# 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:

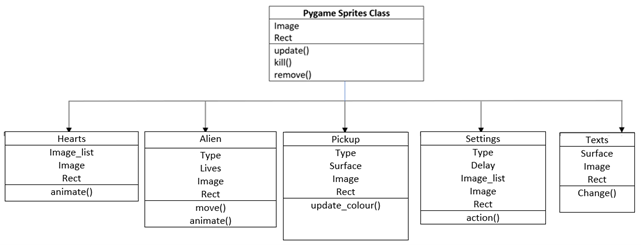


### 2.3.2 How different functions /classes are connected



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 behaves the same but there can be a large 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 security.

### 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. T

### 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 like this.

### 2.5.3.2 Justification of Validation rules

|  |  |  |  |
| --- | --- | --- | --- |
| **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

### 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

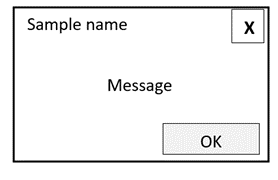
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Type of data** | **Expected outcome** | **Justification** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 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.

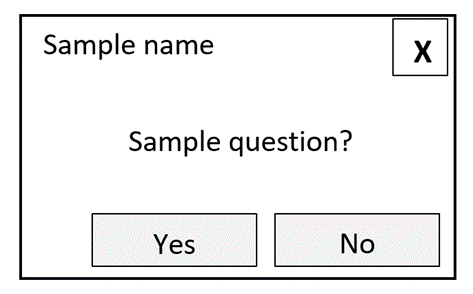
### 2.5.4.1 Form Design and Layout

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

### 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.

## 2.6 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. |