**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](#_Toc126924114)

[1.2 Problem Identification 8](#_Toc126924115)

[1.3 Why the problem is suited to a computational solution 9](#_Toc126924116)

[1.4 Stakeholders analysis 10](#_Toc126924117)

[1.5 Research 11](#_Toc126924118)

[1.6 Features of the proposed solution 15](#_Toc126924119)

[1.7 Limitations of the solution 15](#_Toc126924120)

[1.8 Stakeholder Consultation 16](#_Toc126924121)

[1.9 Hardware and software requirements 18](#_Toc126924122)

[1.10 The requirements of the solution 18](#_Toc126924123)

[1.11 Success Criteria 20](#_Toc126924124)

[Chapter Two: Design 22](file:///H:\Project\Report\Louis_Pattern_analysis.docx#_Toc126924125)

[2.1 Introduction 24](#_Toc126924126)

[2.2 Decomposition of the problem 25](#_Toc126924127)

[2.2.1 Decomposition Diagram 25](#_Toc126924128)

[2.2.2 Data Flow Diagram 26](#_Toc126924129)

[2.2.3 Input Process Output 27](#_Toc126924130)

[2.3 How All Solution Parts are Linked 28](#_Toc126924131)

[2.3.1 State Diagram of the different forms/parts 28](#_Toc126924132)

[2.3.2 How different functions /classes are connected 29](#_Toc126924133)

[2.4 Database Design 31](#_Toc126924134)

[2.4.1 Normalisation 31](#_Toc126924135)

[2.4.2 Data Dictionary 31](#_Toc126924136)

[2.4.3 Entity Relationship Diagram 32](#_Toc126924137)

[2.4.4 SQL Pseudocode 32](#_Toc126924138)

[2.5 Design of Main Parts of the Solution 33](#_Toc126924139)

[2.5.1.2 Form Design and Layout 33](#_Toc126924140)

[Login window 33](#_Toc126924141)

[2.5.1.2 Justification of Validation rules 34](#_Toc126924142)

[2.5.1 .3 Algorithms and PseudoCode 34](#_Toc126924143)

[2.5.1.4 Key Variables/Data Structures /Classes 36](#_Toc126924144)

[2.5.1.5 Test Plan for PART ONE 37](#_Toc126924145)

[2.5.2.1 Form Design and Layout 38](#_Toc126924146)

[Admin window 38](#_Toc126924147)

[2.5.2.2 Justification of Validation rules 39](#_Toc126924148)

[2.5.2. 3 Algorithms and PseudoCode 39](#_Toc126924149)

[2.5.1.5 Test Plan for PART TWO 40](#_Toc126924150)

[2.5.3.1 Main Game Design and Layout 43](#_Toc126924151)

[Game main menu 43](#_Toc126924152)

[Game settings menu 44](#_Toc126924153)

[Highscores Menu 45](#_Toc126924154)

[2.5.3.2 Justification of Validation rules 45](#_Toc126924155)

[2.5.3. 3 Algorithms and PseudoCode 46](#_Toc126924156)

[2.5.3.4 Key Variables/Data Structures /Classes 48](#_Toc126924157)

[2.5.1.5 Test Plan for PART THREE 48](#_Toc126924158)

[2.5.4 Part FOUR – Error Handling 49](#_Toc126924159)

[2.5.4.1 Form Design and Layout 49](#_Toc126924160)

[2.5.4.2 Algorithms and PseudoCode 50](#_Toc126924161)

[2.5.1.5 Test Plan for PART FOUR 51](#_Toc126924162)

[2.5.4 Part FIVE – Design of Main Game 52](#_Toc126924163)

[2.5.4.1 Window Design and Layout 52](#_Toc126924164)

[2.5.4.2 Algorithms 53](#_Toc126924165)

[2.6 Stakeholders involvement 53](#_Toc126924166)

[Revised Admin Window 53](#_Toc126924167)

[Pseudocode 54](#_Toc126924168)

[2.7 Testing plan to inform evaluation 54](#_Toc126924169)

[Integration testing: 54](#_Toc126924170)

[Scenario testing: 55](#_Toc126924171)

[Chapter Three: Development and Testing 57](file:///H:\Project\Report\Louis_Pattern_analysis.docx#_Toc126924172)

[3.1 Introduction 58](#_Toc126924173)

[Modules 58](#_Toc126924174)

[3.2.1 Stage 1: Building the Error Handling 59](#_Toc126924175)

[3.2.1.1 Code for Error handling section 59](#_Toc126924176)

[Imported modules 59](#_Toc126924177)

[Prototype 1 60](#_Toc126924178)

[Prototype 1 Testing: 60](#_Toc126924179)

[Prototype 1 Results: 60](#_Toc126924180)

[Prototype 2 61](#_Toc126924181)

[Prototype 2 Testing: 61](#_Toc126924182)

[Prototype 2 Results: 61](#_Toc126924183)

[Prototype 3 62](#_Toc126924184)

[Prototype 3 Testing: 62](#_Toc126924185)

[Prototype 3 Results: 63](#_Toc126924186)

[3.2.1.2 Feedback from Stakeholder 64](#_Toc126924187)

[3.2.2 Stage 2: Building the Database 65](#_Toc126924188)

[3.2.2.1 SQLITE + Python Code 65](#_Toc126924189)

[Imported modules 65](#_Toc126924190)

[Prototype 1 65](#_Toc126924191)

[Prototype 1 Testing: 66](#_Toc126924192)

[Prototype 2 66](#_Toc126924193)

[Prototype 2 Testing: 67](#_Toc126924194)

[Prototype 2 Results: 68](#_Toc126924195)

[Prototype 3 69](#_Toc126924196)

[Prototype 3 Testing: 72](#_Toc126924197)

[Prototype 3 Results: 73](#_Toc126924198)

[3.2.2.2 Feedback from Stakeholder (if appropriate) 73](#_Toc126924199)

[3.2.2.3 Review 74](#_Toc126924200)

[Prototype 4 74](#_Toc126924201)

[Prototype 4 Results: 75](#_Toc126924202)

[3.2.3 Stage 3: Building Login 76](#_Toc126924203)

[3.2.3.1 Code for Login (including validation) 76](#_Toc126924204)

[Imported modules 76](#_Toc126924205)

[Prototype 1 76](#_Toc126924206)

[Prototype 1 Testing: 77](#_Toc126924207)

[Prototype 1 Results: 78](#_Toc126924208)

[Prototype 2 80](#_Toc126924209)

[Prototype 2 Testing: 82](#_Toc126924210)

[Prototype 2 Results: 82](#_Toc126924211)

[Prototype 3 83](#_Toc126924212)

[Prototype 3 Testing: 85](#_Toc126924213)

[Prototype 3 Results: 86](#_Toc126924214)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 86](#_Toc126924215)

[3.2.3 Stage 4: Building …….. PART FOUR …. 86](#_Toc126924216)

[3.2.3.1 Code For … part FOUR… (including validation) 86](#_Toc126924217)

[3.2.3.2 Testing …. Part FOUR ….including validation 86](#_Toc126924218)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 86](#_Toc126924219)

[3.2.3.4 Review 87](#_Toc126924220)

[3.2.4 Stage 2: Building …….. PART FIVE …. 87](#_Toc126924221)

[3.2.4.1 Code for … Part FIVE… including validation 87](#_Toc126924222)

[3.2.4.2 Testing …. Part FIVE ….including validation 87](#_Toc126924223)

[3.2.4.3 Feedback from Stakeholder (if appropriate) 87](#_Toc126924224)

[3.2.4.4 Review 87](#_Toc126924225)

[3.3 Final Review, Improvements and Corrective Actions 88](#_Toc126924226)

[Chapter Four: Evaluation 90](file:///H:\Project\Report\Louis_Pattern_analysis.docx#_Toc126924227)

[4.1 Introduction 91](#_Toc126924228)

[4.2 Testing to inform evaluation 91](#_Toc126924229)

[4.2.1 Testing 91](#_Toc126924230)

[4.2.2 Feedback from Stakeholders 92](#_Toc126924231)

[4.3 Evaluation 93](#_Toc126924232)

[4.4 Evaluating usability features 93](#_Toc126924233)

[4.5 Limitations and Maintenance 93](#_Toc126924234)

Chapter One: Analysis of the problem

## 1.1 Introduction

In this project, I aim of entertaining children and teenagers by providing them with a 2D videogame for recreational use. The main goal of the game should be to help teenagers relax in their free time when they are not doing homework or revision. My game will be suitable for college students and even younger children. Its theme is science fiction and space. The game will have both options for a single player fighting an AI 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. The game will be very customisable, allowing the user to change settings both in the game section of the project and in the login window that precedes the actual game. My stakeholders for this project will include a college student, a child and a young adult that will be discussed more on in later sections.

## 1.2 Problem Identification

Many teenagers become stressed or worried about exams and schoolwork. Videogames could help relieve this stress 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.  
In addition, most 2D videogames that feature a 1vs1 format are usually fighting games, a genre that most people are turned away from because it can be too competitive, forcing players to learn specific ‘combo’ moves reducing the ability for people to play casually and for fun. This could either put people of retro videogames entirely due to frustration or make younger users shy away from multiplayer gameplay entirely. There is also a limited number of retro shooter games with such a format and even fewer with a sci-fi theme.

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.   
My game would feature a unique background for each level. This would not only help to keep a user’s interest high (particularly for children) but also make the give the player a sense that they are travelling and progressing through the game.

## 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. In addition, 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 to enable a colour-blind mode. Finally, most people with disabilities are still able to play because usual input methods such as mouse and keyboard do not require much movement.

Videogames also allow the user to have more customisable settings, for example: window size and colours used.

## 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

One existing videogame of a similar format is Defender, an arcade game from 1981. 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.



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 will 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.



Defender scoring system

Disadvantages:

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

Another videogame of similar format is the Touhou series, specifically the windows games under the ‘bullet hell’ genre. This is a genre of 2-D shooters with a focus on a large number of projectiles or bullets that the player must avoid while shooting enemies and progressing through a number of levels. These games place emphasis on movement and the player’s dodging capabilities, with it being better to simply avoid getting hit at all whilst picking up score bonuses in a level rather than killing as many enemies as possible. The player has a limited number of lives but getting a high score can lead to bonus lives being awarded. The Touhou games also feature different playable characters, each with their own attack style (for example one character might launch projectiles in a wide arc whereas another might have a more focussed line of bullets). Characters also unlock special attacks such as homing bullets as the player progresses through the game and some characters have higher or lower movement speed.   
The difficulty level of the game can also be changed from a settings menu, making the game overall easier or harder to complete.

The UI for these games often displays the player’s score and other information in a column on the right-hand side of the screen and the game itself is displayed as a column on the left. This is appropriate considering the levels scroll vertically rather than horizontally. However, this approach would not be suitable for my game as it based on moving horizontally.



Touhou – “Double Dealing Character” UI and gameplay

Parts that I will apply to my solution:

The ability for the player to change the difficult from a menu is something I will apply to my game. Changing the difficulty effects a number of key elements such as: enemy projectile speed, the number of lives the player gets at the start of the first level, the amount of health a ‘boss’ enemy gets, the score the player receives (the harder the difficulty the higher the score). Setting the difficulty to its highest level also causes the number of enemies and enemy projectiles to increase a lot.

Disadvantages:

Because of the games focus on a large quantity of bullets, there can be a high number of projectiles on the screen at once, the game can suffer from performance issues on less powerful machines. This means (depending on hardware) it is not uncommon that the frame rate drops to below 40 frames per second. This can clearly be frustrating to the user.  
The games are also in Japanese with   
Another disadvantage of the Touhou series is the colour scheme. There are often many bright and intense colours in the UI, characters and projectiles. The colours can often be painful to look at or strain the eyes. Furthermore, can often make it harder to focus during gameplay, especially since the background almost never a single colour. This also makes the game less accessible to people with visual disabilities such as colour-blindness.

## 1.6 Features of the proposed solution

* 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.

## 1.7 Limitations of the solution

* One limitation of the solution is that the game can only use keyboard inputs alone and there is no support for mouse, controllers, or joystick input. This is acceptable because the most likely audience for the game – college students during their free periods or casual PC gamers would be less likely to prefer controller.
* Another limitation is that the max frames per second of the game is capped at 60 fps. This is because movement in the pygame loop is based frame by frame rather than real world time so increasing the frame rate beyond 60 increases the speed at which the player can move etc. This limitation is not a major issue since 60 fps has been the standard for gamers for a long time, especially for 2D games, where stability and a constant frame rate is often preferred over a higher, fluctuating frame rate. 60 frames per second appears smooth enough to the human eye. In addition, almost all modern hardware would be able to run my game at a stable 60 fps.
* The graphics for the game are all simple 2-dimensional images rather than 3-dimensional. This has the advantages 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.
* 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 so 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.

## 1.8 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”

## 1.9 Hardware and software requirements

Hardware requirements:

* 1.5GHz or faster processor – in order to run the game smoothly with no stuttering
* Minimum 512MB RAM
* 100MB free hard drive space – sufficient to store the game’s files
* Working keyboard and mouse – the game input is all keyboard based and the login menu needs mouse and keyboard for input
* Working monitor – to display the game

Software requirements:

* Operating System: 64-bit Windows 7 or later or OS X 10.11 or later

## 1.10 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.

## 1.11 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 |

# Chapter Two: Design

Table of Contents

[Chapter Two: Design 1](file:///H:\Computer%20Science%20Project\Report\Louis_Pattern_design.docx#_Toc116642012)

[2.1 Introduction 3](#_Toc116642013)

[2.2 Decomposition of the problem 4](#_Toc116642014)

[2.2.1 Decomposition Diagram 4](#_Toc116642015)

[2.2.2 Data Flow Diagram 5](#_Toc116642016)

[2.2.3 Input Process Output 5](#_Toc116642017)

[2.3 How All Solution Parts are Linked 6](#_Toc116642018)

[2.3.1 State Diagram of the different forms/parts 6](#_Toc116642019)

[2.3.2 How different functions /classes are connected 6](#_Toc116642020)

[2.4 Database Design 7](#_Toc116642021)

[2.4.1 Normalisation 7](#_Toc116642022)

[2.4.2 Data Dictionary 7](#_Toc116642023)

[2.4.3 Entity Relationship Diagram 7](#_Toc116642024)

[2.4.4 SQL Pseudocode 7](#_Toc116642025)

[2.5 Design of Main Parts of the Solution 8](#_Toc116642026)

[2.5.1.2 Form Design and Layout 8](#_Toc116642027)

[2.5.1.2 Justification of Validation rules 8](#_Toc116642028)

[2.5.1 .3 Algorithms and PseudoCode 8](#_Toc116642029)

[2.5.1.4 Key Variables/Data Structures /Classes 8](#_Toc116642030)

[2.5.1.5 Test Plan for PART ONE 8](#_Toc116642031)

[2.5.2.1 Form Design and Layout 9](#_Toc116642032)

[2.5.2.2 Justification of Validation rules 9](#_Toc116642033)

[2.5.2. 3 Algorithms and PseudoCode 9](#_Toc116642034)

[2.5.2.4 Key Variables/Data Structures /Classes 9](#_Toc116642035)

[2.5.1.5 Test Plan for PART TWO 9](#_Toc116642036)

[2.5.3.1 Form Design and Layout 10](#_Toc116642037)

[2.5.3.2 Justification of Validation rules 10](#_Toc116642038)

[2.5.3. 3 Algorithms and PseudoCode 10](#_Toc116642039)

[2.5.3.4 Key Variables/Data Structures /Classes 10](#_Toc116642040)

[2.5.1.5 Test Plan for PART THREE 10](#_Toc116642041)

[2.5.4.1 Form Design and Layout 11](#_Toc116642042)

[2.5.4.2 Justification of Validation rules 11](#_Toc116642043)

[2.5.4. 3 Algorithms and PseudoCode 11](#_Toc116642044)

[2.5.4.4 Key Variables/Data Structures /Classes 11](#_Toc116642045)

[2.5.1.5 Test Plan for PART FOUR 11](#_Toc116642046)

[2.6 Stakeholders involvement 12](#_Toc116642047)

[2.7 Testing plan to inform evaluation 13](#_Toc116642048)

## 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 and most liked the space theme. The game will be designed in Python, using the pygame module for the game and tkinter for the login window.   
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.



## 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 at once when testing the whole system.

### 2.2.1 Decomposition Diagram

1. Main top-down diagram:

Space Game

Login

Scoring system

Actual game

Database

1. Login diagram:

Login

Admin login window

User login window

1. Game diagram:

Actual game

Settings

2-player Gameplay

Game over screen

Main menu

Single-player Gameplay

### 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 | If logging in, checks the username exists in the database and the corresponding password is correct. If creating a user, checks the username doesn’t already exist in the database and the username is valid (is between 3 and 18 characters in length and only contains alphanumeric characters and underscores). | Error or success message |
| Password | 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 | 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 |
| Main menu navigation | If enter is pressed, the corresponding menu option happens. | Change of current menu option selected (indicated by arrows). Screen changed when an option is clicked. |
| Movement | 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 | Draws lasers fired by the 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:



For game:

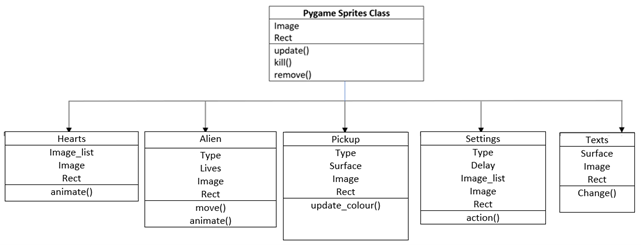


### 2.3.2 How different functions /classes are connected



Sprites class diagram:



Sprites class diagram cont.:

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.   
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.

## 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 | 36,50,98,36,49, 50 … |

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

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

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.

### 2.5.1.2 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 Justification of 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

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:



### 2.5.1.4 Key Variables/Data Structures /Classes



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.

### 2.5.1.5 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:

### 2.5.2.1 Form Design and Layout

##### Admin window



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 Justification of 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 Algorithms and PseudoCode

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

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

### 2.5.1.5 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 Main Game Design and Layout

##### Game main menu

This menu is navigated with keyboard alone rather than mouse and keyboard 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 by space or enter when it is selected



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 selected 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

// 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

//Main menu option class

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

### 2.5.3.4 Key Variables/Data Structures /Classes

|  |  |  |  |
| --- | --- | --- | --- |
| Data Item | Data Type | Validation Rule | Justification |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### 2.5.1.5 Test Plan for PART THREE

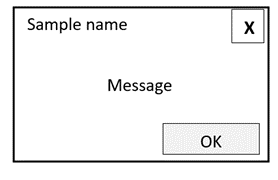
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Description | Test data | Type of data | Expected outcome | Justification |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 2.5.4 Part FOUR – 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. 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.

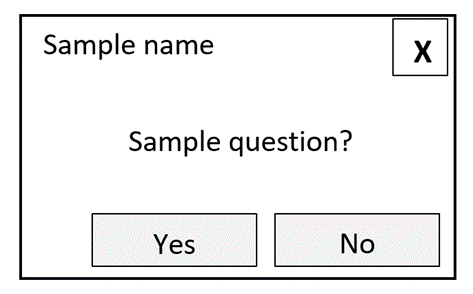
### 2.5.4.1 Form Design and Layout

I will use Tkinter to display messages.

Error message box:

Closes the message box window and returns true

Question message box:



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.

### 2.5.4.2 Algorithms and 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.

### 2.5.1.5 Test Plan for PART FOUR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |  |
| .2 | OK button clicked on info box | Clicking on ‘OK’ | Normal | Message box window closes |  |
| .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 |  |
| .4 | OK button clicked on error message box | Clicking on ‘OK’ | Normal | Message box window closes |  |
| .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 |  |
| .6 | ‘Yes’ button clicked on question message | Clicking on ‘Yes’ | Normal | Window closes and function returns true |  |
| .7 | ‘No’ button clicked on question message | Clicking on ‘No’ | Normal | Window closes and function returns true |  |

### 2.5.4 Part FIVE – 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.

### 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 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.

### 2.5.4.2 Algorithms

## 2.6 Stakeholders involvement

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

## 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. 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

Table of Contents

[Chapter Three: Development and Testing 2](file:///H:\Project\Report\Louis_Pattern_development_and_testing.docx#_Toc125545062)

[3.1 Introduction 3](#_Toc125545063)

[3.2.1 Stage 1: Building the Error Handling 4](#_Toc125545064)

[3.2.3.1 Code for Error handling section 4](#_Toc125545065)

[3.2.1.3 Feedback from Stakeholder 9](#_Toc125545066)

[3.2.1.4 Review 9](#_Toc125545067)

[3.2.1 Stage 1: Building the Database 9](#_Toc125545068)

[3.2.1.1 SQLITE + Python Code 9](#_Toc125545069)

[3.2.1.2 Testing Databases 10](#_Toc125545070)

[3.1.3 Feedback from Stakeholder (if appropriate) 10](#_Toc125545071)

[3.1.4 Review 10](#_Toc125545072)

[3.2.2 Stage 2: Building Log in 10](#_Toc125545073)

[3.2.2.1 The Code for Log in section (including validation) 10](#_Toc125545074)

[3.2.2.2 Testing Log in section (including validation) 11](#_Toc125545075)

[3.2.2.3 Feedback from Stakeholder (if appropriate) 11](#_Toc125545076)

[3.2.2.4 Review 11](#_Toc125545077)

[3.2.3 Stage 3: Building …….. PART THREE …. 11](#_Toc125545078)

[3.2.3.1 Code for …Part THREE … including validation 11](#_Toc125545079)

[3.2.3.2 Testing …. Part THREE ….including validation 11](#_Toc125545080)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 11](#_Toc125545081)

[3.2.3.4 Review 11](#_Toc125545082)

[3.2.3 Stage 4: Building …….. PART FOUR …. 12](#_Toc125545083)

[3.2.3.1 Code For … part FOUR… (including validation) 12](#_Toc125545084)

[3.2.3.2 Testing …. Part FOUR ….including validation 12](#_Toc125545085)

[3.2.3.3 Feedback from Stakeholder (if appropriate) 12](#_Toc125545086)

[3.2.3.4 Review 12](#_Toc125545087)

[3.2.4 Stage 2: Building …….. PART FIVE …. 13](#_Toc125545088)

[3.2.4.1 Code for … Part FIVE… including validation 13](#_Toc125545089)

[3.2.4.2 Testing …. Part FIVE ….including validation 13](#_Toc125545090)

[3.2.4.3 Feedback from Stakeholder (if appropriate) 13](#_Toc125545091)

[3.2.4.4 Review 13](#_Toc125545092)

[3.3 Final Review, Improvements and Corrective Actions 13](#_Toc125545093)

# Chapter Three: Development and Testing

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.

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.

When testing some parts the project, I will use unittest in order to do automatic testing rather than manual testing. This will improve the efficiency when testing and allow modules to be tested quicker.

##### 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 unit tests on each module |  |  |
| validation.py | Used anytime data must be validated | N/A | N/A |

## 3.2.1 Stage 1: Building the Error Handling

### 3.2.1.1 Code for Error handling section

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

##### Imported modules

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

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

##### Prototype 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 prototype before continuing to develop this module.

###### Prototype 1 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Test data | Expected outcome | Comment |
| 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 | Failed |

###### Prototype 1 Results:

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



##### Prototype 2

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.

###### Prototype 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Test data | Expected outcome | Comment |
| 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 | Success |
| 1.2 | Ok button clicked in the message box | Clicking on ‘OK’ | Message box closes | Success |

###### Prototype 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.

##### Prototype 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.

###### Prototype 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 |

###### Prototype 3 Results:

Test 1.3:



Test 1.4:



Test 1.5:



Test 1.6:



Test 1.7:



Test 1.8:



### 3.2.1.2 Feedback from Stakeholder

I have showed prototype 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 Prototype 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.

## 3.2.2 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.

##### Prototype 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.

###### Prototype 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.



##### Prototype 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.

###### Prototype 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 |

###### Prototype 2 Results:

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.

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.

##### Prototype 3

Functions from validation module:

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** ValueError:
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\_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

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.

###### Prototype 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.

###### Prototype 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 db. 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.

##### Prototype 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

###### Prototype 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 (including validation)

##### Imported modules

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

##### Prototype 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).

###### Prototype 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 |

###### Prototype 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.

##### Prototype 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 Prototype 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.

###### Prototype 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 |

###### Prototype 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.

##### Prototype 3

I review of the previous prototypes, there is another feature that I will add. When the user types 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()

###### Prototype 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 |
|  |  |  |  |  |

###### Prototype 3 Results:

### 3.2.3.3 Feedback from Stakeholder (if appropriate)

## 3.2.3 Stage 4: Building …….. PART FOUR ….

### 3.2.3.1 Code For … part FOUR… (including validation)

### 3.2.3.2 Testing …. Part FOUR ….including validation

### 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

Evaluation

# 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