move the window.

3.5:



All windows appear in the same place at the centre of the screen.

#### Version 3

I review of the previous versions, there is another feature that I must add. When the user is typing text into the password entry box, the text is clearly visible. This is a security risk because anyone could find out a user’s password simply. by looking at the plaintext.   
I will fix this by making the text entry box display only asterisks when the enters text. There will be a “show password” checkbox in case a user would like to see the password they typed.

LoginWindow() Class (Improved):

1. **class** LoginWindow(tk.Tk):
2. **def** \_\_init\_\_(self):
3. super().\_\_init\_\_()
5. **self.title("Login Window")**
6. self.geometry("600x350")
7. self.resizable(False, False) *# Window not resizable*
8. self.eval('tk::PlaceWindow . center') *# Appears in centre*
9. *# Window icon*
10. **self.iconbitmap("graphics/saturn.ico")**
11. *# title*
12. self.label = ttk.Label(self, text='Welcome, Please Log In.', font=("Helvetica", 25, "bold"))
13. self.label.pack()
15. ***# text entry boxes***
16. self.entry1 = tk.Entry(self, bd=6, width=40)
17. self.entry1.place(x=250, y=100)
18. self.entry2 = tk.Entry(self, bd=6, width=40)
19. self.entry2.place(x=250, y=150)
20. **self.entry2.config(show="\*")**
21. self.hidden = True
23. *# login button*
24. self.button = ttk.Button(self, text='Login')
25. **self.button['command'] = lambda: self.log\_in()**
26. self.button.place(x=475, y=300)
28. *# exit button*
29. self.button2 = ttk.Button(self, text='Quit', width=10)
30. **self.button2['command'] = self.cancel**
31. self.button2.place(x=50, y=300)
33. *# username and password text*
34. self.label = ttk.Label(self, text='User Name:', font=("Arial", 15))
35. **self.label.place(x=75, y=100)**
36. self.label = ttk.Label(self, text='Password:', font=("Arial", 15))
37. self.label.place(x=75, y=150)
39. *# show password checkbox*
40. **self.check1 = tk.Checkbutton(self, text='Show Password', onvalue=True, offvalue=False)**
41. self.check1['command'] = **lambda**: self.toggle\_pass()
42. self.check1.place(x=420, y=220)
44. **def** toggle\_pass(self):
45. ***# Show or hide password box***
46. **if** self.hidden:
47. self.entry2.config(show="")
48. self.hidden = False
49. **else**:
50. **self.entry2.config(show="\*")**
51. self.hidden = True
53. **def** log\_in(self):
54. username = self.entry1.get()
55. **password = self.entry2.get()**
56. **if** search(str(username), str(password), "Users"):
57. show\_message("", "Welcome " + str(username), 1)
58. LoginWindow.destroy(self)
59. *# Closes login and takes the user to the game (implement later)*
60. **else:**
61. show\_message("", "Incorrect details", 1)
63. **def** cancel(self):
64. ans = show\_message("", "Exit?", 4)
65. **if ans:**
66. LoginWindow.destroy(self)
67. quit()

.config(show=”\*”) is used to replace the text with asterisks and .config(show=””) is used to reveal the text again.

##### Version 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 3.1 | Attempt to display the window | Running create\_window | The window should display correctly, including the display password checkbox | Success |
| 3.4 | Typing in the text entry boxes | Clicking on entry boxes and inputting “Test text” into each one | The username text should display but password text should be hidden by \*\*\*\* | Success |
| 3.7 | Attempt to show password | Clicking on “Show Password” checkbox | The password should display | Success |
| 3.8 | Attempt to hide the password again | Unchecking the “Show Password” checkbox | The password should be hidden | Success |
|  |  |  |  |  |

##### Version 3 Results:

3.1:

Graphical user interface, text, application, email

Description automatically generated

3.4:

Graphical user interface, application

Description automatically generated

3.7:

Graphical user interface, application

Description automatically generated

3.8:

Graphical user interface, text, application

Description automatically generated

### 3.2.3.3 Feedback from Stakeholder (if appropriate)

I showed the most recent version of the Login section of my project to all of my stakeholders. They agreed that the design of the window is suitable and would function as a login system for the game but some of my stakeholders suggested quality of life improvements that would make the login section much more user-friendly.

One of these suggested improvements was to implement key bindings so that the user can press enter to log in quickly rather than having to click the button every time. This could also apply to bind the escape key to run the function of the quit button without having to click the button itself.

Another suggested improvement that would make the login window much more customisable for the user would be the ability to change the background colour. While my stakeholders were okay with the default background colour, they felt as if having the ability to change it would greatly improve the user’s experience. In order for this to be done, the user should be able to bring up a colour picker window and select a colour they want as the background. This would then be saved to a text file so that their choice of colour is saved even after they exit the program.   
In the next iteration of the design of the login section, I will implement these new features and test them to make sure they function as intended before proceeding onto the next stage.

#### Version 4

Additional imported modules:

1. **from** tkinter **import** Menu
2. **from** tkinter.colorchooser **import** askcolor

In these imports, the Menu module is used for the dropdown menu and the akscolor() function is imported to display the colour choose window and return the colour that the user selects (in hexadecimal).

The LoginWindow() class is similar to how it was in the previous version, but with the addition of the bind commands on lines 28 and 34, the dropdown menu on lines 47 – 53, and the introduction of the change\_colour() method. This new method brings up the colour picker, sets this colour to the background and saves this choice of colour to a text file. This text file is read and set as the background whenever window is created, allowing for the user’s choice of colour to be saved even after exiting and reopening the program.

LoginWindow() Class (Improved):

1. **class** LoginWindow(tk.Tk):
2. **def** \_\_init\_\_(self):
3. super().\_\_init\_\_()
5. **self.menubar = Menu(self)**
6. self.title("Login Window")
7. self.geometry("600x350")
8. self.resizable(False, False) *# Window not resizable*
9. self.eval('tk::PlaceWindow . center') *# Appears in centre*
10. ***# Window icon***
11. self.iconbitmap("graphics/saturn.ico")
12. *# title*
13. self.label = ttk.Label(self, text='Welcome, Please Log In.', font=("Helvetica", 25, "bold"))
14. self.label.pack()
16. *# text entry boxes*
17. self.entry1 = tk.Entry(self, bd=6, width=40)
18. self.entry1.place(x=250, y=100)
19. self.entry2 = tk.Entry(self, bd=6, width=40)
20. **self.entry2.place(x=250, y=150)**
21. self.entry2.config(show="\*")
22. self.hidden = True
24. *# login button*
25. **self.button = ttk.Button(self, text='Login')**
26. self.button['command'] = **lambda**: self.log\_in()
27. self.button.place(x=475, y=300)
28. self.bind("<Return>", (**lambda** event: self.log\_in())) *# Binds enter key to log in*
30. ***# exit button***
31. self.button2 = ttk.Button(self, text='Quit', width=10)
32. self.button2['command'] = self.cancel
33. self.button2.place(x=50, y=300)
34. self.bind("<Escape>", (**lambda** event: self.cancel())) *# Binds escape key to exit*
36. *# username and password text*
37. self.label = ttk.Label(self, text='User Name:', font=("Arial", 15))
38. self.label.place(x=75, y=100)
39. self.label = ttk.Label(self, text='Password:', font=("Arial", 15))
40. **self.label.place(x=75, y=150)**
42. *# show password checkbox*
43. self.check1 = tk.Checkbutton(self, text='Show Password', onvalue=True, offvalue=False)
44. self.check1['command'] = **lambda**: self.toggle\_pass()
45. **self.check1.place(x=420, y=220)**
47. *# dropdown menu*
48. self.config(menu=self.menubar)
49. file\_menu = Menu(self.menubar)
50. **file\_menu.add\_command(label='Change colour ', command=lambda: self.change\_color())**
51. file\_menu.add\_command(label='Quit', command=self.destroy)
53. self.menubar.add\_cascade(label="Options", menu=file\_menu, underline=0)
55. ***# Read saved colour from text file and set as bg***
56. **with** open("colour.TXT") **as** f:
57. lines = []
58. **for** line **in** f:
59. lines.append(line.strip())
60. **self.configure(bg=lines[0])**
62. **def** toggle\_pass(self):
63. *# Show or hide password box*
64. **if** self.hidden:
65. **self.entry2.config(show="")**
66. self.hidden = False
67. **else**:
68. self.entry2.config(show="\*")
69. self.hidden = True
71. **def** log\_in(self):
72. username = self.entry1.get()
73. password = self.entry2.get()
74. **if** search(str(username), str(password), "Users"):
75. **show\_message("", "Welcome " + str(username), 1)**
76. LoginWindow.destroy(self)
77. *# Closes login and takes the user to the game (implement later)*
78. **else**:
79. show\_message("", "Incorrect details", 1)
81. **def** cancel(self):
82. ans = show\_message("", "Exit?", 4)
83. **if** ans:
84. LoginWindow.destroy(self)
85. **quit()**
87. **def** change\_color(self):
88. colors = askcolor(title="Colour Chooser") *# Colour picker window*
89. self.configure(bg=colors[1])
90. **if colors[1]:**
91. **with** open("colour.TXT", "w") **as** f:
92. f.write(colors[1])

##### Version 4 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| **3.1** | **Attempt to display the window** | **Running create\_window()** | **The window should display correctly, with the default colour from the text file loaded** |  |
| **3.9a** | **Attempt to quit using the key binding (escape)** | **With the window open, pressing escape** | **The confirm exit message should appear, when “yes” is clicked the window should close** |  |
| **3.9b** | **Attempting use the key binding (enter)** | **Pressing enter while window is open (both entry boxes left empty)** | **This should act as if the user had clicked “login” and show invalid details message** |  |
| **3.9c** | **Logging in to an exist user account with the login key binding** | **Pressing enter with correct details in the text entry boxes (**“test\_user1”and “testpassword01) | **This should display the login welcome message as if the user had clicked “login”** |  |
| **3.10** | **Attempting to change the colour of the window** | **Clicking the dropdown menu, clicking “Change colour” and then selecting the new colour as red** | **The login window colour should immediately change to red** |  |
| **3.11** | **Checking that the colour is saved and correctly loaded on restart** | **Closing the window and running\_window() again** | **The background colour should still be red as the user’s colour choice should be saved even after exiting** |  |

##### Version 4 Results:

3.1:



3.9a:

Video 3.9a

3.9b:

Video 3.9b

3.9c:

Video 3.9c

3.10:

Video 3.10

3.11:

Video 3.11

As evidenced by the above videos, all of the tests for the new features of the login system have passed. Therefore, the stakeholder feedback has been successfully implemented and I can move on to developing the next section of the project – the game.

## 3.2.3 Stage 4: Building Main Game

This stage of development and testing is for the building of the main game. For this I will be coding in two modules: game.py and sprites.py. The file game.py will contain the main pygame loop and all variables and functions for the game that are outside of classes. Sprites.py will contain all of the classes that are used to create sprites or groups of sprites in the game.

### 3.2.3.1 Building the Single Player Mode

#### Imported modules

Game.py:

1. **from** sys **import** exit
3. **import** login
4. **from** messages **import** \*
5. **import os**
7. os.environ['PYGAME\_HIDE\_SUPPORT\_PROMPT'] = "hide" *# Hides pygame welcome message, must be before pygame import*
8. **from** sprites **import** \*

Sprites.py:

1. **from** pygame **import** \*
2. **import** random

For the imports, I have imported exit from sys in order to allow because this will be used to close the pygame window when the user clicks the X in the top right corner. I have imported login because later on I will use it to allow the user to go back to the login section from the game. Importing sprites allows me to use the classes that I will write in it.

### 3.2.3.3 Feedback from Stakeholder (if appropriate)

### 3.2.3.4 Review

## 3.2.4 Stage 2: Building …….. PART FIVE ….

### 3.2.4.1 Code for … Part FIVE… including validation

### 3.2.4.2 Testing …. Part FIVE ….including validation

### 3.2.4.3 Feedback from Stakeholder (if appropriate)

### 3.2.4.4 Review

## 3.3 Final Review, Improvements and Corrective Actions

Chapter Four: Evaluation

Evaluation

## 4.1 Introduction

## 4.2 Testing to inform evaluation

### 4.2.1 Testing

### 4.2.2 Feedback from Stakeholders

## 4.3 Evaluation

## 4.4 Evaluating usability features

## 4.5 Limitations and Maintenance