**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 10](#_Toc130127038)

[1.2 Problem Identification 10](#_Toc130127039)

[1.3 Why the problem is suited to a computational solution 11](#_Toc130127040)

[1.4 Stakeholders analysis 11](#_Toc130127041)

[1.5 Research 12](#_Toc130127042)

[Defender 12](#_Toc130127043)

[Space Invaders 14](#_Toc130127044)

[1.6 Stakeholder Consultation 16](#_Toc130127045)

[1.7 Features of the proposed solution 17](#_Toc130127046)

[1.8 The requirements of the solution 18](#_Toc130127047)

[1.9 Success Criteria 19](#_Toc130127048)

[1.10 Limitations of the solution 20](#_Toc130127049)

[1.11 Hardware and software requirements 21](#_Toc130127050)

[2.1 Introduction 23](#_Toc130127051)

[2.2 Decomposition of the problem 24](#_Toc130127052)

[2.2.1 Decomposition Diagram 24](#_Toc130127053)

[2.2.2 Data Flow Diagram 25](#_Toc130127054)

[Login data flow diagram: 25](#_Toc130127055)

[Game data flow diagram: 26](#_Toc130127056)

[2.2.3 Input Process Output 26](#_Toc130127057)

[2.3 How All Solution Parts are Linked 28](#_Toc130127058)

[2.3.1 State Diagram of the different forms/parts 28](#_Toc130127059)

[For the login system: 28](#_Toc130127060)

[For the game: 28](#_Toc130127061)

[2.3.2 Classes 29](#_Toc130127062)

[Game sprites UML class diagram: 29](#_Toc130127063)

[2.4 Database Design 30](#_Toc130127064)

[2.4.1 Normalisation 30](#_Toc130127065)

[2.4.2 Data Dictionary 30](#_Toc130127066)

[2.4.3 Entity Relationship Diagram 31](#_Toc130127067)

[2.4.4 SQL Pseudocode 31](#_Toc130127068)

[2.5 Design of Main Parts of the Solution 32](#_Toc130127069)

[2.5.1 Part ONE - Login: 32](#_Toc130127070)

[2.5.1.1 Form Design and Layout 32](#_Toc130127071)

[Login window 32](#_Toc130127072)

[2.5.1.2 Validation rules 33](#_Toc130127073)

[2.5.1.3 Algorithms and Pseudocode for Login 33](#_Toc130127074)

[Cancel button: 34](#_Toc130127075)

[Login window class: 35](#_Toc130127076)

[2.5.1.4 Test Plan for PART ONE 35](#_Toc130127077)

[2.5.2 Part TWO – Admin Panel: 36](#_Toc130127078)

[2.5.2.1 Form Design and Layout 36](#_Toc130127079)

[2.5.2.2 Validation rules 36](#_Toc130127080)

[2.5.2.3 Pseudocode for Admin Panel 37](#_Toc130127081)

[2.5.2.4 Login and Admin UML class diagram 38](#_Toc130127082)

[2.5.2.5 Test Plan for PART TWO 38](#_Toc130127083)

[2.5.3 Part THREE – Game menus: 41](#_Toc130127084)

[2.5.3.1 Game Menus Design and Layout 41](#_Toc130127085)

[Game main menu 41](#_Toc130127086)

[Game settings menu 42](#_Toc130127087)

[Highscores Menu 43](#_Toc130127088)

[2.5.3.2 Justification of Validation rules 43](#_Toc130127089)

[2.5.3.3 Algorithms and Pseudocode for Menus 44](#_Toc130127090)

[Main menu flowchart 46](#_Toc130127091)

[2.5.3.4 Key Variables/Classes 47](#_Toc130127092)

[Settings Menu 47](#_Toc130127093)

[2.5.3.5 Test Plan for PART THREE 47](#_Toc130127094)

[2.5.4 Part FOUR – Design of Main Game: 48](#_Toc130127095)

[2.5.4.1 Window Design and Layout 48](#_Toc130127096)

[2.5.4.2 Algorithms for Main Game 49](#_Toc130127097)

[Main flowchart 49](#_Toc130127098)

[Player movement flowchart 50](#_Toc130127099)

[2.5.4.3 Key Classes for Main Game 53](#_Toc130127100)

[2.5.4.4 Test Plan for Main Game 54](#_Toc130127101)

[2.5.5 Part FIVE – Error Handling 54](#_Toc130127102)

[2.5.5.1 Form Design and Layout 54](#_Toc130127103)

[2.5.5.2 Pseudocode for Error handling 55](#_Toc130127104)

[Messages module pseudocode: 55](#_Toc130127105)

[Validation module pseudocode: 56](#_Toc130127106)

[2.5.5.3 Test Plan for Error handling 58](#_Toc130127107)

[2.6 Stakeholders involvement 60](#_Toc130127108)

[Part TWO 60](#_Toc130127109)

[Revised Admin Window 60](#_Toc130127110)

[Pseudocode 61](#_Toc130127111)

[Part FOUR 61](#_Toc130127112)

[Revised Game UI 61](#_Toc130127113)

[Key classes 62](#_Toc130127114)

[2.7 Testing plan to inform evaluation 62](#_Toc130127115)

[Integration testing: 62](#_Toc130127116)

[Scenario testing: 63](#_Toc130127117)

[3.1 Introduction 65](#_Toc130127118)

[Modules 65](#_Toc130127119)

[3.2.1 Stage 1: Building General Modules 66](#_Toc130127120)

[3.2.1.1 Building General Modules (Error Handling) 66](#_Toc130127121)

[Imported modules 67](#_Toc130127122)

[Version 1 67](#_Toc130127123)

[Version 1 Testing: 67](#_Toc130127124)

[Improvement 68](#_Toc130127125)

[Version 2 Testing: 68](#_Toc130127126)

[Version 2 Results: 68](#_Toc130127127)

[Version 3 69](#_Toc130127128)

[Version 3 Testing: 69](#_Toc130127129)

[Version 3 Results: 70](#_Toc130127130)

[3.2.1.2 Feedback from Stakeholder 72](#_Toc130127131)

[3.2.1.3 Building General Modules (Validation) 72](#_Toc130127132)

[Imported modules 72](#_Toc130127133)

[Version 1 72](#_Toc130127134)

[Version 1 Testing: 73](#_Toc130127135)

[Automatic Testing 74](#_Toc130127136)

[Automatic Testing Results: 75](#_Toc130127137)

[Version 2 75](#_Toc130127138)

[Version 2 Testing: 76](#_Toc130127139)

[Automatic Testing 77](#_Toc130127140)

[Automatic Testing Results: 78](#_Toc130127141)

[Version 3 79](#_Toc130127142)

[Version 3 Testing: 79](#_Toc130127143)

[Version 3 Results: 80](#_Toc130127144)

[3.2.2 Stage 2: Building the Database 80](#_Toc130127145)

[3.2.2.1 SQLITE + Python Code 80](#_Toc130127146)

[Imported modules 80](#_Toc130127147)

[Version 1 81](#_Toc130127148)

[Version 1 Testing: 81](#_Toc130127149)

[Version 2 82](#_Toc130127150)

[Version 2 Testing: 83](#_Toc130127151)

[Automatic Testing 85](#_Toc130127152)

[Version 2 Results: 85](#_Toc130127153)

[Version 3 Testing: 87](#_Toc130127154)

[Version 3 Results: 88](#_Toc130127155)

[3.2.2.2 Feedback from Stakeholders 89](#_Toc130127156)

[3.2.2.3 Review 89](#_Toc130127157)

[Version 4 90](#_Toc130127158)

[Version 4 Results: 91](#_Toc130127159)

[3.2.3 Stage 3: Building Login 91](#_Toc130127160)

[3.2.3.1 Code for Login 91](#_Toc130127161)

[Imported modules 91](#_Toc130127162)

[Version 1 91](#_Toc130127163)

[Version 1 Testing: 93](#_Toc130127164)

[Version 1 Results: 93](#_Toc130127165)

[Version 2 96](#_Toc130127166)

[Version 2 Testing: 97](#_Toc130127167)

[Version 2 Results: 98](#_Toc130127168)

[Version 3 99](#_Toc130127169)

[Version 3 Testing: 100](#_Toc130127170)

[Version 3 Results: 101](#_Toc130127171)

[3.2.3.2 Feedback from Stakeholders 104](#_Toc130127172)

[Version 4 105](#_Toc130127173)

[Version 4 Testing: 107](#_Toc130127174)

[Version 4 Results: 108](#_Toc130127175)

[3.2.3.3 Code for Admin 110](#_Toc130127176)

[Imported modules 110](#_Toc130127177)

[Version 1 110](#_Toc130127178)

[Version 1 Testing: 112](#_Toc130127179)

[Version 1 Results: 113](#_Toc130127180)

[Version 2 117](#_Toc130127181)

[Version 2 Testing: 119](#_Toc130127182)

[Version 2 Results: 120](#_Toc130127183)

[3.2.4 Stage 4: Building Main Game 121](#_Toc130127184)

[3.2.4.1 Building the Single Player Mode 121](#_Toc130127185)

[Imported modules 121](#_Toc130127186)

[Version 1 122](#_Toc130127187)

[Version 1 Testing: 123](#_Toc130127188)

[Version 1 Results: 123](#_Toc130127189)

[Version 2 124](#_Toc130127190)

[Version 2 Testing: 126](#_Toc130127191)

[Version 2 Results: 127](#_Toc130127192)

[Version 3 128](#_Toc130127193)

[Version 3 Testing: 131](#_Toc130127194)

[Version 3 Results: 132](#_Toc130127195)

[Version 4 134](#_Toc130127196)

[Version 4 Testing: 139](#_Toc130127197)

[Version 4 Results: 140](#_Toc130127198)

[Version 5 141](#_Toc130127199)

[Version 5 Testing: 145](#_Toc130127200)

[Version 5 Results: 145](#_Toc130127201)

[Version 6 146](#_Toc130127202)

[Version 6 Testing: 151](#_Toc130127203)

[Version 6 Results: 153](#_Toc130127204)

[3.2.4.2 Feedback from Stakeholders on Single Player Mode 154](#_Toc130127205)

[Version 7 154](#_Toc130127206)

[Version 7 Testing: 157](#_Toc130127207)

[Version 7 Results: 158](#_Toc130127208)

[Version 8 158](#_Toc130127209)

[Version 8 Testing: 159](#_Toc130127210)

[Version 8 Results: 160](#_Toc130127211)

[Version 9 160](#_Toc130127212)

[Version 9 Testing: 165](#_Toc130127213)

[Version 9 Results: 166](#_Toc130127214)

[Version 10 167](#_Toc130127215)

[Version 10 Testing: 173](#_Toc130127216)

[Version 10 Results 175](#_Toc130127217)

[Version 11 176](#_Toc130127218)

[Version 11 Testing: 178](#_Toc130127219)

[Version 11 Results: 179](#_Toc130127220)

[Version 12 180](#_Toc130127221)

[Version 12 Testing 188](#_Toc130127222)

[Version 12 Results: 188](#_Toc130127223)

[3.2.4.3 Building the Two-Player Mode 189](#_Toc130127224)

[Version 1 189](#_Toc130127225)

[Version 1 Testing: 197](#_Toc130127226)

[Version 1 Results: 199](#_Toc130127227)

[Version 2 200](#_Toc130127228)

[Version 2 Testing: 200](#_Toc130127229)

[Version 2 Results: 201](#_Toc130127230)

[3.2.4.4 Feedback from Stakeholders on Two-Player Mode 201](#_Toc130127231)

[Version 3 201](#_Toc130127232)

[Version 3 Testing: 203](#_Toc130127233)

[Version 3 Results: 204](#_Toc130127234)

[3.2.5 Stage 5: Building the Scoring System 204](#_Toc130127235)

[Code for the Scoring System 205](#_Toc130127236)

[Imported modules 205](#_Toc130127237)

[Version 1 205](#_Toc130127238)

[Version 1 Testing: 206](#_Toc130127239)

[Version 1 Results: 207](#_Toc130127240)

[Version 2 208](#_Toc130127241)

[Version 2 Testing: 210](#_Toc130127242)

[Version 2 Results: 210](#_Toc130127243)

[3.2.6 Stage 6: Building the Game Menus 211](#_Toc130127244)

[Code for Game Menus 211](#_Toc130127245)

[Version 1 211](#_Toc130127246)

[3.2.4.1 Code for … Part FIVE… including validation 211](#_Toc130127247)

[3.2.4.2 Testing …. Part FIVE ….including validation 211](#_Toc130127248)

[3.2.4.3 Feedback from Stakeholder (if appropriate) 212](#_Toc130127249)

[3.2.4.4 Review 212](#_Toc130127250)

[3.3 Final Review, Improvements and Corrective Actions 212](#_Toc130127251)

[4.1 Introduction 215](#_Toc130127252)

[4.2 Testing to inform evaluation 215](#_Toc130127253)

[4.2.1 Testing 215](#_Toc130127254)

[4.2.2 Feedback from Stakeholders 216](#_Toc130127255)

[4.3 Evaluation 217](#_Toc130127256)

[4.4 Evaluating usability features 217](#_Toc130127257)

[4.5 Limitations and Maintenance 217](#_Toc130127258)

Chapter One: Analysis of the problem

## 1.1 Introduction

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

## 1.2 Problem Identification

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

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

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

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

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

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

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

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

## 1.4 Stakeholders analysis

My stakeholders for this project will be, a group of 20 college and high school students. I will give each of my stakeholders a rundown on the proposed features of the game and interview them for feedback and criticism. If any of the stakeholders suggest a way to improve the game or a new feature to add to the game, I will try to implement it.

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

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

## 1.5 Research

### Defender

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



Fig. 1 - Defender Gameplay

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

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

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



Fig. 2 – Defender Scoring System

**Disadvantages:**

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

### Space Invaders

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

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



*Fig. 3 – Orignal Space invaders game*

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

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

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

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

**Limitations:**

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

## 1.6 Stakeholder Consultation

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

**Interview with College Student First Year - 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 College Student Second Year – Kurk Milo**

* **Have you ever played a videogame?**“I played some 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 green for the enemies”

Francis Kip

Kurk Milo

Patrice Hayes

Lucas Jones

## 1.7 Features of the proposed solution

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

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

## 1.8 The requirements of the solution

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

**Input requirements**

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

**Process requirements**

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

**Output requirements**

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

**Storage requirements**

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

## 1.9 Success Criteria

|  |  |  |
| --- | --- | --- |
| **Requirement** | **What this success means** | **Evidence** |
| Login screen before the game is played | A login window where the user can enter their username and password. If they are correct, it will login the user to the game. The username and password will be stored in a database that is local to the system. The password will be hashed. | Video of the login window with a correct username and password being entered. |
| User can customise the login window | The user should be able to change the colour of the login window and this preferred option should be saved. | Video of login window being customised. |
| Intuitive UI | The user can utilise the window without needing prior knowledge or guidance. Buttons should be clearly labelled. | Questioning and obtaining feedback from stakeholders after they have used the UI |
| Simple design | Buttons and text should not be too small, the colour scheme should also be appropriate. | Screenshots of the login window and game menus |
| Main menu for the game | A number of options that can be selected with the keyboard alone.  It should be clear which option the user is selecting. | Screenshot of the main menu of the game |
| Settings screen for the game | A settings menu where the user can change the window resolution, difficulty, audio volume and colourblind mode.  The currently active settings should be easily visible. | Screenshot of the setting menu of the game |
| Window size is changeable | When the window width and height is changed from the settings menu, the game will restart. Sprites’ size and position should scale based on window width and height. | Video of window size being changed |
| Number of lives displayed during gameplay | The HUD (heads up display) should contain a number of hearts that indicate the number of lives the player has left. In 2-player mode, each player’s number of lives must be displayed separately. | Screenshots of single player mode and 2-player mode showing different numbers of lives. |
| High-scores leader board | The top 5 scores achieved must be shown in the highscores 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 framerate at 60 frames per second. There should be no stuttering while the game is running and animations should appear smooth. | Video of game running |

## 1.10 Limitations of the solution

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

## 1.11 Hardware and software requirements

Hardware requirements:

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

Software requirements:

* Operating System: 64-bit Windows 10 version 21H2 or later – this is needed in order to run a PC and manage the files of the game. The game executable file will also only work on windows 10 because it will be compiled for this operating system.

Chapter Two: Design

## 2.1 Introduction

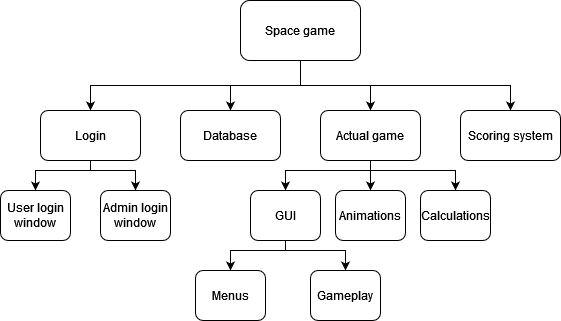
The design objectives for my game will be based upon the interviews conducted on the stakeholders and are similar to the requirements already specified. I will create a list of design requirements that will be implemented into the game. For the general design of the game, all of the stakeholders were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



## 2.2 Decomposition of the problem

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

### 2.2.1 Decomposition Diagram



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

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

### 2.2.2 Data Flow Diagram

#### Login data flow diagram:



#### Game data flow diagram:



### 2.2.3 Input Process Output

Login:

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

Actual game:

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

## 2.3 How All Solution Parts are Linked

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

#### For the login system:



**Start**

**Takes to game**

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

#### For the game:



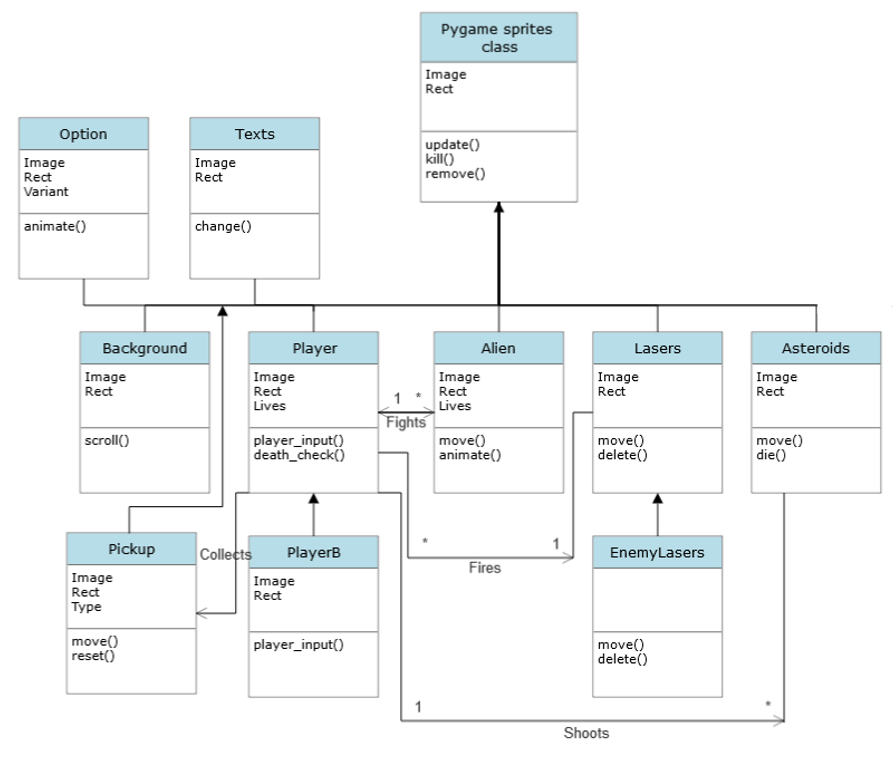
**Returns to login**

**Start**

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

### 2.3.2 Classes

#### Game sprites UML class diagram:



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

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

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

## 2.4 Database Design

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

### 2.4.1 Normalisation

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

### 2.4.2 Data Dictionary

Users table:

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

Highscores table:

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

### 2.4.3 Entity Relationship Diagram

∞

1

Highscore

User

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

### 2.4.4 SQL Pseudocode

// Creating the Users table that will store user details

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

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

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

## 2.5 Design of Main Parts of the Solution

### 2.5.1 Part ONE - Login:

I will use tkinter for the login system windows.

#### 2.5.1.1 Form Design and Layout

##### Login window



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

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

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

#### 2.5.1.2 Validation rules

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

#### 2.5.1.3 Algorithms and Pseudocode for Login

Function search(username, password, table):

// First must get data from correct table

IF table == “Admins”:

Records = execute SQL: SELECT \* FROM Admins

Else:

Records = execute SQL: SELECT \* FROM Users

ENDIF

FOR each row in records:

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

Return True

Next row

ENDFOR

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

Return False

End function

Procedure log\_in(username, password):

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

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

Play game

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

// Function to close login window and open admin window

Open admin window

ELSE:

// Error message displayed as popup box

Show error message

ENDIF

End procedure

##### Cancel button:



##### Login window class:



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

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

#### 2.5.1.4 Test Plan for PART ONE

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

### 2.5.2 Part TWO – Admin Panel:

#### 2.5.2.1 Form Design and Layout



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

Calls the delete\_user function when clicked.

Button that calls the create\_user function when clicked.

#### 2.5.2.2 Validation rules

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

#### 2.5.2.3 Pseudocode for Admin Panel

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

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

If password == confirmed\_password:

If username is valid and password is valid:

execute SQL: insert into Users (Username, Password)

return “Entered user”

end if

Else:

return “Error – Passwords do not match”

end if

end function

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

Function delete\_user(name):

IF is\_existent\_user(name):

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

Return (“Successfully deleted”)

Catch:

Return (“Error deleting user”)

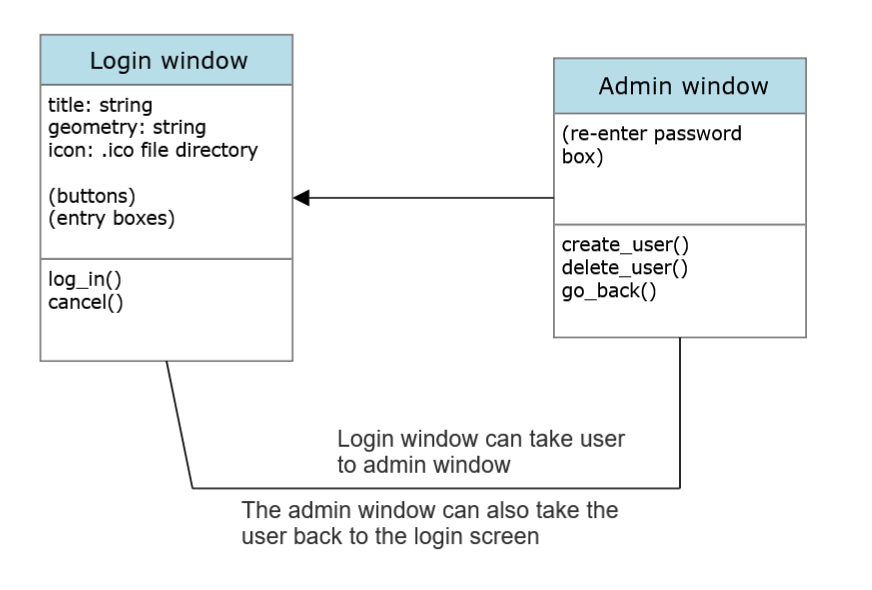
ELSE:

Return (“Error – user does not exist”)

ENDIF

End function

#### 2.5.2.4 Login and Admin UML class diagram



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

#### 2.5.2.5 Test Plan for PART TWO

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

### 2.5.3 Part THREE – Game menus:

#### 2.5.3.1 Game Menus Design and Layout

##### Game main menu

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



Starts the main game (single player)

Starts the 2-player game

Opens the settings menu

Opens the highscores menu

Currently selected option will have different animation

##### Game settings menu



Arrow indicates currently applied option

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

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

##### Highscores Menu



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

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

#### 2.5.3.2 Justification of Validation rules

Menu validation:

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

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

#### 2.5.3.3 Algorithms and Pseudocode for Menus

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

Procedure get\_scores():

Set scores to empty list

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

FOR each row in records:

Append to scores row[2]

ENDFOR

RETURN scores

End Procedure

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

Procedure get\_names():

Set scores to empty list

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

FOR each row in records:

Append to scores row[1]

ENDFOR

RETURN scores

End Procedure

// Procedure used for initially creating the highscores table

Procedure create\_h\_table():

'''CREATE TABLE IF NOT EXISTS Highscores

(ID INTEGER PRIMARY KEY AUTOINCREMENT,

Name VARCHAR NOT NULL,

Score INT NOT NULL,

Date TEXT(16) NOT NULL,

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

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

Class Option(Sprite):

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

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

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

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

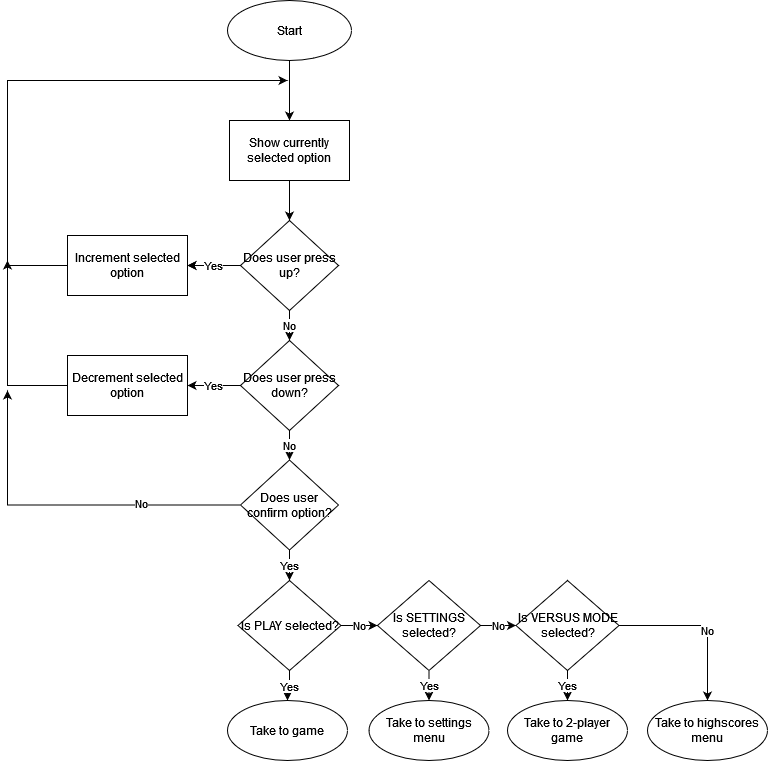
elif variant = “versus”:  
 type = “versus”  
 set image\_sprites to images for versus

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

END Procedure

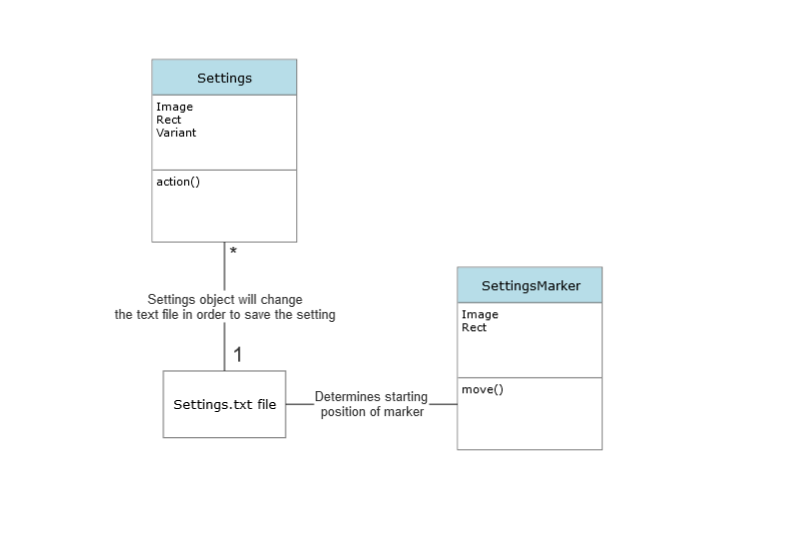
END Class

##### Main menu flowchart



#### 2.5.3.4 Key Variables/Classes

##### Settings Menu



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

#### 2.5.3.5 Test Plan for Game Menus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Justification** |
| 6.1a | Attempt to run the main menu | Running play() | The main menu should appear with the options and text visible | The main menu is a vital part of the game that will be used to navigate to different sections. It should be functional |
| 6.6a | Attempt to open settings menu | Selecting “settings” button from the game’s main menu | The settings menu should load with settings, markers and miscellaneous images displayed correctly. | The settings menu must be shown correctly because it is important that the user is able to customise the game. |
| 6.6b | Attempt to close settings menu | Pressing backspace while in settings menu | User should be taken back to viewing the main menu | The user should not have to re-launch the game to return to a previous menu |
| 6.7a | Testing moving the currently selected setting | Pressing arrow keys whilst on settings menu | The highlighting setting should move, similar to the main menu. | The user should be able to navigate the settings menu. |
| 6.7b | Testing changing a setting | Selecting the “easy” setting for difficulty | The marker should be updated to be above “easy” and the settings text file should be updated. | The marker should accurately show the user which settings is currently applied. |

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

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

#### 2.5.4.1 Window Design and Layout

Counter that displays the player’s lives

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



Player ship. Movement controlled with WASD.

Enemies that appear from the right of the screen.

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

Projectiles fired by the player with SPACE.

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

#### 2.5.4.2 Algorithms for Main Game

##### Main flowchart



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

##### Player movement flowchart

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



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

To illustrate this point:

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

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

Shape

Description automatically generated

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



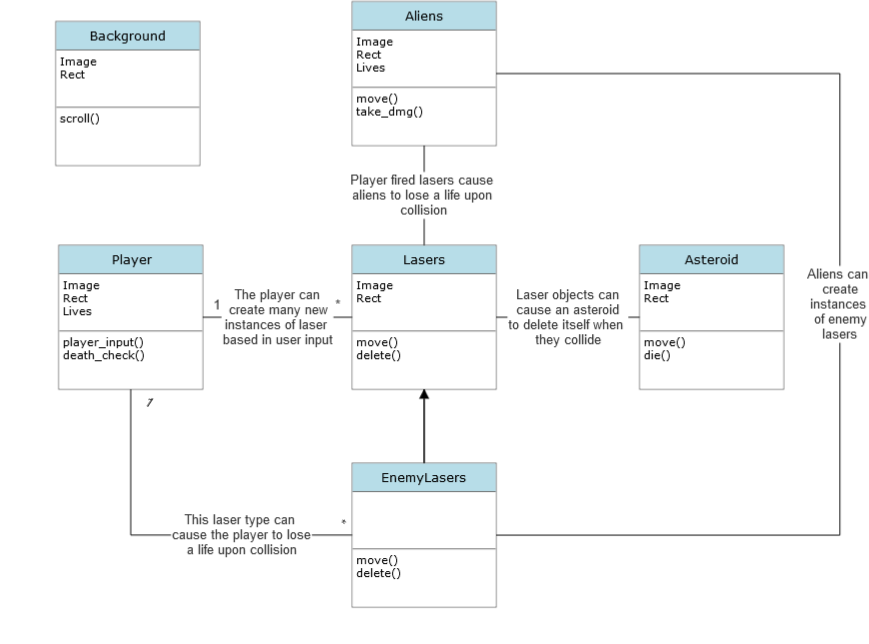
This would be achieved by considering the traingle that forms a hypotenuse of length **a**. If the distance that must be added to horizontal (and vertical) position is **x**, then:

A picture containing baseball, player

Description automatically generated

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

#### 2.5.4.3 Key Classes for Main Game



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

#### 2.5.4.4 Test Plan for Main Game

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Justification** |
| 4.1a | Running the game | Running the play() function | A blank pygame window should open and be displayed | This is an initial test to check if pygame creates the game window with no errors |
| 4.1b | Attempt to close the pygame window | Clicking on the X in the top right corner of the window | The window should close with no errors | The game should not crash upon exiting |
| 4.3a | Attempting to move the player in a single direction | While the window is open, holding each movement key individually and checking the ship moves accordingly. | The ship should move upwards when W is held, left when A is held, down when S is held, and right when D is held. The movement speed should be equal in all directions | The player movement must be tested because it is a vital part of the game. Therefore, must function as expected. |
| 4.3b | Attempting to move the player in two opposing directions. | Holding two opposite keys at once (W and S, A and D) | The player should not move as | Holding both up and down or left and right at once should not cause the player to move in any direction. |
| 4.3c | Attempting to move the player in a diagonal direction | Holding diagonal movement combinations (e.g., W and D) | The player should move in the corresponding diagonal direction at the same speed | The diagonal movement speed must be the same as the horizontal movement speed for consistency. |
| 4.7a | Attempt to fire lasers | With the game running, holding the space key | A number of laser objects should be created at the player position and move to the right | Shooting will be a main feature of the game and projectiles fired by the player must work correctly. |
| 4.7b | Attempt to shoot an alien | With the game running, pressing space multiple times when aiming at an alien | All lasers that collide with the alien should be deleted. The alien should take multiple hits before being killed. | Obviously shooting the alien enemies is the main objective of the game so this must be implemented without bugs in the final version. |

### 2.5.5 Part FIVE – Error Handling

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

#### 2.5.5.1 Form Design and Layout

I will use Tkinter to display messages.

Diagram

Description automatically generatedError message box:

Closes the message box window and returns true

Question message box:

Graphical user interface, diagram, application

Description automatically generated

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

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

#### 2.5.5.2 Pseudocode for Error handling

##### Messages module pseudocode:

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

show info box (title, message)

elif opt == 2:

show error message (title, message)

elif opt == 3:

show warning message (title, message)

else:

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

End if

End Procedure

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

##### Validation module pseudocode:

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

Function is\_length(data, length, opt)

TRY

IF opt == 1 THEN

IF length of data == length THEN

RETURN True

ELSE

RETURN False

ELSE IF opt == 2 THEN

IF length of data >= length THEN

RETURN True

ELSE

RETURN False

ELSE IF opt == 3 THEN

IF length of data <= length THEN

RETURN True

ELSE

RETURN False

END IF

EXCEPT Exception AS ex:

RETURN ex

END TRY

END FUNCTION

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

Function is\_inrange(data, lo, hi)

TRY

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

RETURN True

ELSE

RETURN False

ENDIF

EXCEPT ValueError

RETURN "Error 2"

ENDTRY

ENDFUNCTION

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

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

IF opt == "username" THEN

IF u is a string THEN

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

SET u = u in uppercase

FOR n in range(length of u) DO

IF u[n] is not valid character THEN

SET valid = False

ENDIF

ENDFOR

ELSE

SET valid = False

ENDIF

ELSE

SET valid = False

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

SET valid = True

ENDIF

ENDIF

RETURN valid

ENDFUNCTION

#### 2.5.5.3 Test Plan for Error handling

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

## 2.6 Stakeholders involvement

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

### Part TWO

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

#### Revised Admin Window





Calls the reset\_scores() function when clicked

Goes back the login menu when clicked

Opens the dropdown menu when clicked

#### Pseudocode

Procedure reset\_scores():

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

create\_h\_table()

End Procedure

### Part FOUR

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

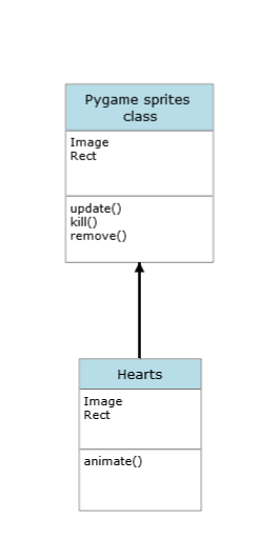
#### Revised Game UI



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

#### Key classes

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

****

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

## 2.7 Testing plan to inform evaluation

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

### Integration testing:

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

### Scenario testing:

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

Development and Testing

Chapter Three: Development and Testing

## 3.1 Introduction

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

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

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

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

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

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

### Modules

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

## 3.2.1 Stage 1: Building General Modules

### 3.2.1.1 Building General Modules (Error Handling)

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

#### Imported modules

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

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

#### Version 1

Show\_message procedure:

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

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

##### Version 1 Testing:

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

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



#### Improvement

Show\_message procedure (fixed):

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

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

##### Version 2 Testing:

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

##### Version 2 Results:

Test 1.1 was successful as shown in this evidence:



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

#### Version 3

Show\_message procedure (improved)

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

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

##### Version 3 Testing:

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

##### Version 3 Results:

Test 1.3:



Test 1.4:



Test 1.5:



Test 1.6:



Test 1.7:

Video 1.7

Test 1.8:

Video 1.8

### 3.2.1.2 Feedback from Stakeholder

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

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

### 3.2.1.3 Building General Modules (Validation)

#### Imported modules

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

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

#### Version 1

Is\_inrange() function:

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

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

Is\_length function:

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

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

##### Version 1 Testing:

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

##### Automatic Testing

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

**Unittest (testing.py):**

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

**TestMethods() class:**

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

##### Automatic Testing Results:



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

#### Version 2

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

Is\_valid\_date function:

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

Is\_valid\_user() function:

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

##### Version 2 Testing:

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

##### Automatic Testing

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

TestMethods class:

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

##### Automatic Testing Results:



As evident, the automatic testing for Version 2 was successful, indicating that tests 2.1 – 2.10 returned the expected values.

#### Version 3

This version will simply be adding one function that will be used to validate a score. This will be used later on in the development of the project, in the scoring system.

Is\_valid\_score() function:

1. *# Checks if a high score is valid - positive integer that isn't too high*
2. *# 0 is still a valid score*
3. **def** is\_valid\_score(score):
4. **if** isinstance(score, int):
5. **if 0 <= score < 1000000:**
6. **return** True
7. **return** False

##### Version 3 Testing:

**Automatic Testing**

I have decided to once again, use automatic testing for this section because the function is extremely simple and unittest will allow me to test a wide range of test data to ensure the code is robust.

1. **class** TestMethods(unittest.TestCase):
2. *# -----------------------------------------------------------------------------*
3. *# TESTING VALIDATION*
5. ***# testing scoes***
6. **def** test\_valid\_scores(self):
7. *# Valid scores*
8. **for** i **in** range(100):
9. self.assertTrue(is\_valid\_score(100\*i))
11. *# Invalid scores*
12. **for** i **in** range(1,100):
13. self.assertFalse(is\_valid\_score(-100\*i))
15. ***# Invalid score***
16. self.assertFalse(is\_valid\_score(500.5))
17. *# Invalid score*
18. self.assertFalse(is\_valid\_score("Test"))
19. *# Borderline*
20. **self.assertTrue(is\_valid\_score(999999))**

This unittest will check that is\_valid\_score returns true for positive integers that should be seen as valid. It then checks that a range of negative number are marked invalid with a second for loop. The third test case tests an invalid decimal score of 500.5.   
The fourth test case tests an invalid score of "Test". This is invalid because it is a string, not an integer. The fifth test case tests a borderline valid score of 999999.

##### Version 3 Results:

Shape

Description automatically generated with low confidence

As shown in this evidence, all test cases for the automatic testing of this function have passed.

This section of the project is now fully functioning and tested, meaning I will now move on to developing and testing the next stage – the database.

## 3.2.2 Stage 2: Building the Database

### 3.2.2.1 SQLITE + Python Code

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

#### Imported modules

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

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

#### Version 1

Create\_table() function:

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

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

##### Version 1 Testing:

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



#### Version 2

Enter\_user() function:

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

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

Search() function:

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

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

##### Version 2 Testing:

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

Test 2.1:



Test 2.2:



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

#### Automatic Testing

**Unittest (testing.py):**

**I have also used unittest to test this module because it will allow me to test these functions with a wide range of input data relatively quickly compared to manual testing. Testing with both valid and invalid will allow me to check if the function always returns the expected result.**

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

#### Version 2 Results:

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

Text

Description automatically generated

Functions from LoginData module:

Enter\_user() function (improved):

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

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

Delete\_user() procedure:

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

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

#### Version 3 Testing:

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

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

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

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

#### Version 3 Results:

2.8, 2.9:



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



### 3.2.2.2 Feedback from Stakeholders

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

### 3.2.2.3 Review

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



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

#### Version 4

Enter\_user function (fixed):

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

Search function (fixed):

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

##### Version 4 Results:



The password is now successfully hashed in the database.

## 3.2.3 Stage 3: Building Login

### 3.2.3.1 Code for Login

#### Imported modules

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

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

#### Version 1

LoginWindow() class:

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

create\_window function:

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

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

##### Version 1 Testing:

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

##### Version 1 Results:

3.1:



3.3:



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

3.4



3.5:



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

#### Version 2

LoginWindow() class (fixed):

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

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

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

##### Version 2 Testing:

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

##### Version 2 Results:

3.3:

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

3.5:



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

#### Version 3

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

LoginWindow() Class (Improved):

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

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

##### Version 3 Testing:

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

##### Version 3 Results:

3.1:

Graphical user interface, text, application, email

Description automatically generated

3.4:

Graphical user interface, application

Description automatically generated

3.7:

Graphical user interface, application

Description automatically generated

3.8:

Graphical user interface, text, application

Description automatically generated

From the above screenshots, it is clear to see that all of the features that were implemented in the third build of the login system have been done so successfully. The show password box works as intended, with the password being hidden when the box is unchecked and shown when the box is ticked.

### 3.2.3.2 Feedback from Stakeholders

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

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

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

#### Version 4

Additional imported modules:

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

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

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

LoginWindow() Class (Improved):

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

##### Version 4 Testing:

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

##### Version 4 Results:

3.1:



3.9a:

Video 3.9a

3.9b:

Video 3.9b

3.9c:

Video 3.9c

3.10:

Video 3.10

3.11:

Graphical user interface, text

Description automatically generated

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

### 3.2.3.3 Code for Admin

#### Imported modules

1. **from** tkinter **import** ttk
2. **from** tkinter **import** Menu
3. **from** messages **import** \*
4. **import** LoginData
5. **import login**

Login has been imported here in order for the admin window class to inherit from the login window class. This is done because the two windows are similar and this will save me writing some code twice. As usual, messages has been imported in order to display error and info messages using the show\_message() function.  
LoginData has been imported in order to use its functions that allow entering and removal of users from the database.

#### Version 1

AdminWindow() class:

1. **class** AdminWindow(login.LoginWindow):
2. **def** \_\_init\_\_(self):
3. super().\_\_init\_\_()
4. self.menubar = Menu(self)
5. **self.title("Admin Window")**
6. self.config(bg='#CDE2FF')
7. self.geometry("600x360")
8. self.resizable(False, False)
9. self.eval('tk::PlaceWindow . center')
10. ***# title***
11. self.label = ttk.Label(self, text='Welcome admin', font=("Helvetica", 25, "bold"))
12. self.label.pack()
14. *# text entry boxes*
15. **self.entry1 = tk.Entry(self, bd=6, width=40)**
16. self.entry1.place(x=250, y=100)
17. self.entry2 = tk.Entry(self, bd=6, width=40)
18. self.entry2.place(x=250, y=150)
19. self.entry2.config(show="\*")
20. **self.entry3 = tk.Entry(self, bd=6, width=40)**
21. self.entry3.place(x=250, y=180)
22. self.entry3.config(show="\*")
23. self.hidden = True
25. ***# create user button***
26. self.button = ttk.Button(self, text='Create User')
27. self.button['command'] = **lambda**: self.create\_user()
28. self.button.place(x=475, y=300)
29. self.bind("<Return>", (**lambda** event: self.create\_user()))
31. *# delete user button*
32. self.button = ttk.Button(self, text='Delete User')
33. self.button['command'] = **lambda**: self.delete\_user()
34. self.button.place(x=50, y=300)
36. *# username and password text*
37. self.label = ttk.Label(self, text='New User Name:', font=("Arial", 15))
38. self.label.place(x=75, y=100)
39. self.label = ttk.Label(self, text='Password:', font=("Arial", 15))
40. **self.label.place(x=75, y=150)**
41. self.label = ttk.Label(self, text='Re-enter password', font=("Arial", 15))
42. self.label.place(x=75, y=180)
44. *# show password checkbox*
45. **self.check1 = tk.Checkbutton(self, text='Show Password', onvalue=True, offvalue=False)**
46. self.check1['command'] = **lambda**: self.togglepass()
47. self.check1.place(x=420, y=220)
49. **def** create\_user(self):
50. **username = self.entry1.get()**
51. password = self.entry2.get()
52. confirm = self.entry3.get()
54. *# Checks the passwords match before attempting to create user*
55. **if password == confirm:**
56. **if** LoginData.enter\_user(username, password):
57. show\_message("Success", "Entered user", 1)
58. **else**:
59. show\_message("Error", "Invalid user", 3)
60. **else:**
61. show\_message("Error", "Passwords do not match", 3)
63. **def** delete\_user(self):
64. username = self.entry1.get()
65. **LoginData.delete\_user(username)**
67. **def** go\_back(self):
68. AdminWindow.destroy(self)
69. login.create\_window()

This class functions and appears similar to the login window, but has the buttons “create user” and “delete user” instead of “login” and “cancel”. These buttons have corresponding methods that are called when they are clicked. This should allow an administrator to create a new user by inputting the new username and password, as well as the confirm password. The password must match the confirm password in order for a new user to be inserted into the database. An appropriate error message will display if this is not the case.

Create\_window() procedure:

1. **def** create\_window():
2. admin\_login = AdminWindow()
3. admin\_login.mainloop()

##### Version 1 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 3.2.1 | Attempt to display the admin window | Running create\_window() | The admin window should appear with widgets in the correct places | Partial success |
| 3.2.2a | Attempt to create a user with fully valid details | Clicking “create user” with Username = “test\_user2” Password = “testpassword2” Re-enter password = “testpassword2” | A confirmation message should appear, and a new user should be added to the Users table of the database |  |
| 3.2.2b | Attempt to create a user with valid details, but password doesn’t match | Clicking “create user” with Username = “test\_user3” Password = “testpassword3” Re-enter password = “” | An error message should appear saying passwords do not match. User should not be entered to the database. |  |
| 3.2.2c | Attempt to create a user with invalid details and non-matching password | Clicking “create user” with Username = “test” Password = “test” Re-enter password = “” | Passwords don’t match message should appear, and the user should not be entered to the database. |  |
| 3.2.2d | Attempt to create a user with invalid details and matching passwords | Clicking “create user” with Username = “test” Password = “test” Re-enter password = “test” | Invalid details message should appear, and the user should not be entered to the database. |  |
| 3.2.3a | Attempt to delete an existent user | Clicking “delete user” with Username = “test\_user2” Password = “testpassword2” | The user deleted message should appear and this user should be removed from the database. |  |
| 3.2.3b | Attempt to delete a non-existent user | Clicking “delete user” with all boxes blank | “user does not exist” message should appear |  |

##### Version 1 Results:

3.2.1:

Graphical user interface, application

Description automatically generated

The buttons and text boxes appear as expected, but I forgot to include the admin dropdown menu, which will allow them to go back to the user login window. This will be addressed in the next version.

3.2.2a:

Graphical user interface, application

Description automatically generated

3.2.2b:

Graphical user interface, application

Description automatically generated

3.2.2c

Graphical user interface, application

Description automatically generated

3.2.2d:

Graphical user interface, application

Description automatically generated

3.2.3a:

Graphical user interface, application

Description automatically generated

3.2.3b:

Graphical user interface, application, Word

Description automatically generated

As shown in these screenshots, the AdminWindow() attributes and methods are functioning as expected. A user can only be entered into the database when both the username and password are valid and the re-entered password matches the original password. This is followed by a success message appearing.  
When passwords do not match, the “passwords don’t match” error message appears and nothing is added to the database.  
When the passwords match but the username or password is invalid, the “invalid details” error message will show. As before, no new user is inserted into the database. Other than the lack of the options menu as shown in the improved design in the design section, this build of the admin window is successful.

#### Version 2

Reset\_hs() procedure:

1. **def** reset\_hs():
2. *# Reset scores function to go here. This will be completed later.*
3. show\_message("Done", "Highscores reset!", 1)

This is a small placeholder function which will later be used to reset the scores table in the database. This must be updated once the scoring system for the game is complete, which will be done simply by changing the comment to a function from the HighscoresData module that will reset the scores table of the database to the default values.

AdminWindow() class (improved):

1. **class** AdminWindow(login.LoginWindow):
2. **def** \_\_init\_\_(self):
3. super().\_\_init\_\_()
4. self.menubar = Menu(self)
5. **self.title("Admin Window")**
6. self.config(bg='#CDE2FF')
7. self.geometry("600x360")
8. self.resizable(False, False)
9. self.eval('tk::PlaceWindow . center')
10. ***# title***
11. self.label = ttk.Label(self, text='Welcome admin', font=("Helvetica", 25, "bold"))
12. self.label.pack()
14. *# text entry boxes*
15. **self.entry1 = tk.Entry(self, bd=6, width=40)**
16. self.entry1.place(x=250, y=100)
17. self.entry2 = tk.Entry(self, bd=6, width=40)
18. self.entry2.place(x=250, y=150)
19. self.entry2.config(show="\*")
20. **self.entry3 = tk.Entry(self, bd=6, width=40)**
21. self.entry3.place(x=250, y=180)
22. self.entry3.config(show="\*")
23. self.hidden = True
25. ***# create user button***
26. self.button = ttk.Button(self, text='Create User')
27. self.button['command'] = **lambda**: self.create\_user()
28. self.button.place(x=475, y=300)
29. self.bind("<Return>", (**lambda** event: self.create\_user()))
31. *# delete user button*
32. self.button = ttk.Button(self, text='Delete User')
33. self.button['command'] = **lambda**: self.delete\_user()
34. self.button.place(x=50, y=300)
36. *# exit dropdown menu*
37. self.config(menu=self.menubar)
38. file\_menu = Menu(self.menubar)
39. file\_menu.add\_command(label='User Menu', command=self.go\_back)
40. **file\_menu.add\_command(label='Reset scores', command=reset\_hs)**
41. file\_menu.add\_command(label='Quit', command=self.destroy)
43. self.menubar.add\_cascade(label="File", menu=file\_menu, underline=0)
45. ***# username and password text***
46. self.label = ttk.Label(self, text='New User Name:', font=("Arial", 15))
47. self.label.place(x=75, y=100)
48. self.label = ttk.Label(self, text='Password:', font=("Arial", 15))
49. self.label.place(x=75, y=150)
50. **self.label = ttk.Label(self, text='Re-enter password', font=("Arial", 15))**
51. self.label.place(x=75, y=180)
53. *# show password checkbox*
54. self.check1 = tk.Checkbutton(self, text='Show Password', onvalue=True, offvalue=False)
55. **self.check1['command'] = lambda: self.togglepass()**
56. self.check1.place(x=420, y=220)
58. **def** create\_user(self):
59. username = self.entry1.get()
60. **password = self.entry2.get()**
61. confirm = self.entry3.get()
63. *# Checks the passwords match before attempting to create user*
64. **if** password == confirm:
65. **if LoginData.enter\_user(username, password):**
66. show\_message("Success", "Entered user", 1)
67. **else**:
68. show\_message("Error", "Invalid user", 3)
69. **else**:
70. **show\_message("Error", "Passwords do not match", 3)**
72. **def** delete\_user(self):
73. username = self.entry1.get()
74. LoginData.delete\_user(username)
76. **def** go\_back(self):
77. AdminWindow.destroy(self)
78. login.create\_window()

##### Version 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 3.2.1 | Attempt to display the admin window | Running create\_window() | The admin window should appear with widgets in the correct places | Success |
| 3.2.4 | Attempt to go back to the user login menu | Clicking the file dropdown menu and selecting “user menu” | The admin window should close and the user login window should appear |  |
| 3.2.5 | Attempt to ‘reset the scores’ | Clicking the file dropdown menu and selecting “reset scores” | The reset scores message should appear |  |
| 3.2.6 | Attempt to quit using the dropdown menu | Clicking the file dropdown menu and selecting “quit” | The window should close |  |

##### Version 2 Results:

3.2.1:

Graphical user interface, application

Description automatically generated

3.2.4:

Video 3.2.4

As expected, this button immediately changes the state back to the login window. From here, the login window behaves as normal and it is possible for a user to sign in.

3.2.5:

Video 3.2.5

While the scores system hasn’t been implemented yet, this shows that the reset score button successfully called the correct function because the “highscores reset” message appears. When I complete development of the scoring system, this will reset the scores table of the database.

3.2.6:

Video 3.2.6

The window closes and the program exits when the quit option is clicked as expected. With all of the options for the admin dropdown menu tested and proved to be working as intended, the development of the admin panel is complete.

This now means that the entire login section for the project is complete, and the development of the game section can begin.

## 3.2.4 Stage 4: Building Main Game

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

### 3.2.4.1 Building the Single Player Mode

I will start by developing the single player mode of the game. By first programming the key aspects for this main part, it will allow me to later reuse assets, classes, and functions in the two-player section. This is because the two modes will be similar, but with some key differences that will be applied when I begin developing the two-player mode.

#### Imported modules

Game.py:

1. **from** sys **import** exit
3. **import** login
4. **from** messages **import** \*
5. **import os**
7. **from** sprites **import** \*

Sprites.py:

1. **import** pygame
2. **import** random

For the imports, I have imported exit from sys in order to allow the pygame window to be closed easily. An X in the top right corner will close the window. I have imported login because later on I will use it to allow the user to go back to the login section from the game. Importing sprites allows me to create objects from the classes that I will write in it in the main game loop, allowing me to use them.   
For the game, the random library will be used to determine random starting positions for some enemies.

#### Version 1

Play procedure:

1. **def** play():
2. play\_game = True
3. pygame.init()
4. screen = pygame.display.set\_mode((1000, 600))
5. **pygame.display.set\_caption("Space Game")**
6. clock = pygame.time.Clock()
7. **while** play\_game: *# Game loop*
8. **for** event **in** pygame.event.get():
9. **if** event.type == pygame.QUIT:
10. **pygame.quit()**
11. *# Update everything*
12. pygame.display.update()
13. clock.tick(60) *# Caps at 60 fps*

This is the first build for the play procedure. This procedure contains the main game loop, which will be used by pygame to repeatedly update every object. It will also display the sprite surfaces on the screen. For now, I have not added any objects in order to check that the window runs and displays correctly and can be closed when needed. During each iteration of the loop, the code checks for events (such as a user quitting the game) using the pygame.event.get() method. I have also included a clock object in order to limit the game’s frame rate because how fast the game will run is tied to frame rate so this should be kept constant.

##### Version 1 Testing:

For the testing of the game, I have decided to use manual testing, because automatic testing using unittest is not possible.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.1a | Running the play() function | N/A | A blank pygame window should open and be displayed |  |
| 4.1b | Attempt to close the pygame window | Clicking on the X in the top right corner of the window | The window should close with no errors | Failed |

##### Version 1 Results:

4.1a:

Shape, rectangle

Description automatically generated

The pygame window displayed correctly.

4.1b:

Text

Description automatically generated

However, despite the window closing correctly, the program causes an error upon exit. Because this causes a crash, this should be patched immediately in order to avoid problems later on with closing the game in order to return to the login window.  
This error occurs because the code quits pygame on line 10, but since the code is still running it attempts to use the pygame function update() on line 12. This can easily be solved by exiting the entire function after quitting pygame.

#### Version 2

For this version, I fixed the game loop and also added the Player() class in order to start developing the actual game. I will first code player movement and later implement the other features specified in the design (lives, shooting, taking damage, etc.).

Play() procedure (fixed):

1. **def** play():
2. play\_game = True
3. pygame.init()
4. screen = pygame.display.set\_mode((1000, 600))
5. **pygame.display.set\_caption("Space Game")**
6. clock = pygame.time.Clock()
7. player = pygame.sprite.GroupSingle()
8. player.add(Player(1000, 600))
10. **while play\_game: *# Game loop***
11. **for** event **in** pygame.event.get():
12. **if** event.type == pygame.QUIT:
13. pygame.quit()
14. exit()
15. ***# Update everything***
16. screen.fill("black")
17. player.draw(screen)
18. player.update()
19. pygame.display.update()
20. **clock.tick(60) *# Caps at 60 fps***

This procedure has now been updated to exit after pygame is quit.   
The game loop now updates the screen, clears it with a black colour before drawing the player sprite on the screen. The player and display are updated every iteration.

Player() class:

1. *# Spaceship class controlled by the user*
2. **class** Player(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, wd, ht):
4. super().\_\_init\_\_()
5. **self.wd = wd**
6. self.ht = ht
7. self.image\_sprite = pygame.image.load("graphics/ship1.png")
8. self.image = pygame.transform.scale(self.image\_sprite, (wd/11, ht/11))
9. self.position = pygame.math.Vector2(0, 0)
10. **self.rect = self.image.get\_rect()**
12. **def** player\_input(self): *# Ship movement from input*
13. keys = pygame.key.get\_pressed()
14. dx = keys[pygame.K\_d] - keys[pygame.K\_a]
15. **dy = keys[pygame.K\_s] - keys[pygame.K\_w]**
16. self.direction = pygame.math.Vector2(dx, dy)
17. *# Accounting for diagonal speed by dividing by root 2*
18. **if** dx != 0 **and** dy != 0:
19. self.direction /= 1.41421
21. *# Focus active when shift is held*
22. **if** keys[pygame.K\_LSHIFT]:
23. self.position += self.direction \* 4
24. **else**:
25. **self.position += self.direction \* 8**
27. *# Set rect position to position vector*
28. self.rect.x = round(self.position.x)
29. self.rect.y = round(self.position.y)

This is the class used to generate the player object in the play procedure. When initialising, it loads an image file and adjusts the size when setting it as the image for the sprite. It then sets the rect (used for positioning the object) to fit this image.  
For now, the class has only one method – player\_input(). This gets all the keys being pressed and applies a direction vector based on the WASD keys being pressed. If the player is moving in a diagonal, the movement speed on x and y has to be divided by √2 as discussed before. The position of the sprites rectangle is then updated to that of the vector self.position.

##### Version 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.1b | Attempt to close the pygame window | Clicking on the X in the top right corner of the window | The window should close with no errors |  |
| 4.2 | Running the game and checking that the player is being drawn | Run the play() procedure | The player ship should be visible in the pygame window |  |
| 4.3a | Attempting to move the player in a single direction | While the window is open, holding each movement key (W, A, S, D) individually and checking the ship moves accordingly. | The ship should move upwards when W is held, left when A is held, down when S is held, and right when D is held. The movement speed should be equal in all directions |  |
| 4.3b | Attempting to move the player in two opposing directions. | Holding two opposite keys at once (W and S, A and D) | The player should not move as moving both up + down or left + right should cancel movement |  |
| 4.3c | Attempting to move the player in a diagonal direction | Holding diagonal movement combinations (e.g., W and D) | The player should move in the corresponding diagonal direction (at the same speed as the movement in test 4.3a) |  |

##### Version 2 Results:

4.1b:

Graphical user interface

Description automatically generated with low confidence

4.2:

A screenshot of a computer

Description automatically generated with medium confidence

4.3a:

Video 4.3a

4.3b:

Video 4.3b

4.3c:

Video 4.3c

A feature that doesn’t yet exist in the program is the player does not interact with the sides of the screen. The ship will just pass through the edge of the window and go out of view. This will be updated in the next version where the player won’t be able to continue moving in the corresponding direction if they are positioned near the edge of the screen.

#### Version 3

For this version I will add the Lasers and Alien classes and allow these to interact with each other. The lasers will be generated by the player’s input and will cause an alien object to decrease its remaining lives by one. When an alien reaches 0 lives it will be killed.

Lasers() class:

1. **class** Lasers(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, x, y, wd, ht):
3. super().\_\_init\_\_()
4. self.wd = wd
5. **self.ht = ht**
6. self.surface = pygame.image.load("graphics/laser.png").convert\_alpha()
7. self.image = pygame.transform.scale(self.surface, (wd / 40, ht / 160))
8. self.rect = self.image.get\_rect(center=(x, y))
10. **def shoot(self):**
11. self.rect.x += self.wd/40
13. **def** delete(self): *# Deletes sprite when it goes off-screen*
14. **if** self.rect.right > self.wd **or** self.rect.right < 0:
15. **self.kill()**
17. **def** update(self):
18. self.shoot()
19. self.delete()

Alien() class :

1. **class** Alien(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, wd, ht, alien\_type):
3. super().\_\_init\_\_()
4. self.type = alien\_type
5. **self.wd = wd**
6. self.ht = ht
8. **if** alien\_type == "normal":
9. self.lives = 3
10. **self.surface = pygame.image.load("graphics/alien1.png")**
11. self.image = pygame.transform.scale(self.surface, (wd / 12, ht / 9))
12. self.rect = self.image.get\_rect(center=(wd\*1.1, random.uniform(ht\*0.1, ht\*0.9)))
14. **def** move(self):
15. **if self.type == "normal":**
16. self.rect.centerx -= self.wd / 500
18. **def** update(self, timer):
19. self.move()
21. *# Alien death when it goes off-screen or when it's health is 0*
22. **if** self.rect.centerx < 1 **or** self.lives <= 0:
23. self.kill()

Play() procedure:

1. **def** play():
2. width = 960
3. height = 540
4. play\_game = True
5. **game\_timer = 3600**
6. pygame.init()
7. screen = pygame.display.set\_mode((width, height))
8. pygame.display.set\_caption("Space Game")
9. clock = pygame.time.Clock()
10. **player = pygame.sprite.GroupSingle()**
11. player.add(Player(width, height))
12. laser = pygame.sprite.Group()
13. aliens = pygame.sprite.Group()
14. cooldown = 0
16. **while** play\_game: *# Game loop*
17. **for** event **in** pygame.event.get():
18. **if** event.type == pygame.QUIT:
19. pygame.quit()
20. **exit()**
21. keys = pygame.key.get\_pressed()
23. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
24. laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))
25. **cooldown = 10**
26. cooldown -= 1
28. **if** game\_timer % 200 == 0: *# Adding aliens*
29. aliens.add(Alien(width, height, "normal"))
31. *# Alien hit detection*
32. **for** n **in** laser:
33. **for** alien **in** aliens:
34. **if** pygame.sprite.collide\_rect(n, alien):
35. **n.kill()**
36. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
38. game\_timer -= 1
39. *# Update everything*
40. **screen.fill("black")**
41. player.draw(screen)
42. laser.draw(screen)
43. aliens.update(game\_timer)
44. aliens.draw(screen)
45. **player.update()**
46. laser.update()
47. pygame.display.update()
48. clock.tick(60) *# Caps at 60 fps*

The lasers and aliens are now also updated and drawn towards the end of the game loop.

Player() class (updated):

1. *# Spaceship class controlled by the user*
2. **class** Player(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, wd, ht):
4. super().\_\_init\_\_()
5. **self.wd = wd**
6. self.ht = ht
7. self.image\_sprite = pygame.image.load("graphics/ship1.png")
8. self.image = pygame.transform.scale(self.image\_sprite, (wd/11, ht/11))
9. self.position = pygame.math.Vector2(0, 0)
10. **self.rect = self.image.get\_rect()**
12. **def** player\_input(self): *# Ship movement from input*
13. keys = pygame.key.get\_pressed()
14. dx = keys[pygame.K\_d] - keys[pygame.K\_a]
15. **dy = keys[pygame.K\_s] - keys[pygame.K\_w]**
16. self.direction = pygame.math.Vector2(dx, dy)
17. *# Accounting for diagonal speed by dividing by root 2*
18. **if** dx != 0 **and** dy != 0:
19. self.direction /= 1.41421
21. **if** self.rect.centery < 0 **and** keys[pygame.K\_w]:
22. self.direction = pygame.math.Vector2(self.direction.x, 0)
24. **if** self.rect.centery > self.ht **and** keys[pygame.K\_s]:
25. **self.direction = pygame.math.Vector2(self.direction.x, 0)**
27. *# Focus active when shift is held*
28. **if** keys[pygame.K\_LSHIFT]:
29. self.position += self.direction \* 4
30. **else:**
31. self.position += self.direction \* 8
33. **if** self.rect.left > self.wd:
34. self.rect.right = 0
35. **self.position.x = self.rect.x**
36. **elif** self.rect.right < 0:
37. self.rect.left = self.wd
38. self.position.x = self.rect.x
40. ***# Set rect position to position vector***
41. self.rect.x = round(self.position.x)
42. self.rect.y = round(self.position.y)
44. **def** update(self):
45. **self.player\_input()**

In player\_input(), the program now checks if the player is not at the top/bottom of the screen before moving them up/down. There is also an attempt to implement a feature where the player will loop around if they hit the edge of the screen horizontally.

The main game loop now features a timer (not yet displayed to the user), the ability to create laser objects at the player’s position and collision detection between the lasers and the aliens. Because lasers and aliens are both groups of sprites, the code must cycle through a nested for loop to check every possible interaction between sprites in both groups. When there is an instance of a laser colliding with an alien, the laser will be deleted and the alien’s health will be reduced by one.   
When the update() method is performed for an alien object, it checks to see if its lives are 0 or lower. If this is the case, the alien will be killed.

##### Version 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.1a | Attempt to run the play() function to see if the game still runs | Running play() | The player should still be visible, and the program should not crash |  |
| 4.5 | Checking that the alien sprites appear on screen correctly | Running play() and waiting for multiple aliens to be on screen | The alien should be drawn correctly and move towards the left of the screen | Partial success |
| 4.6a | Attempt to move past vertical edges of the screen | With the game running, moving to the top/bottom edges of the screen | The player should not be able to leave the screen and be stopped at the edge |  |
| 4.6b | Attempt to move past horizontal edges of the screen | With the game running, moving to the left/right edges of the screen | The player should loop back around to the other side of the screen |  |
| 4.7a | Attempt to fire lasers | With the game running, holding the space key | A number of laser objects should be created at the player position and move to the right |  |
| 4.7b | Attempt to shoot an alien | With the game running, pressing space multiple times when aiming at an alien | All lasers that collide with the alien should be deleted. The alien should take exactly three direct hits from lasers before being killed. |  |

##### Version 3 Results:

4.1a:

A screenshot of a computer

Description automatically generated with medium confidence

The pygame window still functions and the game still runs correctly without crashing.

4.5:

Graphical user interface, application, Teams

Description automatically generated

The aliens appear, however the white background as seen in image 4.5 surrounding them should not exists. While this is a minor bug and only visual, it still will be fixed in the next version in order to improve the appearance of the game.

4.6a:

Video 4.6a

4.6b:

Video 4.6b

4.7a:

Video 4.7a

4.7b:

Video 4.7b

These videos show the results from the corresponding tests. As, shown the interaction between the alien and laser objects is successful and the player movement has also been successfully updated.

#### Version 4

For this next build of this stage, I will fix the visual bug with the aliens’ image, add a score counter that is visible to the user, and add a different type of enemy that the player can destroy – asteroids. Asteroids will be created from a single class and for now, will behave the same as aliens in previous version except that they will die in one hit rather than three. Aliens in this new version will also move differently, depending on the position of the player.

Alien() class (fixed and updated):

1. **class** Alien(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, wd, ht, alien\_type):
3. super().\_\_init\_\_()
4. self.type = alien\_type
5. **self.wd = wd**
6. self.ht = ht
8. **if** alien\_type == "normal":
9. self.lives = 3
10. **self.surface = pygame.image.load("graphics/alien1.png").convert\_alpha()**
11. self.image = pygame.transform.scale(self.surface, (wd / 12, ht / 9))
12. self.image.set\_colorkey("white")
13. self.rect = self.image.get\_rect(center=(wd\*1.1, random.uniform(ht\*0.1, ht\*0.9)))
15. **def move(self, px, py):**
16. **if** self.type == "normal":
17. self.rect.centerx -= self.wd / 500
18. *# player 1 side movement*
19. **if** self.rect.centery < py:
20. **self.rect.centery += self.ht / 300**
21. **elif** self.rect.centery > py:
22. self.rect.centery -= self.ht / 300
24. **if** px > self.wd / 2:
25. **self.rect.centerx -= self.wd / 500**
26. **elif** px > self.wd / 10:
27. self.rect.centerx -= self.wd / px
28. **else**:
29. self.rect.centerx -= self.wd / 100
31. **def** update(self, playerx, playery, timer):
32. self.move(playerx, playery)
34. *# Alien death when it goes off-screen or when it's health is 0*
35. **if self.rect.centerx < 1 or self.lives <= 0:**
36. self.kill()

For the Alien() class, .convert\_aplha and .setcolorkey have been used to fixed the issues with the background of the image. The move() method now also takes the player’s x and y position as parameters. An alien will now slowly change its height in an attempt to match that of the player. It will also move faster when the player is close to the left side of the screen. This should make the game more interesting for the user as if every enemy moves in the same way then the game would quickly become boring.

Asteroids class():

1. *# Enemy obstacles class*
2. **class** Asteroids(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, wd, ht, x, y):
4. super().\_\_init\_\_()
5. **self.wd = wd**
6. self.ht = ht
7. self.surface = pygame.image.load("graphics/ast.png").convert\_alpha()
8. self.image = pygame.transform.scale(self.surface, (wd/12, ht/9))
9. self.image.set\_colorkey("white")
10. **self.rect = self.image.get\_rect(center=(x, y))**
12. **def** move(self):
13. self.rect.x -= self.wd/250
15. **def die(self):**
16. self.kill()
18. **def** update(self):
19. self.move()
20. **if self.rect.right < 1:**
21. self.die()

Play procedure (updated):

1. **def** play():
2. width = 960
3. height = 540
4. play\_game = True
5. **game\_timer = 3600**
6. score = 0
7. pygame.init()
8. screen = pygame.display.set\_mode((width, height))
9. pygame.display.set\_caption("Space Game")
11. font1 = pygame.font.Font("graphics/fonts/ARCADE\_I.ttf", round(width / 19))
12. font2 = pygame.font.Font("graphics/fonts/ARCADE\_N.ttf", round(width / 19))
14. clock = pygame.time.Clock()
15. **player = pygame.sprite.GroupSingle()**
16. player.add(Player(width, height))
17. laser = pygame.sprite.Group()
18. enemies = pygame.sprite.Group()
19. aliens = pygame.sprite.Group()
21. score\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()
22. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))

25. **cooldown = 0**

28. **while** play\_game: *# Game loop*
29. **for** event **in** pygame.event.get():
30. **if event.type == pygame.QUIT:**
31. pygame.quit()
32. exit()
33. keys = pygame.key.get\_pressed()
35. **if keys[pygame.K\_SPACE] and cooldown < 1: *# Shooting input + max fire rate***
36. laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))
37. cooldown = 10
38. cooldown -= 1
40. ***# Adding enemies***
41. **if** game\_timer % 352 == 0 **and** game\_timer > 250:
42. attack\_pattern1(enemies, width, height, random.randint(0, height))
44. **if** game\_timer % 401 == 0 **and** game\_timer > 250:
45. **attack\_pattern2(enemies, width, height, random.randint(0, height))**
47. **if** game\_timer % 547 == 0 **and** game\_timer > 250:
48. attack\_pattern3(enemies, width, height, random.randint(0, height))
50. ***# Collision Detection***
51. **if** pygame.sprite.groupcollide(laser, enemies, True, True):
52. score += 100
54. **if** game\_timer % 200 == 0: *# Adding aliens*
55. **aliens.add(Alien(width, height, "normal"))**
57. *# Alien hit detection*
58. **for** n **in** laser:
59. **for** alien **in** aliens:
60. **if pygame.sprite.collide\_rect(n, alien):**
61. n.kill()
62. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
63. **if** alien.\_\_getattribute\_\_("lives") <= 0:
64. score += 500
66. game\_timer -= 1
67. *# Update everything*
68. screen.fill("black")
69. player.draw(screen)
70. **laser.draw(screen)**
71. aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)
72. aliens.draw(screen)
73. enemies.draw(screen)
74. enemies.update()
75. **player.update()**
76. laser.update()
77. screen.blit(score\_surface, score\_rect)
78. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
79. pygame.display.update()
80. **clock.tick(60) *# Caps at 60 fps***

The play() function now tracks the player’s score as a variable. The score is initialised to 0 when starting the game. Score is incremented by 100 when a laser collides with an asteroid (which deletes both the laser and the asteroid), and by 500 when the lives attribute of an alien is decreased to 0 (killing it).  
The game loop now draws the score surface on the screen every frame. Because the score counter is relatively simple it is done without using OOP – the score\_surface and score\_rect are just two variables with the score\_surface being updated using a font loaded from the graphics folder and pygame’s .render function.

I have also added three new procedures to control the placement of asteroids on the screen. These will be used whenever asteroids are created rather than positioning them at a random height on the screen on when they are instantiated. This will make the game more engaging for the user by making the different enemies behave in different ways with where they start.

1. *# Three enemies in horizontal line*
2. **def** attack\_pattern1(sprite\_group, width, height, y):
3. sprite\_group.add(Asteroids(width, height, width \* 1.1, y))
4. sprite\_group.add(Asteroids(width, height, width \* 1.25, y))
5. **sprite\_group.add(Asteroids(width, height, width \* 1.4, y))**

8. *# Three enemies in vertical line*
9. **def** attack\_pattern2(sprite\_group, width, height, y):
10. **sprite\_group.add(Asteroids(width, height, width \* 1.1, y - (0.2 \* height)))**
11. sprite\_group.add(Asteroids(width, height, width \* 1.1, y))
12. sprite\_group.add(Asteroids(width, height, width \* 1.1, y + (0.2 \* height)))

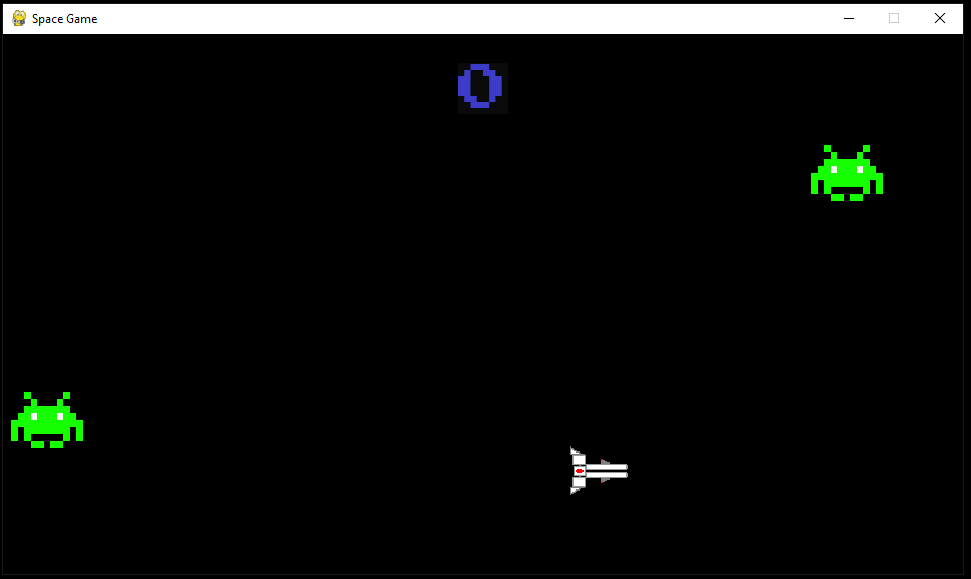
15. ***# Three enemies in diagonal line***
16. **def** attack\_pattern3(sprite\_group, width, height, y):
17. sprite\_group.add(Asteroids(width, height, width \* 1.1, y))
18. sprite\_group.add(Asteroids(width, height, width \* 1.2, y + (0.2 \* height)))
19. sprite\_group.add(Asteroids(width, height, width \* 1.3, y + (0.4 \* height)))

##### Version 4 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.5 | Checking to see that alien sprites are displayed correctly | Running play() | The aliens should appear without a white background |  |
| 4.8a | Testing the enhanced alien movement that responds to player y position | Running play() and moving the player up and down | The aliens should follow and move towards the player height |  |
| 4.8b | Testing the enhanced alien movement that responds to player x position | Running play() and moving the player left | The alien speed should increase as the player reaches the left edge of the screen |  |
| 4.9a | Testing the first attack pattern of asteroids | Running play() and attack\_pattern1() | Three asteroids should appear in a horizontal line and move towards the left of the screen |  |
| 4.9b | Testing the second attack pattern of asteroids | Running play() and attack\_pattern2() | Three asteroids should appear in a vertical line and move towards the left of the screen |  |
| 4.9c | Testing the third attack pattern of asteroids | Running play() and attack\_pattern3() | Three asteroids should appear in a diagonal line and move towards the left of the screen |  |
| 4.10a | Checking that the score timer at the top of the screen updates correctly when an asteroid is killed | With the game running, shooting asteroids | The score counter should increase by 100 for every asteroid killed |  |
| 4.10b | Checking that the score timer at the top of the screen updates correctly when an alien is killed | With the game running, killing aliens | The score counter should increase by 500 for every asteroid killed |  |

##### Version 4 Results:

4.5:



The alien sprites are now drawn without the white in the background of the image.

4.8a:

Video 4.8a  
  
As shown in the video, the aliens correctly respond to the player’s y position, slowly moving to where the player is vertically.

4.8b:

Video 4.8b

This video shows the aliens successfully increasing in speed when the player is near the left-hand side of screen.

4.9a:

Video 4.9a

This evidence shows that the attack\_pattern1() function is successful and produces three asteroid objects in a straight horizontal line.

4.9b:

Video 4.9b

This evidence shows that the attack\_pattern2() function is successful and produces three asteroid objects in a straight vertical line.

4.9c:

Video 4.9c

This shows that the attack\_pattern3() function is successful and produces three asteroid objects in a diagonal line.

4.10a:

Video 4.10a

This video shows the score counter at the top of the screen successfully working because it shows the correct total score after the player destroys asteroids.

4.10b:

Video 4.10b

This video also shows the score counter functioning correctly because the correct total score after the player destroys aliens is displayed.

As all the new features here have been completely successful, I will now move on to the next iteration of building the single player mode of the game.

#### Version 5

Currently, there exists no way in my game for the player to take damage or be killed by the enemies present. I will be fixing this and introducing the player having a limited number of lives which will be displayed in the top left corner of the screen, as planned in the design section.

In order to display the number of lives, I have used a class where an object of it will be constantly updating itself with the number of lives the player has remaining. A list of images loaded from the graphics folder is used to cycle through the animation for the hearts. The total hearts displayed is one object, not multiple.

Hearts() class:

1. *# Number of lives UI*
2. **class** Hearts(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, wd, ht, plr):
4. super().\_\_init\_\_()
5. **self.wd = wd**
6. self.ht = ht
7. self.images = [pygame.image.load("graphics/hearts1.png"),
8. pygame.image.load("graphics/hearts2.png"),
9. pygame.image.load("graphics/hearts3.png"),
10. **pygame.image.load("graphics/hearts4.png"),**
11. pygame.image.load("graphics/hearts5.png")]
13. self.image = pygame.transform.scale(self.images[2], (wd/3, ht/10))
14. **if** plr:
15. **self.rect = self.image.get\_rect(center=(wd/6, ht/30))**
17. **def** animate(self, lvs):
18. self.image = pygame.transform.scale(self.images[lvs-1], (self.wd/3, self.ht/10))
20. **def update(self, lives):**
21. self.animate(lives)

Play() procedure (updated):

1. **def** play():
2. width = 960
3. height = 540
4. play\_game = True
5. **game\_timer = 3600**
6. score = 0
7. lives = 3
8. pygame.init()
9. screen = pygame.display.set\_mode((width, height))
10. **pygame.display.set\_caption("Space Game")**
12. font1 = pygame.font.Font("graphics/fonts/ARCADE\_I.ttf", round(width / 19))
13. font2 = pygame.font.Font("graphics/fonts/ARCADE\_N.ttf", round(width / 19))
15. **clock = pygame.time.Clock()**
16. player = pygame.sprite.GroupSingle()
17. player.add(Player(width, height))
18. laser = pygame.sprite.Group()
19. enemies = pygame.sprite.Group()
20. **aliens = pygame.sprite.Group()**
21. player1\_lives = pygame.sprite.GroupSingle()
22. player1\_lives.add(Hearts(width, height, True))
24. score\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()
25. **score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))**
27. cooldown = 0
29. **while** play\_game: *# Game loop*
30. **for event in pygame.event.get():**
31. **if** event.type == pygame.QUIT:
32. pygame.quit()
33. exit()
34. keys = pygame.key.get\_pressed()
36. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
37. laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))
38. cooldown = 10
39. cooldown -= 1
41. *# Adding enemies*
42. **if** game\_timer % 352 == 0 **and** game\_timer > 500:
43. attack\_pattern1(enemies, width, height, random.randint(0, height))
45. **if game\_timer % 401 == 0 and game\_timer > 500:**
46. attack\_pattern2(enemies, width, height, random.randint(0, height))
48. **if** game\_timer % 547 == 0 **and** game\_timer > 500:
49. attack\_pattern3(enemies, width, height, random.randint(0, height))
51. *# Collision Detection*
52. **if** pygame.sprite.groupcollide(laser, enemies, True, True):
53. score += 100
55. **if game\_timer % 200 == 0: *# Adding aliens***
56. aliens.add(Alien(width, height, "normal"))
58. *# Alien hit detection*
59. **for** n **in** laser:
60. **for alien in aliens:**
61. **if** pygame.sprite.collide\_rect(n, alien):
62. n.kill()
63. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
64. **if** alien.\_\_getattribute\_\_("lives") <= 0:
65. **score += 500**
67. *# Player hit detection*
68. **if** (pygame.sprite.spritecollide(player.sprite, enemies, False) **or**
69. pygame.sprite.spritecollide(player.sprite, aliens, False)) **and** lives > 0:
70. **lives -= 1**
71. **if** lives <= 0:
72. play\_game = False
74. game\_timer -= 1
75. ***# Update everything***
76. screen.fill("black")
77. player.draw(screen)
78. laser.draw(screen)
79. aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)
80. **aliens.draw(screen)**
81. enemies.draw(screen)
82. enemies.update()
83. player.update()
84. laser.update()
85. **screen.blit(score\_surface, score\_rect)**
86. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
87. player1\_lives.update(lives)
88. player1\_lives.draw(screen)
89. pygame.display.update()
90. **clock.tick(60) *# Caps at 60 fps***

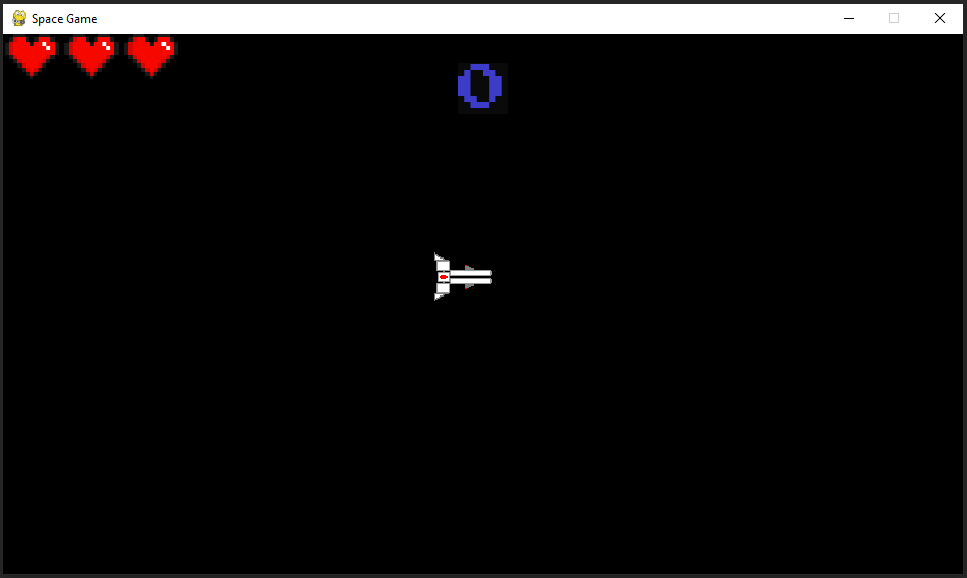
The game loop has now been updated to update and draw the hearts sprite (called player1\_lives). The player’s lives are stored as an integer variable and passed to the hearts sprite every time it is updated. There is now also hit detection for the player, which checks the rectangles of the aliens and the player, or the rectangles of the asteroids and the player overlap. If they do, the player will lose one life if they are not already at 0 lives or lower. If the player is at 0 lives or lower, for now the pygame window should just close after play\_game is set to false and the loop stops.

##### Version 5 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.1a | Checking that the window opens and pygame still doesn’t crash | Running play() | The program should not crash and the window should be displayed as before |  |
| 4.11a | Testing if the hearts sprite is visible | Running play() | The hearts counter should be visible in the top left corner of the screen. The number of hearts should be 3, the value that ‘lives’ is initially set to. |  |
| 4.11b | Testing the player getting by an alien once | Running the game and colliding the player sprite with an alien | The player should lose one life and the number of lives displayed should update | Failed |
| 4.11c | Testing the player getting by an asteroid once | Running play() and colliding the player sprite with an asteroid | The player should lose one life and the number of lives displayed should update | Failed |

##### Version 5 Results:

4.1a/4.11a:



The number of hearts is clearly visible in the top left corner, meaning test 4.11a was successful.

4.11b:

Video 4.11b

This video highlights the issue with this version – the player takes more than one life for colliding with one enemy. This is because the collision detection happens every frame and the asteroid is not deleted upon colliding with the player. For example, if the player collides with an asteroid for 3 frames, they will lose all 3 of their lives almost instantly. This is a major bug and will be fixed immediately by effectively implementing an ‘invincibility frames’.   
In addition, the number of lives displayed also shorts five hearts briefly before the game crashes. This is because the player is losing so many lives at once that it goes to -1. This cycles to the last image in the images list of the hearts object, which is an image that displays 5 hearts.

4.11c:

Video 4.11c

As with the previous video, the player loses many lives over a few frames. This shows that the issue exists for the entire lives system, not just the player taking damage from aliens because the video of the player hitting an asteroid shows the exact same thing with the hearts number changing to 5.

#### Version 6

I have completely reworked the system for how the player loses a life and implemented the ‘invincibility’ frames system in an attempt to stop the issue occurring in the previous version.

Invincibility() procedure:

1. **def** invincibility(inv\_frames, sprite):
2. *# Invincibility frames flashing animation*
3. *# Pycharm marks the passing of a sprite as a warning: "Expected type 'Player', got 'Sprite' instead"*
4. *# but this still functions normally with no bugs.*
5. **if inv\_frames >= 0:**
6. **if** inv\_frames >= 100:
7. Player.take\_dmg2(sprite)
8. **elif** inv\_frames >= 80:
9. Player.take\_dmg1(sprite)
10. **elif inv\_frames >= 60:**
11. Player.take\_dmg2(sprite)
12. **elif** inv\_frames >= 40:
13. Player.take\_dmg1(sprite)
14. **elif** inv\_frames >= 20:
15. **Player.take\_dmg2(sprite)**
16. **else**:
17. Player.take\_dmg1(sprite)
18. **else**:
19. Player.take\_dmg1(sprite)

This Python function is intended to create an invincibility frames flashing animation for a sprite. The function takes in two parameters: inv\_frames, which represents the number of invincibility frames left, and the sprite.

If the invincibility frames are greater than or equal to 0, the function will execute a series of conditions to determine which damage function to call based on the number of invincibility frames remaining. The take\_dmg1 and take\_dmg2 functions are used to simulate the character taking damage.  
If the invincibility frames are less than 0, indicating that the character is no longer invincible, the function will call the take\_dmg1 function on the Player object.

Player() class (updated):

1. *# Spaceship class controlled by the user*
2. **class** Player(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, wd, ht):
4. super().\_\_init\_\_()
5. **self.wd = wd**
6. self.ht = ht
7. self.image\_sprite = pygame.image.load("graphics/ship1.png")
8. self.image\_inv = pygame.image.load("graphics/shipInv.png")
9. self.image = pygame.transform.scale(self.image\_sprite, (wd/11, ht/11))
10. **self.position = pygame.math.Vector2(0, 0)**
11. self.rect = self.image.get\_rect()
13. **def** player\_input(self): *# Ship movement from input*
14. keys = pygame.key.get\_pressed()
15. **dx = keys[pygame.K\_d] - keys[pygame.K\_a]**
16. dy = keys[pygame.K\_s] - keys[pygame.K\_w]
17. self.direction = pygame.math.Vector2(dx, dy)
18. *# Accounting for diagonal speed by dividing by root 2*
19. **if** dx != 0 **and** dy != 0:
20. **self.direction /= 1.41421**
22. **if** self.rect.centery < 0 **and** keys[pygame.K\_w]:
23. self.direction = pygame.math.Vector2(self.direction.x, 0)
25. **if self.rect.centery > self.ht and keys[pygame.K\_s]:**
26. self.direction = pygame.math.Vector2(self.direction.x, 0)
28. *# Focus active when shift is held*
29. **if** keys[pygame.K\_LSHIFT]:
30. **self.position += self.direction \* 4**
31. **else**:
32. self.position += self.direction \* 8
34. **if** self.rect.left > self.wd:
35. **self.rect.right = 0**
36. self.position.x = self.rect.x
37. **elif** self.rect.right < 0:
38. self.rect.left = self.wd
39. self.position.x = self.rect.x
41. *# Set rect position to position vector*
42. self.rect.x = round(self.position.x)
43. self.rect.y = round(self.position.y)
45. **def take\_dmg1(self):**
46. self.image = pygame.transform.scale(self.image\_sprite, (self.wd / 11, self.ht / 11))
48. **def** take\_dmg2(self):
49. self.image = pygame.transform.scale(self.image\_inv, (self.wd / 11, self.ht / 11))
51. **def** death\_check(self, li):
52. **if** li <= 0:
53. self.take\_dmg2()
55. **def update(self, lives):**
56. self.player\_input()
57. self.death\_check(lives)

The player class now was new methods – take\_dmg1, which sets the player image to visible, and take\_dmg\_2, which sets it to hidden. Death\_check is also used to automatically hide the player when they reach 0 lives.

Play() procedure(updated):

1. **def** play():
2. width = 960
3. height = 540
4. play\_game = True
5. **game\_timer = 3600**
6. score = 0
7. lives = 3
8. inv\_frames = 0
9. pygame.init()
10. **screen = pygame.display.set\_mode((width, height))**
11. pygame.display.set\_caption("Space Game")
13. font1 = pygame.font.Font("graphics/fonts/ARCADE\_I.ttf", round(width / 19))
14. font2 = pygame.font.Font("graphics/fonts/ARCADE\_N.ttf", round(width / 19))
16. clock = pygame.time.Clock()
17. player = pygame.sprite.GroupSingle()
18. player.add(Player(width, height))
19. laser = pygame.sprite.Group()
20. **enemies = pygame.sprite.Group()**
21. aliens = pygame.sprite.Group()
22. player1\_lives = pygame.sprite.GroupSingle()
23. player1\_lives.add(Hearts(width, height, True))
25. **score\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()**
26. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))
28. cooldown = 0
29. game\_state = 1
31. **while** play\_game: *# Game loop*
32. **for** event **in** pygame.event.get():
33. **if** event.type == pygame.QUIT:
34. pygame.quit()
35. **exit()**
36. keys = pygame.key.get\_pressed()
37. *# Play game*
38. **if** game\_state == 1:
39. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
40. **laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))**
41. cooldown = 10
42. cooldown -= 1
44. *# Adding enemies*
45. **if game\_timer % 352 == 0 and game\_timer > 500:**
46. attack\_pattern1(enemies, width, height, random.randint(0, height))
48. **if** game\_timer % 401 == 0 **and** game\_timer > 500:
49. attack\_pattern2(enemies, width, height, random.randint(0, height))
51. **if** game\_timer % 547 == 0 **and** game\_timer > 500:
52. attack\_pattern3(enemies, width, height, random.randint(0, height))
54. *# Collision Detection*
55. **if pygame.sprite.groupcollide(laser, enemies, True, True):**
56. score += 100
58. **if** game\_timer % 200 == 0: *# Adding aliens*
59. aliens.add(Alien(width, height, "normal"))
61. *# Alien hit detection*
62. **for** n **in** laser:
63. **for** alien **in** aliens:
64. **if** pygame.sprite.collide\_rect(n, alien):
65. **n.kill()**
66. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
67. **if** alien.\_\_getattribute\_\_("lives") <= 0:
68. score += 500
70. ***# Player hit detection***
71. **if** (pygame.sprite.spritecollide(player.sprite, enemies, False) **or**
72. pygame.sprite.spritecollide(player.sprite, aliens, False)) **and** inv\_frames <= 0:
73. lives -= 1
74. inv\_frames = 120
76. invincibility(inv\_frames, player.sprite)
78. game\_timer -= 1
79. inv\_frames -= 1
80. ***# Update everything***
81. screen.fill("black")
82. **if** lives > 0:
83. player.draw(screen)
84. player.update(lives)
85. **else:**
86. game\_state = 2
87. laser.draw(screen)
88. aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)
89. aliens.draw(screen)
90. **enemies.draw(screen)**
91. enemies.update()
92. laser.update()
93. screen.blit(score\_surface, score\_rect)
94. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
95. **player1\_lives.update(lives)**
96. player1\_lives.draw(screen)
97. **else**:
98. *# Game over screen*
99. screen.fill("red")
100. **screen.blit(score\_surface, score\_rect)**
101. pygame.display.update()
102. clock.tick(60) *# Caps at 60 fps*

The main changes to the play() procedure include the addition of a game over screen when the player reaches 0 lives, which is done through the game\_state variable. For now, in game state 2 the program will stop drawing and updating all sprites apart from the score counter. The background will also change to red (a placeholder until a proper game over screen is completed).   
Another change is the check that the player doesn’t have invincibility frames before applying a life reduction, even if they are still colliding with an enemy. When the player is hit, they receive 120 frames where they cannot be damaged.

##### Version 6 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.1a | Checking to see if the pygame window still runs without crashing | Running play() | The window should open and all sprites should be drawn correctly |  |
| 4.11b | Testing the player getting by an alien once | Running the game and colliding the player sprite with an alien | The player should lose one life, the number of lives displayed should update and the player sprite should flash (showing the invincibility frames) |  |
| 4.11c | Testing the player getting by an asteroid once | Running play() and colliding the player sprite with an asteroid | The player should lose one life, the number of lives displayed should updated, and the player sprite should flash |  |
| 4.12a | Testing whether the player can lose lives during the invincibility flashing animation | During invincibility frames, moving the player sprite to collide with another enemy | The player should not lose any more lives. For example, if the player was set to 2 lives upon getting hit, they should stay at 2 lives. |  |
| 4.12b | Observing what happens when the player is reduced to 0 lives | After invincibility frames have worn off, getting the player hit again. This can be repeated until the player dies. | Normal gameplay should stop and the ‘game over’ screen should be displayed. |  |
| 4.12c | Attempt to fire lasers during invincibility frames | Pressing the shoot button during invincibility flashing animation. | The ship should continue to fire lasers as normal. |  |

##### Version 6 Results:

4.1a:

Graphical user interface

Description automatically generated

The game still runs, and all objects are displayed correctly.

4.11b:

Video 4.11b-2

4.11c:

Video 4.11c-2

As shown in both videos, the player can now lose a single life correctly without the bug from the previous version occurring.

4.12a:

Video 4.12a

This video shows that the player cannot be damaged or killed in the brief time the invincibility frames occur. This is intentional and a success.

4.12b:

Video 4.12b

This video showcases how when the player loses their final life, the main game ends and the game over screen is displayed.

4.12c:

Video 4.12c

As shown, the player can still fire lasers while invincible. This is intentional because it allows the player to possible kill the enemy that cost them a life. This shows test 4.12c was successful.

### 3.2.4.2 Feedback from Stakeholders on Single Player Mode

I have shown version 6 of the single player game to my group of student stakeholders, in addition to explaining that this is an early built of the game and that the program is not finished. In general, after interviewing them, they were somewhat happy with how this early build of the game functions but found some issues and had many new features that they wanted implemented into the game.

One of the main issues with the game currently is that my stakeholders say that it is “too easy”.  
The only way that the player can be damaged is through directly collide with an enemy, making it easy to avoid losing a life. I will aim to fix this in the next version. Many of my stakeholders suggested to implement projectiles that the enemy can fire which the player will not be able to destroy. They said that there should be bullets that the player has to dodge in order to make the game more like a conventional shoot ‘em up game.

Another improvement my stakeholders suggested is to add an animation on the hearts counter whenever the player takes damage. This will help to indicate when the player takes damage and make the game easier to understand for the user.

There is also another visual improvement which my stakeholders suggested – to make the background more appealing visually. At the moment, the background is just a solid black colour. I will rectify this by making the background a sprite that moves in the next version.

The stakeholders also didn’t like the fact that the game only really had a single level. In future versions the students expect to see multiple levels in the game which are different visually.

Some of my stakeholders also thought the game would benefit from a pause system, which will allow the user to stop the game when a certain input is pressed. This would obviously be a quality-of-life improvement for a user playing the game as they can pause the game if they have something to do in the real would and come back to the game later, with nothing lost.

#### Version 7

In this version, I will aim to implement the changes mentioned above by the stakeholders. This will be done by developing a number of new sprites and updating the existing game loop to include them. First, I will develop a class for changing background of the game to a moving image rather than just a solid colour as well as a system to pause the game.

Background() class:

1. **class** Background(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, width, height, level):
3. super().\_\_init\_\_()
4. **if** level == 1:
5. **surface = pygame.image.load("graphics/bg.png").convert\_alpha()**
6. **else**:
7. surface = pygame.image.load("graphics/bg2.jpg").convert\_alpha()
9. self.image = pygame.transform.scale(surface, (width\*20, height))
11. self.rect = self.image.get\_rect(center=(width\*10, height/2))
13. **def** scroll(self, wd):
14. self.rect.centerx -= wd\*0.008
15. **if self.rect.right <= wd:**
16. self.rect.centerx = wd\*10
18. **def** update(self, width):
19. self.scroll(width)

This class defines the Background sprite object and its associated methods. The constructor method initializes the image of the background according to the level. The scroll and update methods allow for the background to scroll and update its position on the game window.

Play() procedure (updated):

1. **while** play\_game: *# Game loop*
2. **for** event **in** pygame.event.get():
3. **if** event.type == pygame.QUIT:
4. pygame.quit()
5. **exit()**
6. **if** event.type == pygame.KEYDOWN **and** (game\_state == 1):
7. **if** event.key == pygame.K\_ESCAPE:
8. game\_state = 10
9. **if** event.type == pygame.KEYDOWN **and** (game\_state == 10):
10. **if event.key == pygame.K\_RETURN:**
11. game\_state = 1
13. keys = pygame.key.get\_pressed()
14. *# Play game*
15. **if game\_state == 1:**
16. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
17. laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))
18. cooldown = 10
19. cooldown -= 1
21. *# Adding enemies*
22. **if** game\_timer % 352 == 0 **and** game\_timer > 500:
23. attack\_pattern1(enemies, width, height, random.randint(0, height))
25. **if game\_timer % 401 == 0 and game\_timer > 500:**
26. attack\_pattern2(enemies, width, height, random.randint(0, height))
28. **if** game\_timer % 547 == 0 **and** game\_timer > 500:
29. attack\_pattern3(enemies, width, height, random.randint(0, height))
31. *# Collision Detection*
32. **if** pygame.sprite.groupcollide(laser, enemies, True, True):
33. score += 100
35. **if game\_timer % 200 == 0: *# Adding aliens***
36. aliens.add(Alien(width, height, "normal"))
38. *# Alien hit detection*
39. **for** n **in** laser:
40. **for alien in aliens:**
41. **if** pygame.sprite.collide\_rect(n, alien):
42. n.kill()
43. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
44. **if** alien.\_\_getattribute\_\_("lives") <= 0:
45. **score += 500**
47. *# Player hit detection*
48. **if** (pygame.sprite.spritecollide(player.sprite, enemies, False) **or**
49. pygame.sprite.spritecollide(player.sprite, aliens, False)) **and** inv\_frames <= 0:
50. **lives -= 1**
51. inv\_frames = 120
53. invincibility(inv\_frames, player.sprite)
55. **game\_timer -= 1**
56. inv\_frames -= 1
58. *# Update everything*

61. **if** lives > 0:
62. player.draw(screen)
63. player.update(lives)
64. **else**:
65. **game\_state = 2**
66. laser.draw(screen)
67. aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)
68. aliens.draw(screen)
69. enemies.draw(screen)
70. **enemies.update()**
71. laser.update()
72. screen.blit(score\_surface, score\_rect)
73. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
74. player1\_lives.update(lives)
75. **player1\_lives.draw(screen)**
76. *# Drawing new background*
77. bg.draw(screen)
78. bg.update(width)
79. **elif** game\_state == 2:
80. ***# Game over screen***
81. screen.fill("red")
82. screen.blit(score\_surface, score\_rect)
83. *# Pause screen*
84. **else**:
85. **pass**
86. pygame.display.update()
87. clock.tick(60) *# Caps at 60 fps*

The game loop now should update and draw the background. A new variable called ‘level’ has also been added which is used to update the background. This will make the code easy to update later when a full level system is added. I have also added a pause screen which is opened when the user presses escape during the game. This will stop the sprites updating until the user presses enter.

##### Version 7 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.7.1 | Testing if the new background displays correctly | Running play() level = 1 | The new background should appear and move from right to left. The old sprites should still be visible. | Failed |

##### Version 7 Results:

4.7.1:

A screen shot of a computer

Description automatically generated with low confidence

Test 4.7.1 failed because, while the background is visible and was moving correctly, the other sprites were not. I now know this is because the background was drawn after the other sprites, causing it to be shown on the top ‘layer’. This bug can be easily fixed by drawing the background sprite first, before all the other sprites.

#### Version 8

Play() procedure (updated):

1. **def** play():
2. ...
3. *# Update everything*
4. bg.draw(screen)
5. **bg.update(width)**
6. *# Background drawn first so other sprites appear in front of it*
7. **if** lives > 0:
8. player.draw(screen)
9. player.update(lives)
10. **else:**
11. game\_state = 2
12. laser.draw(screen)
13. aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)
14. aliens.draw(screen)
15. **enemies.draw(screen)**
16. enemies.update()
17. laser.update()
18. screen.blit(score\_surface, score\_rect)
19. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
20. **player1\_lives.update(lives)**
21. player1\_lives.draw(screen)
23. **elif** game\_state == 2:
24. *# Game over screen*
25. **screen.fill("red")**
26. screen.blit(score\_surface, score\_rect)
27. *# Pause screen*
28. **else**:
29. **pass**
30. **pygame.display.update()**
31. clock.tick(60) *# Caps at 60 fps*

##### Version 8 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.7.1 | Testing if the new background displays correctly | Running play() level = 1 | The new background should appear and move from right to left. The old sprites should still be visible. | Success |
| 4.7.2 | Testing if the background will display correctly on level 2 | Running play() level = 2 | The new background (different to part 1) should appear and move from right to left. The old sprites should still be visible. |  |
| 4.7.3a | Attempt to pause the game | With the game running, pressing escape | The game should pause, and sprites should stop moving |  |
| 4.7.3b | Attempt to resume the game | On the pause menu, pressing enter | The game should resume |  |

##### Version 8 Results:

4.7.1:

Video 4.7.1

4.7.2:

Video 4.7.2

The background now successfully renders for both levels with no further bugs.

4.7.3a/4.7.3b:

Video 4.7.3

The pause menu can be seen working in video 4.7.3. The game is successfully paused and un-paused with everything going as expected. This means that both tests 4.7.3a and 4.7.3b have passed with no complications.

#### Version 9

Because the background is now finished, I will work on the stakeholders’ other requests. For this section, I will work on improving the hearts counter to fit the stakeholders’ expectations of a taking damage animation as well as adding projectiles that the player must dodge in order to make the game more difficult.

Hearts() class (updated):

1. *# Number of lives UI*
2. **class** Hearts(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, wd, ht, plr):
4. super().\_\_init\_\_()
5. **self.wd = wd**
6. self.ht = ht
7. self.images = [pygame.image.load("graphics/hearts1.png"),
8. pygame.image.load("graphics/hearts2.png"),
9. pygame.image.load("graphics/hearts3.png"),
10. **pygame.image.load("graphics/hearts4.png"),**
11. pygame.image.load("graphics/hearts5.png"),
12. pygame.image.load("graphics/hearts1.5.png"),
13. pygame.image.load("graphics/hearts2.5.png"),
14. pygame.image.load("graphics/hearts3.5.png"),
15. **pygame.image.load("graphics/hearts4.5.png")]**
17. self.image = pygame.transform.scale(self.images[2], (wd/3, ht/10))
18. **if** plr:
19. self.rect = self.image.get\_rect(center=(wd/6, ht/30))
20. **else:**
21. self.rect = self.image.get\_rect(center=(wd/6, ht/1.05))
23. **def** animate(self, lvs, inv): *# Inv frames prevent multiple lives lost at once*
24. **if** inv <= 90:
25. **self.image = pygame.transform.scale(self.images[lvs-1], (self.wd/3, self.ht/10))**
26. **elif** lvs == 1:
27. self.image = pygame.transform.scale(self.images[-4], (self.wd/3, self.ht/10))
28. **elif** lvs == 2:
29. self.image = pygame.transform.scale(self.images[-3], (self.wd/3, self.ht/10))
30. **elif lvs == 3:**
31. self.image = pygame.transform.scale(self.images[-2], (self.wd/3, self.ht/10))
32. **elif** lvs == 4:
33. self.image = pygame.transform.scale(self.images[-1], (self.wd/3, self.ht/10))
35. **def update(self, lives, frames):**
36. self.animate(lives, frames)

For this class, the animate() method has been updated to include the number of invincibility frames remaining as a parameter. This is used to show a simple animation in the UI when the player gets hit. The images list contains these additional

EnemyBullets() class:

1. *# Circular enemy bullets*
2. **class** EnemyBullets(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, wd, ht, y):
4. super().\_\_init\_\_()
5. **surface = pygame.image.load("graphics/circle\_bullet.png")**
6. self.image = pygame.transform.scale(surface, (wd/70, ht/40))
7. self.rect = self.image.get\_rect(center=(wd, y))
8. self.dir = random.uniform(-2, 2)
10. **def move(self):**
11. **if** self.rect.centerx < 0:
12. self.kill()
13. **else**:
14. self.rect.centerx -= 10 - abs(self.dir\*self.dir)
15. **self.rect.centery += self.dir**
17. **def** update(self):
18. self.move()

This class creates EnemyBullets objects that move across the screen and have random directions. The class initializes the enemy bullet with a width, height, and y coordinate and loads a graphic for the bullet from the graphics folder. The move and update methods allow the bullet to move across the screen and delete itself if it reaches the edge.

Play() procedure(updated):

1. **def** play():
2. width = 960
3. height = 540
4. play\_game = True
5. **game\_timer = 3600**
6. score = 0
7. lives = 3
8. level = 1
9. inv\_frames = 0
10. **pygame.init()**
11. screen = pygame.display.set\_mode((width, height))
12. pygame.display.set\_caption("Space Game")
14. font1 = pygame.font.Font("graphics/fonts/ARCADE\_I.ttf", round(width / 19))
15. **font2 = pygame.font.Font("graphics/fonts/ARCADE\_N.ttf", round(width / 19))**
17. clock = pygame.time.Clock()
19. bg = pygame.sprite.GroupSingle()
20. **bg.add(Background(width, height, level))**
22. player = pygame.sprite.GroupSingle()
23. player.add(Player(width, height))
24. laser = pygame.sprite.Group()
25. **enemies = pygame.sprite.Group()**
26. aliens = pygame.sprite.Group()
27. badlaser = pygame.sprite.Group()
28. player1\_lives = pygame.sprite.GroupSingle()
29. player1\_lives.add(Hearts(width, height, True))
31. score\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()
32. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))
34. cooldown = 0
35. **game\_state = 1**
37. **while** play\_game: *# Game loop*
38. **for** event **in** pygame.event.get():
39. **if** event.type == pygame.QUIT:
40. **pygame.quit()**
41. exit()
42. **if** event.type == pygame.KEYDOWN **and** (game\_state == 1):
43. **if** event.key == pygame.K\_ESCAPE:
44. game\_state = 10
45. **if event.type == pygame.KEYDOWN and (game\_state == 10):**
46. **if** event.key == pygame.K\_RETURN:
47. game\_state = 1
49. keys = pygame.key.get\_pressed()
50. ***# Play game***
51. **if** game\_state == 1:
52. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
53. laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))
54. cooldown = 10
55. **cooldown -= 1**
57. *# Adding enemies*
58. **if** game\_timer % 352 == 0 **and** game\_timer > 500:
59. attack\_pattern1(enemies, width, height, random.randint(0, height))
61. **if** game\_timer % 401 == 0 **and** game\_timer > 500:
62. attack\_pattern2(enemies, width, height, random.randint(0, height))
64. **if** game\_timer % 547 == 0 **and** game\_timer > 500:
65. **attack\_pattern3(enemies, width, height, random.randint(0, height))**
67. **if** random.randint(0, game\_timer + 1000) <= 50 **and** game\_timer > 250:
68. badlaser.add(EnemyBullets(width, height, random.uniform(height \* 0.1, height \* 0.9)))

71. *# Collision Detection*
72. **if** pygame.sprite.groupcollide(laser, enemies, True, True):
73. score += 100
75. **if game\_timer % 300 == 0: *# Adding aliens***
76. aliens.add(Alien(width, height, "normal"))
78. *# Alien hit detection*
79. **for** n **in** laser:
80. **for alien in aliens:**
81. **if** pygame.sprite.collide\_rect(n, alien):
82. n.kill()
83. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
84. **if** alien.\_\_getattribute\_\_("lives") <= 0:
85. **score += 500**
87. *# Player hit detection*
88. **if** (pygame.sprite.spritecollide(player.sprite, enemies, False) **or**
89. pygame.sprite.spritecollide(player.sprite, badlaser, False) **or**
90. **pygame.sprite.spritecollide(player.sprite, aliens, False)) and inv\_frames <= 0:**
91. lives -= 1
92. inv\_frames = 120
94. invincibility(inv\_frames, player.sprite)
96. game\_timer -= 1
97. inv\_frames -= 1
99. *# Update everything*
100. **bg.draw(screen)**
101. bg.update(width)
103. badlaser.draw(screen)
104. badlaser.update()
105. **if lives > 0:**
106. player.draw(screen)
107. player.update(lives)
108. **else**:
109. game\_state = 2
110. **laser.draw(screen)**
111. aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)
112. aliens.draw(screen)
113. enemies.draw(screen)
114. enemies.update()
115. **laser.update()**
116. screen.blit(score\_surface, score\_rect)
117. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
118. player1\_lives.update(lives, inv\_frames)
119. player1\_lives.draw(screen)
121. **elif** game\_state == 2:
122. *# Game over screen*
123. screen.fill("red")
124. screen.blit(score\_surface, score\_rect)
125. ***# Pause screen***
126. **else**:
127. **pass**
128. pygame.display.update()
129. clock.tick(60) *# Caps at 60 fps*

At the start of the play() procedure on line 27 a new sprite group called ‘badlaser’. This group will be used to store the sprites of any projectiles that can damage the player. Because of this, this group has been included in the player hit detection on line 89. For now, enemyBullets objects (which are created randomly) will be the only sprites in this group.

##### Version 9 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.9.1 | Testing that the new bullets display correctly | Running play() | Small, circular bullets should appear and move with a partially random direction. |  |
| 4.9.2 | Testing the new losing a life animation | With the game running, getting hit multiple times. | The hearts UI should have a small animation on every life lost. |  |
| 4.9.3 | Checking that the new bullets can damage the player | Colliding the player with a small, circular bullet. | A life should be lost. |  |
| 4.9.4 | Ensuring that invincibility frames work for the new bullets. | Colliding the player with new bullets multiple times. | Another life should not be lost while the invincibility flashing animation occurs. |  |

##### Version 9 Results:

4.9.1:

A screenshot of a video game

Description automatically generated

The enemy bullets are clearly visible just ahead of the player in this screenshot. This shows test 4.9.1 was successful.

4.9.2:

Video 4.9.2

4.9.3/4.9.4:

Video 4.9.3

This video showcases that the hit detection for the new bullets colliding with the player is working as expected. The player will lose a life like normal when an enemy bullet hits them.

#### Version 10

For this version, I will properly implement the level system into the game as well as adding lasers than the aliens fire at the player. This should make the game harder, which is what the stakeholders thought was needed in the game outlined in 3.2.3.2. In order to properly add a level system, there will be a message that displays on screen when the player reaches the end of a level. This will display the next level number.   
Currently, there is also no way for the player to win the game. I will change this by making it so there is a final level, and when the game timer reaches 0 on this final level, the player wins. There will be a separate win screen when this happens, similar to how there is a game over screen when the player dies.

EnemyLasers() class:

1. *# Enemy weapon class*
2. **class** EnemyLasers(Lasers):
3. **def** shoot(self):
4. surface = pygame.image.load("graphics/laser2.png").convert\_alpha()
5. **self.image = pygame.transform.scale(surface, (self.wd / 40, self.ht / 180))**
6. self.rect.x -= self.wd/80
8. **def** delete(self): *# Deletes sprite when it goes off-screen*
9. **if** self.rect.right < 0:
10. **self.kill()**

This class inherits from the existing Lasers() class (used for the player’s weapon). However, polymorphism is applied which should make this laser look and move differently to the player’s lasers. These projectiles should be coloured green and travel from the right side of the screen to the left.

Texts() class:

1. **class** Texts(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, wd, ht, word, font, size):
3. super().\_\_init\_\_()
4. self.wd = wd
5. **self.ht = ht**
6. self.surface = font.render(str(word), False, (200, 200, 200))
7. self.image = pygame.transform.scale(self.surface, (ht\*size, wd\*size))
8. self.rect = self.image.get\_rect(center=(wd/2, ht/2))
10. **def change(self, word, font, size):**
11. self.surface = font.render(word, False, (200, 200, 200))
12. self.image = pygame.transform.scale(self.surface, (self.wd \* size \* 2, self.ht \* size))
13. self.rect = self.image.get\_rect(center=(self.wd / 2, self.ht / 2))
15. **def update(self, word, font, size, is\_shown):**
16. **if** is\_shown:
17. self.change(word, font, size)
18. **else**:
19. self.image = font.render("", False, (0, 0, 0))

This class is a generic template for displaying a message on the game screen. It will be used for the text displayed on the win screen. I will also update the game-over screen to feature a message in the centre. This class allows for maintainable code because it can be used in the future anytime, I need to show text in the game. It takes the string to be shown, the font used, and the size of the text as parameters.

Play() procedure (updated):

1. **def** play():
2. width = 960
3. height = 540
4. play\_game = True
5. **game\_timer = 360**
6. text\_delay = 0
7. score = 0
8. lives = 3
9. level = 1
10. **dif = "NORMAL"**
11. inv\_frames = 0
12. pygame.init()
13. screen = pygame.display.set\_mode((width, height))
14. pygame.display.set\_caption("Space Game")
16. font1 = pygame.font.Font("graphics/fonts/ARCADE\_I.ttf", round(width / 19))
17. font2 = pygame.font.Font("graphics/fonts/ARCADE\_N.ttf", round(width / 19))
18. level\_text = font2.render("", False, (255, 255, 255), (0, 0, 0))
19. level\_rect = level\_text.get\_rect(center=(width / 3, height / 2))
20. **message = pygame.sprite.GroupSingle()**
21. message.add(Texts(width, height, "word", font2, 0.1))
23. clock = pygame.time.Clock()
25. **bg = pygame.sprite.GroupSingle()**
26. bg.add(Background(width, height, level))
28. player = pygame.sprite.GroupSingle()
29. player.add(Player(width, height))
30. **laser = pygame.sprite.Group()**
31. enemies = pygame.sprite.Group()
32. aliens = pygame.sprite.Group()
33. badlaser = pygame.sprite.Group()
34. player1\_lives = pygame.sprite.GroupSingle()
35. **player1\_lives.add(Hearts(width, height, True))**
37. score\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()
38. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))
40. **cooldown = 0**
41. alien\_cooldown = 10
42. game\_state = 1
44. **while** play\_game: *# Game loop*
45. **for event in pygame.event.get():**
46. **if** event.type == pygame.QUIT:
47. pygame.quit()
48. exit()
49. **if** event.type == pygame.KEYDOWN **and** (game\_state == 1):
50. **if event.key == pygame.K\_ESCAPE:**
51. text\_delay = 40
52. game\_state = 10
53. **if** event.type == pygame.KEYDOWN **and** (game\_state == 10):
54. **if** event.key == pygame.K\_ESCAPE **and** text\_delay <= 0:
55. **game\_state = 1**
57. keys = pygame.key.get\_pressed()
58. *# Play game*
59. **if** game\_state == 1:
60. ***# Gameplay***
61. bg\_changed = False
62. game\_timer -= 1
63. keys = pygame.key.get\_pressed()
65. **if game\_timer <= 1:**
66. game\_timer = 360
67. text\_delay = 0
68. level\_text = font2.render("", False, (255, 255, 255), (0, 0, 0))
69. i = 0
70. **if level == 3: *# Final level***
71. game\_state = 9
72. **else**:
73. game\_state = 4
74. level += 1
76. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
77. laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))
78. **if** dif == "NORMAL":
79. cooldown = 10
81. cooldown -= 1
83. *# Collision Detection*
84. **if** pygame.sprite.groupcollide(laser, enemies, True, True):
85. **score += 100**
87. **if** (pygame.sprite.spritecollide(player.sprite, enemies, False) **or**
88. pygame.sprite.spritecollide(player.sprite, badlaser, False) **or**
89. pygame.sprite.spritecollide(player.sprite, aliens, False)) **and** inv\_frames <= 0:
90. **lives -= 1**
91. inv\_frames = 120
93. *# Adding enemies*
94. **if** game\_timer % 352 == 0 **and** game\_timer > 250:
95. **attack\_pattern1(enemies, width, height, random.randint(0, height))**
97. **if** game\_timer % 401 == 0 **and** game\_timer > 250:
98. attack\_pattern2(enemies, width, height, random.randint(0, height))
100. **if game\_timer % 547 == 0 and game\_timer > 250:**
101. attack\_pattern3(enemies, width, height, random.randint(0, height))
103. **if** random.randint(0, game\_timer + 1000) <= 50 **and** game\_timer > 250:
104. badlaser.add(EnemyBullets(width, height, random.uniform(height \* 0.1, height \* 0.9)))
106. *# Alien hit detection*
107. **for** n **in** laser:
108. **for** alien **in** aliens:
109. **if** pygame.sprite.collide\_rect(n, alien):
110. **n.kill()**
111. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
112. **if** alien.\_\_getattribute\_\_("lives") <= 0:
113. score += 500
115. **if game\_timer % 300 == 0:**
116. aliens.add(Alien(width, height, "normal"))
118. **else**:
119. **for** alien **in** aliens:
120. **if alien\_cooldown <= 0 and alien.\_\_getattribute\_\_("type") == "normal":**
121. badlaser.add(EnemyLasers(alien.rect.centerx, alien.rect.centery, width, height))

124. alien\_cooldown -= 1
125. **invincibility(inv\_frames, player.sprite)**
127. *# Draw background first*
128. bg.draw(screen)
129. bg.update(width)
131. inv\_frames -= 1
132. *# Sprites drawing and updating*
133. laser.draw(screen)
134. laser.update()
135. **aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)**
137. aliens.draw(screen)
138. enemies.draw(screen)
139. enemies.update()

142. badlaser.draw(screen)
143. badlaser.update()
145. **if lives > 0:**
146. player.draw(screen)
147. player.update(lives)
148. **else**:
149. game\_state = 2
150. **screen.blit(score\_surface, score\_rect)**
151. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
152. player1\_lives.update(lives, inv\_frames)
153. player1\_lives.draw(screen)
155. **elif game\_state == 2:**
156. *# Game over screen*
157. screen.fill("black")
158. screen.blit(score\_surface, score\_rect)
159. message.update("GAME OVER", font1, 0.2, True)
160. **message.draw(screen)**
162. *# Level transition screen*
163. **elif** game\_state == 4:
164. screen.fill((0, 0, 0))
165. **if not bg\_changed:**
166. bg.empty()
167. bg.add(Background(width, height, level))
168. bg\_changed = True
170. **text = "LEVEL " + str(level)**
171. **if** text\_delay >= 10:
172. **if** i <= len(text):
174. level\_text = font2.render(text[:i], False, (255, 255, 255), (0, 0, 0))
175. **i += 1**
176. text\_delay = 0
177. **elif** text\_delay >= 50:
178. game\_state = 1
179. text\_delay += 1
180. **screen.blit(level\_text, level\_rect)**
182. **elif** game\_state == 9:
183. *# Win screen*
184. keys = pygame.key.get\_pressed()
185. **game\_timer += 1**
187. screen.fill("black")
188. *# score\_rect = score\_surface.get\_rect(center=(width / 2, height / 1.5))*
190. **if game\_timer % 40 >= 20:**
191. message.update("You win!", font2, 0.2, False)
192. **else**:
193. message.update("You win!", font2, 0.2, True)
195. **message.draw(screen)**
196. screen.blit(score\_surface, score\_rect)
198. laser.empty()
199. aliens.empty()
200. **badlaser.empty()**
201. enemies.empty()
202. *# save score function to go here*
204. *# Pause screen*
205. **else:**
206. text\_delay -= 1
207. message.update("PAUSED", font1, 0.15, True)
208. message.draw(screen)
209. pygame.display.update()
210. **clock.tick(60) *# Caps at 60 fps***

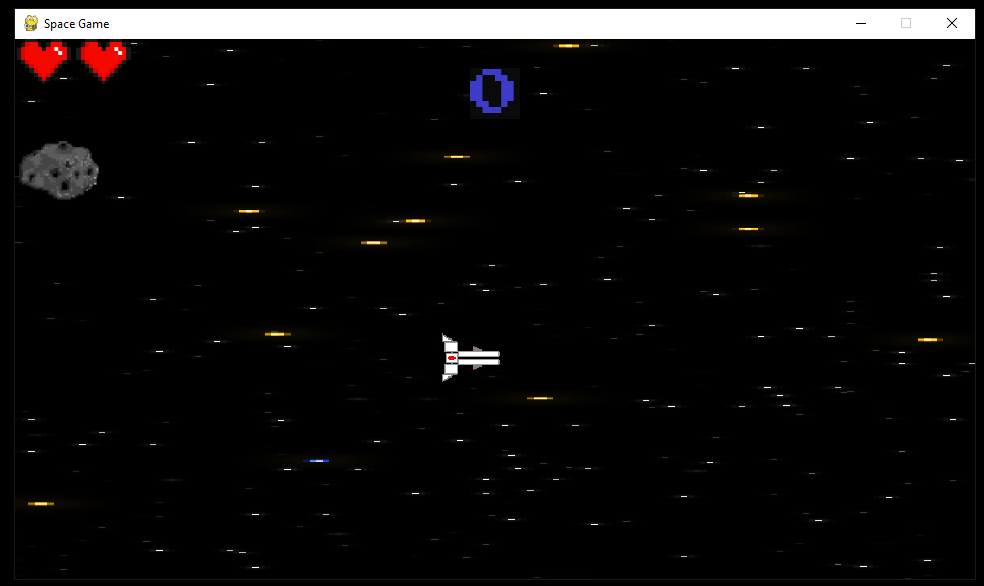
Many changes have been made to the play() procedure and the main game loop since the last version. Firstly, there is now the additional game states used for the level transition screen and the win screen. These screen both use the message sprite to temporarily display text on the screen. When I finish developing the highscores system, when the player reaches the win screen their score will be saved using a function.  
 Secondly, the game timer is now reset going into a new level. This timer has temporarily been set to 100 frames in order to make testing easier. When the timer reaches 0 of the final level, the game state variable is changed to the win screen.  
Additionally, there is now a for loop on line 119 that cycles through every alien in the sprite group and (if the value of cooldown is 0 or lower) creates an enemy laser object at their location. I have done it this way to add a new feature where aliens can fire their own lasers which can damage the player.  
A variable called ‘dif’ has been introduced at the start. For now, it can only be set to “NORMAL” but will be more important later on in development, when the player will be able to change the difficulty.  
The pause menu also should show a “paused” message on screen by updating and drawing the message sprite. The pause menu should now also be opened and closed using the escape key.

##### Version 10 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.1a | Testing if the game can still run without errors | Running play() | The pygame window should appear as before without crashing. |  |
| 4.13a | Testing if aliens can fire lasers correctly | Running the game | Every alien should fire green lasers at a normal rate. | Failed – aliens were firing without cooldown |
| 4.13b | Testing if the player is damaged by an alien laser | With the game running, colliding the player with a green laser | The player should take damage as receive invincibility frames as normal |  |
| 4.14a | Testing if the level transition screen works for  level 1 -> level 2 | Waiting for game timer to hit 0 on level 1 | The level 2 message should appear on the screen and be animated correctly, displaying on letter at a time. |  |
| 4.14b | Testing if the level transition screen works for  level 2 -> level 3 | Waiting for game timer to hit 0 on level 2 | The level 3 message should appear on the screen and be animated correctly, displaying on letter at a time. |  |
| 4.14c | Testing if the win screen displays correctly | Waiting for game timer to hit 0 on level 3 | The win screen should appear, with the “you win” text flashing in the centre. |  |
| 4.14d | Testing if the game-over screen displays correctly | Letting the player reach 0 lives | The new game over screen should display, with the message “game over” in the centre |  |
| 4.14e | Testing if the pause screen displays correctly | Pressing escape during the game | The pause message should appear, and the game should be paused. |  |
| 4.14 f | Testing if the game un-pauses correctly | Pressing escape during the pause menu | The pause message should disappear, and the game should resume |  |

##### Version 10 Results

4.1a:



The pygame window still runs and displays correctly.

4.13a/4.13b:

Video 4.13

This video highlights the issue with the lasers currently. There exists a logic error where they are being fired continuously without any cooldown. This is not intentional and should not happen.

4.14a:

Video 4.14a

4.14b:

Video 4.14b

These videos show that the score transition part of the game loop is working correctly and shows the correct level number for the next level. The animation for the text is also working as expected.

4.14c:

Video 4.14c

As seen in this video, the victory screen appears at the correct time (the end of level 3) and displays the correct message as well as the player’s score. The flashing animation of the message is also working as intended.

4.14d:

Video 4.14d

The game over screen displays the correct message in the expected font.

4.14e/4.14f:

Video 4.14e

This video shows the pause menu has been successfully updated and the ‘paused’ message works as expected and is only shown during the paused state. The game can resume from the pause menu without issues.

#### Version 11

After fixing the issues with the alien lasers, I will add a new feature – items that the player can pick up by colliding with them. I will create these items using a Pickup() class to allow the generation of multiple items efficiently.

Play() procedure (fixed):

1. **def** play():
2. ...
3. **if** game\_state == 1:
5. **if game\_timer % 270 == 0: *# Adding aliens***
6. aliens.add(Alien(width, height, "normal"))
7. **else**:
8. **for** alien **in** aliens:
9. **if** alien\_cooldown <= 0 **and** alien.\_\_getattribute\_\_("type") == "normal":
10. **badlaser.add(EnemyLasers(alien.rect.centerx, alien.rect.centery, width, height))**
11. alien\_cooldown = 20
12. alien\_cooldown -= 1
13. ...

The variable alien\_cooldown will now be set to 20 every time a laser fired. Enemy lasers can only be fired when the cooldown reaches 0 and this cooldown is decremented every frame. Having the code like this allows for implementation of a maximum firing rate for the aliens (every 20 frames currently).

Pickup() class:

1. **class** Pickup(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, wd, ht):
3. super().\_\_init\_\_()
4. self.type = "star"
5. **self.surface = pygame.image.load("graphics/star.png")**
6. self.image = pygame.transform.scale(self.surface, (wd/32, ht/20))
7. self.rect = self.image.get\_rect(center=(wd\*1.1, random.uniform(ht\*0.1, ht\*0.9)))
9. **def** move(self, width):
10. **self.rect.centerx -= width/150**
12. **def** change(self, wd, ht, pickup\_type):
13. *# changes type of pickup, heart will give the player a life rather*
14. *# than points*
15. **if pickup\_type == "heart":**
16. self.type = "heart"
17. self.surface = pygame.image.load("graphics/smallheart.png")
18. self.image = pygame.transform.scale(self.surface, (wd/32, ht/20))
19. **else**:
20. **self.type = "star"**
21. self.surface = pygame.image.load("graphics/star.png")
22. self.image = pygame.transform.scale(self.surface, (wd / 32, ht / 20))
24. **def** reset(self, width, height, hide):
25. **if random.randint(0, 4) == 0:**
26. self.change(width, height, "heart")
27. **else**:
28. self.change(width, height, "star")
30. **if hide:**
31. self.rect.centerx = -100
32. **else**:
33. self.rect.centerx = width\*1.1
34. self.rect.centery = random.uniform(height\*0.1, height\*0.9)
36. **def** update(self, wd, ht, timer, hide):
37. self.move(wd)
38. **if** timer % 600 == 0 **or** hide:
39. self.reset(wd, ht, hide)

This class is used to generate star and heart objects that the player can pick up and receive a benefit to either score or number of lives. The 'move' method is used to move the image horizontally at a fixed speed across the screen. The 'change' method is used to change the type of the pickup, either to a heart or a star by updating the value of the attribute ‘type’. The 'reset' method is used to reset the pickup's position and type, and the 'update' method is used to move the pickup and reset it periodically.

Play() procedure (updated to incorporate pickups):

1. **def** play():
2. star = pygame.sprite.GroupSingle()
3. star.add(Pickup(width, height))
5. **...**
6. **while** play\_game:
7. **if** pygame.sprite.groupcollide(player, star, False, False):
8. **if** star.sprite.\_\_getattribute\_\_("type") == "star":
9. score += 1000
10. **hide\_star = True**
11. **else**:
12. hide\_star = True
13. **if** lives < 5:
14. lives += 1
15. **...**

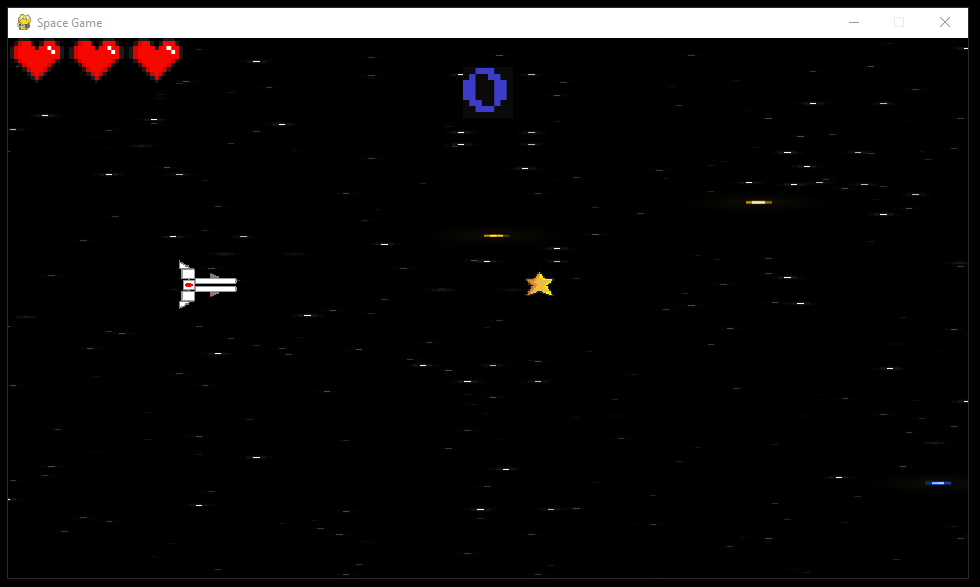
I have decided to make the star sprite a GroupSingle rather than a Group because I don’t plan to have more than one pickup displayed on the screen at once. Instead, there will just be a single pickup object that resets its position and hides itself whenever it collides with the player.   
The parameters (player, star, False, False) are given to the groupcollide() function here because the star sprite will not actually be deleted when the player picks it up.

##### Version 11 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.1a | Testing if the game can still run without errors | Running play() | The pygame window should appear as before without crashing. |  |
| 4.15a | Attempt to collect a star pickup | Colliding the player with the star sprite | Score should increase by 1000 and the star should disappear |  |
| 4.15b | Attempt to collect a life pickup | Colliding the player with the heart sprite | The number of lives should increase by 1 (unless the player is already at 5 lives) and the heart should disappear |  |
| 4.15c | Picking up the life pickup when the player is already at 5 lives | Colliding the player with the heart sprite whilst at five lives | The pickup should disappear, and the number of lives should stay the same. |  |

##### Version 11 Results:

4.1a:



The pygame window still functions and doesn’t crash.

4.15a:

Video 4.15a

4.15b:

Video 4.15b

4.15c

Video 4.15c

The pickup system has been successfully implemented, as shown by these videos. The lives number clearly doesn’t change when the player attempts to pick up a heart while already on five lives.  
Therefore, this version has been completed and I will move onto adding the final feature to the game.

#### Version 12

The final feature that I will build into the single-player section of the game is a ‘final boss’. This will be a strong alien that has many lives and poses a significant threat to the player. I will use the existing Alien() class to accomplish this.  
I will only create this type of alien towards the end of the final stage and if the player manages to kill this alien, the player will end the game early and receive bonus points.

Alien() class (updated to include boss type):

1. **class** Alien(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, wd, ht, alien\_type):
3. super().\_\_init\_\_()
4. self.type = alien\_type
5. **self.wd = wd**
6. self.ht = ht
8. **if** alien\_type == "normal":
9. self.lives = 3
10. **self.surface = pygame.image.load("graphics/alien1.png").convert\_alpha()**
11. self.image = pygame.transform.scale(self.surface, (wd / 12, ht / 9))
12. self.image.set\_colorkey("white")
13. self.rect = self.image.get\_rect(center=(wd\*1.1, random.uniform(ht\*0.1, ht\*0.9)))
14. **else**:
15. ***# Boss type alien***
16. self.lives = 50
17. self.surface\_list = [pygame.image.load("graphics/boss\_frame\_0.gif"),
18. pygame.image.load("graphics/boss\_frame\_2.gif"),
19. pygame.image.load("graphics/boss\_frame\_3.gif"),
20. **pygame.image.load("graphics/boss\_frame\_4.gif"),**
21. pygame.image.load("graphics/boss\_frame\_5.gif"),
22. pygame.image.load("graphics/boss\_frame\_6.gif"),
23. pygame.image.load("graphics/boss\_frame\_7.gif"),
24. pygame.image.load("graphics/boss\_frame\_8.gif")]
26. self.image = pygame.transform.scale(self.surface\_list[0], (wd / 4, ht / 4))
27. self.rect = self.image.get\_rect(center=(wd \* 1.1, ht \* 0.5))
29. **def** move(self, px, py):
30. **if self.type == "normal":**
31. self.rect.centerx -= self.wd / 500
32. *# player 1 side movement*
33. **if** self.rect.centery < py:
34. self.rect.centery += self.ht / 300
35. **elif self.rect.centery > py:**
36. self.rect.centery -= self.ht / 300
38. **if** px > self.wd / 2:
39. self.rect.centerx -= self.wd / 500
40. **elif px > self.wd / 10:**
41. self.rect.centerx -= self.wd / px
42. **else**:
43. self.rect.centerx -= self.wd / 100
44. **elif** self.type == "boss":
45. **self.rect.centerx -= self.wd/2000**
47. **def** animate(self, timer):
48. n = (timer % 16)//2
49. self.image = pygame.transform.scale(self.surface\_list[n], (self.wd / 4, self.ht / 4))
50. **self.image.set\_colorkey("black")**
52. **def** update(self, playerx, playery, timer):
53. self.move(playerx, playery)
55. **if self.type == "boss":**
56. self.animate(timer)
58. *# Alien death when it goes off-screen or when it's health is 0*
59. **if** self.rect.centerx < 1 **or** self.lives <= 0:
60. **self.kill()**

I have updated the alien class to allow it to generate the boss object as well. One method I have added is the animate() method. This takes the game timer as a parameter and changes the current image of the boss by using the list of .gif frames declared in the constructor. The boss type is also given a lives attribute of 50, meaning it will take 50 hits from the player’s lasers to kill it.

Play() procedure (updated):

1. **def** play():
2. width = 960
3. height = 540
4. play\_game = True
5. **boss\_active = False**
6. game\_timer = 3600
7. text\_delay = 0
8. score = 0
9. lives = 3
10. **level = 3**
11. dif = "NORMAL"
12. inv\_frames = 0
13. pygame.init()
14. screen = pygame.display.set\_mode((width, height))
15. **pygame.display.set\_caption("Space Game")**
17. font1 = pygame.font.Font("graphics/fonts/ARCADE\_I.ttf", round(width / 19))
18. font2 = pygame.font.Font("graphics/fonts/ARCADE\_N.ttf", round(width / 19))
19. level\_text = font2.render("", False, (255, 255, 255), (0, 0, 0))
20. **level\_rect = level\_text.get\_rect(center=(width / 3, height / 2))**
21. message = pygame.sprite.GroupSingle()
22. message.add(Texts(width, height, "word", font2, 0.1))
24. clock = pygame.time.Clock()
26. bg = pygame.sprite.GroupSingle()
27. bg.add(Background(width, height, level))
29. player = pygame.sprite.GroupSingle()
30. **player.add(Player(width, height))**
31. laser = pygame.sprite.Group()
32. enemies = pygame.sprite.Group()
33. aliens = pygame.sprite.Group()
34. badlaser = pygame.sprite.Group()
35. **player1\_lives = pygame.sprite.GroupSingle()**
36. player1\_lives.add(Hearts(width, height, True))
37. star = pygame.sprite.GroupSingle()
38. star.add(Pickup(width, height))
40. **score\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()**
41. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))
43. cooldown = 0
44. alien\_cooldown = 10
45. **game\_state = 1**
46. hide\_star = False
48. **while** play\_game: *# Game loop*
49. **for** event **in** pygame.event.get():
50. **if event.type == pygame.QUIT:**
51. pygame.quit()
52. exit()
53. **if** event.type == pygame.KEYDOWN **and** (game\_state == 1):
54. **if** event.key == pygame.K\_ESCAPE:
55. **text\_delay = 40**
56. game\_state = 10
57. **if** event.type == pygame.KEYDOWN **and** (game\_state == 10):
58. **if** event.key == pygame.K\_ESCAPE **and** text\_delay <= 0:
59. game\_state = 1
61. keys = pygame.key.get\_pressed()
62. *# Play game*
63. **if** game\_state == 1:
64. *# Gameplay*
65. **bg\_changed = False**
66. hide\_star = False
67. game\_timer -= 1
68. keys = pygame.key.get\_pressed()
70. **if game\_timer <= 1:**
71. game\_timer = 3600
72. text\_delay = 0
73. level\_text = font2.render("", False, (255, 255, 255), (0, 0, 0))
74. i = 0
75. **if level == 3: *# Final level***
76. game\_state = 9
77. **else**:
78. game\_state = 4
79. level += 1
81. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
82. laser.add(Lasers(player.sprite.rect.centerx, player.sprite.rect.centery, width, height))
83. **if** dif == "NORMAL":
84. cooldown = 10
86. cooldown -= 1
88. *# Collision Detection*
89. **if** pygame.sprite.groupcollide(laser, enemies, True, True):
90. **score += 100**
92. **if** pygame.sprite.groupcollide(player, star, False, False):
93. **if** star.sprite.\_\_getattribute\_\_("type") == "star":
94. score += 1000
95. **hide\_star = True**
96. **else**:
97. hide\_star = True
98. **if** lives < 5:
99. lives += 1
101. **if** (pygame.sprite.spritecollide(player.sprite, enemies, False) **or**
102. pygame.sprite.spritecollide(player.sprite, badlaser, False) **or**
103. pygame.sprite.spritecollide(player.sprite, aliens, False)) **and** inv\_frames <= 0:
104. lives -= 1
105. **inv\_frames = 120**
107. *# Adding enemies*
108. **if** game\_timer % 352 == 0 **and** game\_timer > 250:
109. attack\_pattern1(enemies, width, height, random.randint(0, height))
111. **if** game\_timer % 401 == 0 **and** game\_timer > 250:
112. attack\_pattern2(enemies, width, height, random.randint(0, height))
114. **if** game\_timer % 547 == 0 **and** game\_timer > 250:
115. **attack\_pattern3(enemies, width, height, random.randint(0, height))**
117. **if** random.randint(0, game\_timer + 1000) <= 50 **and** game\_timer > 250:
118. badlaser.add(EnemyBullets(width, height, random.uniform(height \* 0.1, height \* 0.9)))
120. ***# Alien hit detection***
121. **for** n **in** laser:
122. **for** alien **in** aliens:
123. **if** pygame.sprite.collide\_rect(n, alien):
124. n.kill()
125. **alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)**
126. **if** alien.\_\_getattribute\_\_("lives") <= 0:
127. score += 500
129. **if** game\_timer % 200 == 0:
130. **aliens.add(Alien(width, height, "normal"))**
132. **else**:
133. **for** alien **in** aliens:
134. **if** alien\_cooldown <= 0 **and** alien.\_\_getattribute\_\_("type") == "normal":
135. **badlaser.add(EnemyLasers(alien.rect.centerx, alien.rect.centery, width, height))**
136. alien\_cooldown = 30
137. **if** (alien.\_\_getattribute\_\_("type") == "boss") **and** ((0 >= boss\_timer >= -150) **or**
138. (-250 >= boss\_timer >= -2000)) **and** \
139. boss\_timer % 3 == 0:
140. **badlaser.add(EnemyLasers(alien.rect.centerx,**
141. alien.rect.centery + (height / random.randint(7, 9)), width, height))
142. badlaser.add(EnemyLasers(alien.rect.centerx,
143. alien.rect.centery - (height / random.randint(7, 9)), width, height))

146. **if** game\_timer == 1800 **and** level == 3:
147. aliens.add(Alien(width, height, "boss"))
148. boss\_active = True
149. boss\_timer = 300
151. alien\_cooldown -= 1
152. invincibility(inv\_frames, player.sprite)
153. ***# Boss death check***
154. **if** boss\_active:
155. **boss\_timer -= 1**
156. **if** len(aliens) == 0:
157. boss\_active = False
158. score += 5000
159. boss\_timer = 200
160. **elif level == 3 and game\_timer < 1800:**
161. boss\_timer += 1
162. **if** boss\_timer > 400:
163. game\_state = 9
165. ***# Draw background first***
166. bg.draw(screen)
167. bg.update(width)
169. inv\_frames -= 1
170. ***# Sprites drawing and updating***
171. laser.draw(screen)
172. laser.update()
173. aliens.update(player.sprite.rect.centerx, player.sprite.rect.centery, game\_timer)
174. aliens.draw(screen)
175. **enemies.draw(screen)**
176. enemies.update()
177. star.draw(screen)
178. star.update(width, height, game\_timer, hide\_star)
179. badlaser.draw(screen)
180. **badlaser.update()**
182. **if** lives > 0:
183. player.draw(screen)
184. player.update(lives)
185. **else:**
186. game\_state = 2
187. screen.blit(score\_surface, score\_rect)
188. score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
189. player1\_lives.update(lives, inv\_frames)
190. **player1\_lives.draw(screen)**
192. **elif** game\_state == 2:
193. *# Game over screen*
194. screen.fill("black")
195. **screen.blit(score\_surface, score\_rect)**
196. message.update("GAME OVER", font1, 0.2, True)
197. message.draw(screen)
199. *# Level transition screen*
200. **elif game\_state == 4:**
201. screen.fill((0, 0, 0))
202. **if** **not** bg\_changed:
203. bg.empty()
204. bg.add(Background(width, height, level))
205. **bg\_changed = True**
207. text = "LEVEL " + str(level)
208. **if** text\_delay >= 10:
209. **if** i <= len(text):
211. level\_text = font2.render(text[:i], False, (255, 255, 255), (0, 0, 0))
212. i += 1
213. text\_delay = 0
214. **elif** text\_delay >= 50:
215. **game\_state = 1**
216. text\_delay += 1
217. screen.blit(level\_text, level\_rect)
219. **elif** game\_state == 9:
220. ***# Win screen***
221. keys = pygame.key.get\_pressed()
222. game\_timer += 1
224. screen.fill("black")
225. ***# score\_rect = score\_surface.get\_rect(center=(width / 2, height / 1.5))***
227. **if** game\_timer % 40 >= 20:
228. message.update("You win!", font2, 0.2, False)
229. **else**:
230. **message.update("You win!", font2, 0.2, True)**
232. message.draw(screen)
233. screen.blit(score\_surface, score\_rect)
235. **laser.empty()**
236. aliens.empty()
237. badlaser.empty()
238. enemies.empty()
239. *# save score function to go here*
241. *# Pause screen*
242. **else**:
243. text\_delay -= 1
244. message.update("PAUSED", font1, 0.15, True)
245. **message.draw(screen)**
246. pygame.display.update()
247. clock.tick(60) *# Caps at 60 fps*

The main game loop now will create an alien object with the type attribute set to “boss” if the game is on level 3 and the timer is below a certain amount. This is because the boss enemy should only appear towards the end of the final stage. If the player manages to reduce the number of lives the boss has to <=0, then boss active will be set to False again and the game\_state will be updated to the value 9 (the win screen).

##### Version 12 Testing

For the testing of this section, I will immediately set the in-game level to 3 and the timer to 2000 in order to immediately get to the boss for debugging purposes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.16a | Testing the boss animation | Running play() | The boss should appear shortly and move towards the left of the screen. The animation should be functioning. |  |
| 4.16b | Testing the boss’ attack | With the boss visible, colliding the player with the boss’ lasers if they exist | The boss should fire a large number of lasers down the middle of the screen. These should damage the player upon collision. |  |
| 4.16c | Attempting to kill the boss alien | Landing 50 hits on the boss | The boss should eventually die when lasers collide with it. The player should be rewarded with 5000 points when this happens and the victory screen should appear shortly after. |  |

##### Version 12 Results:

4.16b:

Video 4.16b

4.16c:

Video 4.16c

The expected results for 4.16b and 4.16c can be seen in the corresponding videos. The boss lasers work exactly like alien lasers and are capable of damaging and killing the player. The win screen appearing shortly after killing the boss is also an intentional feature of the game.

Both videos also show the boss animation behaving as expected, with the windows of the spaceship moving. Therefore, test 4.16a was successful.

With the successful implementation and testing of this new feature, I can develop and test the next stage of the project, the two-player mode.

### 3.2.4.3 Building the Two-Player Mode

Since I have already coded the majority of the classes and the fact that I will be able to reuse sprites and assets from the single-player mode, this section where I develop the two-player ‘versus’ mode should be much shorter. I will collect feedback from the stakeholders part-way through building this part of the game in order to check that it meets their expectations and to see how it can be improved further.

#### Version 1

For this version, I will create the new classes needed and update the play() procedure to include the versus mode game state and any additional sprites needed.

PlayerA() class:

1. **class** PlayerA(Player):
2. **def** player\_input(self): *# Player 1 ship movement from input*
3. keys = pygame.key.get\_pressed()
4. dx = keys[pygame.K\_d] - keys[pygame.K\_a]
5. **dy = keys[pygame.K\_s] - keys[pygame.K\_w]**
6. self.direction = pygame.math.Vector2(dx, dy)
7. *# Accounting for diagonal speed by dividing by root 2*
8. **if** dx != 0 **and** dy != 0:
9. self.direction /= 1.41421
11. **if** self.rect.centery < 0 **and** keys[pygame.K\_w]:
12. self.direction = pygame.math.Vector2(self.direction.x, 0)
14. **if** self.rect.centery > self.ht/2.2 **and** keys[pygame.K\_s]:
15. **self.direction = pygame.math.Vector2(self.direction.x, 0)**
17. *# Focus active when shift is held*
18. **if** keys[pygame.K\_LSHIFT]:
19. self.position += self.direction \* 4
20. **else:**
21. self.position += self.direction \* 8
23. **if** self.rect.left > self.wd:
24. self.rect.right = 0
25. **self.position.x = self.rect.x**
26. **elif** self.rect.right < 0:
27. self.rect.left = self.wd
28. self.position.x = self.rect.x
30. ***# Set rect position to position vector***
31. self.rect.x = round(self.position.x)
32. self.rect.y = round(self.position.y)
34. **def** update(self, lives):
35. **self.player\_input()**
36. self.death\_check(lives)

This PlayerA() class inherits from the base player class, but the player\_input() method is overwritten in order to fit the versus mode. The main change is to prevent the player 1 sprite (positioned at the top of the screen) from being able to move to the bottom of the screen. This is done by including a check that the y coordinate of the player is not greater than the height of the window / 2.2. If this is the case, then the y component of the vector ‘direction’ is set to 0.

PlayerB() class:

1. **class** PlayerB(pygame.sprite.Sprite): *# 2nd ship for versus*
2. **def** \_\_init\_\_(self, wd, ht):
3. super().\_\_init\_\_()
4. self.wd = wd
5. **self.ht = ht**
6. self.image\_sprite = pygame.image.load("graphics/ship1b.png")
7. self.image\_inv = pygame.image.load("graphics/shipInv.png")
8. self.image = pygame.transform.scale(self.image\_sprite, (wd / 11, ht / 11))
9. self.rect = self.image.get\_rect(center=(wd / 10, ht / 1.25))
10. **self.position = pygame.math.Vector2(self.rect.x, self.rect.y)**
12. **def** player\_input(self): *# Player 2 input*
13. keys = pygame.key.get\_pressed()
14. dx = keys[pygame.K\_RIGHT] - keys[pygame.K\_LEFT]
15. **dy = keys[pygame.K\_DOWN] - keys[pygame.K\_UP]**
16. self.direction = pygame.math.Vector2(dx, dy)
17. *# Accounting for diagonal speed by dividing by root 2*
18. **if** dx != 0 **and** dy != 0:
19. self.direction /= 1.41421
21. **if** self.rect.centery < self.ht/1.8 **and** keys[pygame.K\_UP]:
22. self.direction = pygame.math.Vector2(self.direction.x, 0)
24. **if** self.rect.centery > self.ht **and** keys[pygame.K\_DOWN]:
25. **self.direction = pygame.math.Vector2(self.direction.x, 0)**
27. *# Focus active when control is held*
28. **if** keys[pygame.K\_LCTRL]:
29. self.position += self.direction \* 4
30. **else:**
31. self.position += self.direction \* 8
33. **if** self.rect.left > self.wd:
34. self.rect.right = 0
35. **self.position.x = self.rect.x**
36. **elif** self.rect.right < 0:
37. self.rect.left = self.wd
38. self.position.x = self.rect.x
40. ***# Set rect position to position vector***
41. self.rect.x = round(self.position.x)
42. self.rect.y = round(self.position.y)
43. ...

I have made a separate class in order to generate the player 2 sprite. This is because it loads a different image than the player 1 sprite as it has a different appearance. The input controls for this class are also modified for a second player using the arrow keys rather than WASD. The other methods are similar to the player class.

Play() procedure (versus mode state and updated initialisation):

1. **def** play():
2. pygame.init()
3. width = 960
4. height = 540
6. screen = pygame.display.set\_mode((width, height)) *# Game window*
7. pygame.display.set\_caption("Space Game")
8. clock = pygame.time.Clock()
10. **font1 = pygame.font.Font("graphics/fonts/ARCADE\_I.ttf", round(width / 19))**
11. font2 = pygame.font.Font("graphics/fonts/ARCADE\_N.ttf", round(width / 19))
12. score = 0
13. level = 1
14. game\_timer = 3600
16. div\_rect = pygame.Rect(0, height / 2.1, width \* 2, height / 25) *# Dividing line for versus mode*
18. text\_surface = font1.render("SPACE GAME", True, (180, 10, 10))
19. text\_rect = text\_surface.get\_rect(center=(width / 2, height / 8))
21. level\_text = font2.render("", False, (255, 255, 255), (0, 0, 0))
22. level\_rect = level\_text.get\_rect(center=(width / 3, height / 2))
24. score\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()
25. **score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))**
27. score2\_surface = font2.render(str(score), True, (60, 60, 200)).convert\_alpha()
28. score2\_rect = score2\_surface.get\_rect(center=(width / 2, height / 1.5))
29. timer\_surf = font2.render(str(game\_timer // 60), True, (20, 200, 20)).convert\_alpha()
30. **timer\_rect = timer\_surf.get\_rect(center=(width / 2, height / 2))**
32. bg = pygame.sprite.GroupSingle()
33. bg.add(Background(width, height, 1))
35. **player = pygame.sprite.GroupSingle()**
36. player.add(Player(width, height))
37. player1 = pygame.sprite.GroupSingle()
38. player1.add(PlayerA(width, height))
39. player2 = pygame.sprite.GroupSingle()
40. **player2.add(PlayerB(width, height))**
41. star = pygame.sprite.GroupSingle()
42. star.add(Pickup(width, height))
44. player1\_lives = pygame.sprite.GroupSingle()
45. **player1\_lives.add(Hearts(width, height, True))**
46. player2\_lives = pygame.sprite.GroupSingle()
47. player2\_lives.add(Hearts(width, height, False))
49. message = pygame.sprite.GroupSingle()
50. **message.add(Texts(width, height, "word", font2, 0.1))**
52. laser = pygame.sprite.Group()
53. laser2 = pygame.sprite.Group()
54. enemies = pygame.sprite.Group()
55. **aliens = pygame.sprite.Group()**
56. badlaser = pygame.sprite.Group()
58. dif = "NORMAL"
60. **if dif == "EASY":**
61. lives = 5
62. lives\_b = 5
63. **elif** dif == "NORMAL":
64. lives = 3
65. **lives\_b = 3**
66. **else**:
67. lives = 2
68. lives\_b = 2
70. ***# initialise variables***
72. score2 = 0
73. inv\_frames = 0
74. inv\_frames\_b = 0
75. **cooldown = 0**
76. cooldown\_b = 0
77. alien\_cooldown = 10
78. game\_state = 5 *# temp set game state to versus mode for testing*
79. start\_delay = 30
81. set\_delay = 10
82. play\_game = True
83. timer\_change = False
84. saved = False
85. **bg\_changed = False**
86. boss\_active = False
88. **while** play\_game: *# Game loop*
89. ...
90. ***# Versus mode gameplay***
91. **elif** game\_state == 5:
92. keys = pygame.key.get\_pressed()
93. inv\_frames -= 1
94. inv\_frames\_b -= 1
95. **game\_timer -= 1**
97. **if** keys[pygame.K\_SPACE] **and** cooldown < 1: *# Shooting input + max fire rate*
98. laser.add(Lasers(player1.sprite.rect.centerx, player1.sprite.rect.centery, width, height, True))
99. cooldown = 20
100. **cooldown -= 1**
102. **if** keys[pygame.K\_KP0] **and** cooldown\_b < 1: *# Shooting input + max fire rate*
103. laser2.add(Lasers(player2.sprite.rect.centerx, player2.sprite.rect.centery, width, height, True))
104. cooldown\_b = 20
105. **cooldown\_b -= 1**
107. **if** (pygame.sprite.spritecollide(player1.sprite, enemies, False) **and** inv\_frames <= 0
108. **or** pygame.sprite.spritecollide(player1.sprite, aliens, False) **and** inv\_frames <= 0
109. **or** pygame.sprite.spritecollide(player1.sprite, badlaser, False) **and** inv\_frames <= 0):
110. **lives -= 1**
111. inv\_frames = 120
113. **if** (pygame.sprite.spritecollide(player2.sprite, enemies, False) **and** inv\_frames\_b <= 0
114. **or** pygame.sprite.spritecollide(player2.sprite, aliens, False) **and** inv\_frames\_b <= 0
115. **or pygame.sprite.spritecollide(player2.sprite, badlaser, False) and inv\_frames\_b <= 0):**
116. lives\_b -= 1
117. inv\_frames\_b = 120
119. invincibility(inv\_frames, player1.sprite)
120. **invincibility(inv\_frames\_b, player2.sprite)**
122. **if** game\_timer < 1:
123. game\_state = 6
125. **if len(laser2) > 0:**
126. **for** lase **in** laser2:
127. temp = pygame.image.load("graphics/laser.png")
128. temp.fill("#00a6e4")
129. lase.image = pygame.transform.scale(temp, (width / 40, height / 160))
131. **if** pygame.sprite.groupcollide(laser, enemies, True, True):
132. score += 10
134. **if** pygame.sprite.groupcollide(laser2, enemies, True, True):
135. **score2 += 10**
137. *# Alien hit detection*
138. **for** n **in** laser:
139. **for** alien **in** aliens:
140. **if pygame.sprite.collide\_rect(n, alien):**
141. n.kill()
142. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
143. **if** alien.\_\_getattribute\_\_("lives") <= 0:
144. score += 100

147. *# Adding enemies*
148. **if random.randint(0, 50) == 0:**
149. enemies.add(Asteroids(width, height, width \* 1.1, random.uniform(height \* 0.1, height \* 0.5)))
150. enemies.add(Asteroids(width, height, width \* 1.1, random.uniform(height \* 0.6, height \* 0.9)))
152. **if** game\_timer % 300 == 0 **and** game\_timer < 3500:
153. **aliens.add(Alien(width, height, "normal"))**
155. **else**:
156. **for** alien **in** aliens:
157. **if** alien\_cooldown <= 0:
158. **badlaser.add(EnemyLasers(alien.rect.centerx, alien.rect.centery, width, height, True))**
159. **if** get\_setting("difficulty").upper() == "EASY":
160. alien\_cooldown = 60
161. **elif** get\_setting("difficulty") == "NORMAL":
162. alien\_cooldown = 40
163. **else:**
164. alien\_cooldown = 20
166. alien\_cooldown -= 1
168. **score\_surface = font2.render(str(score), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()**
169. score2\_surface = font2.render(str(score2), True, (60, 60, 200), (10, 10, 10)).convert\_alpha()
171. timer\_surf = font2.render(str(game\_timer // 60), True, (20, 200, 20), (0, 0, 50)).convert\_alpha()
173. **screen.fill((0, 0, 0))**
175. screen.blit(score\_surface, score\_rect)
176. screen.blit(score2\_surface, score2\_rect)
178. **enemies.update()**
179. enemies.draw(screen)
180. aliens.update(player1.sprite.rect.centerx, player1.sprite.rect.centery, player2.sprite.rect.centerx,
181. player2.sprite.rect.centery, game\_timer)
182. aliens.draw(screen)
184. player1\_lives.draw(screen)
185. player1\_lives.update(lives, inv\_frames)
186. player2\_lives.draw(screen)
187. player2\_lives.update(lives\_b, inv\_frames\_b)
188. **badlaser.draw(screen)**
189. badlaser.update()
190. laser.draw(screen)
191. laser2.draw(screen)
192. laser.update()
193. **laser2.update()**
195. player1.draw(screen)
196. player1.update(lives)
197. player2.draw(screen)
198. **player2.update(lives\_b)**
200. pygame.draw.rect(screen, "#FFFFFF", div\_rect)
201. screen.blit(timer\_surf, timer\_rect)
203. **if lives == 0 or lives\_b == 0 or game\_timer <= 0:**
204. game\_state = 6
205. ...

I have now updated the play() to include a two player mode. This has been done by including a number of new variables that are counterparts to existing ones from the single-player mode and will be used by the second player. For example, there is the variable ‘lives\_b’ to track the second player’s number of lives, ‘score2’ to track the second player’s score separately etc. This is done because the two players will be playing to compete, so their resources must be tracked separately so the program can determine who wins once the timer reaches 0. Unlike single-player mode, the timer is displayed in the centre of the screen using timer\_surf and timer\_rect. When the decrementing timer reaches 0 or one of the players reaches 0 lives, the game will be taken to a two-player end screen using game\_state = 6.  
A new sprite group called ‘laser2’ has been generated in order to store the second player’s lasers. These still use the Lasers() class and behave a similar way, but will be colour will be blue.   
The second player lives sprite is also separate and must be displayed at the bottom of the screen rather than the top. To accomplish this I have updated the constructor of the Hearts() class to take an additional parameter.

Hearts() class constructor (updated):

1. **class** Hearts(pygame.sprite.Sprite): *# Number of lives UI*
2. **def** \_\_init\_\_(self, wd, ht, plr):
3. super().\_\_init\_\_()
4. self.wd = wd
5. **self.ht = ht**
6. self.images = [pygame.image.load("graphics/hearts1.png"),
7. pygame.image.load("graphics/hearts2.png"),
8. pygame.image.load("graphics/hearts3.png"),
9. pygame.image.load("graphics/hearts4.png"),
10. **pygame.image.load("graphics/hearts5.png"),**
11. pygame.image.load("graphics/hearts1.5.png"),
12. pygame.image.load("graphics/hearts2.5.png"),
13. pygame.image.load("graphics/hearts3.5.png"),
14. pygame.image.load("graphics/hearts4.5.png")]
16. self.image = pygame.transform.scale(self.images[2], (wd/3, ht/10))
17. **if** plr:
18. self.rect = self.image.get\_rect(center=(wd/6, ht/30))
19. **else**:
20. **self.rect = self.image.get\_rect(center=(wd/6, ht/1.05))**
21. ...

The ‘plr’ parameter is a Boolean value indicating whether the hearts object should be generated for player 1. If this is true, then the rect for the object is placed at the top of the screen.   
Else, the object will be played near the bottom of the screen because it will represent the number of lives that the second player has remaining. All other methods for this class are unchanged.

##### Version 1 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.17 | Attempt to run the game window without error | Running play() | The game should run without an error occurring |  |
| 4.18a | Attempt to move player 1 | With the 2-player game running, testing WASD input | The player 1 ship (at the top of the screen) should move. It should not be able to move past the dividing line. |  |
| 4.18b | Attempt to move player 2 | With the 2-player game running, testing arrow keys input | The player 2 ship (at the bottom of the screen) should move. It should not be able to move past the dividing line. |  |
| 4.18c | Attempt to shoot asteroids with player 1 | Firing red lasers with player 1 that collide with asteroid objects | For each hit, player 1’s score should increment by 10 and the asteroid and laser should both be destroyed. |  |
| 4.18d | Attempt to shoot asteroids with player 2 | Firing blue lasers with player 2 that collide with asteroid objects | For each hit, player 2’s score should increment by 10 and the asteroid and laser should both be destroyed. |  |
| 4.18e | Testing if player 1 can take damage | Colliding player 1 with an enemy object | The top lives counter should be decreased by 1 and player 1 should receive invincibility frames. |  |
| 4.18f | Testing if player 2 can take damage | Colliding player 2 with an enemy object | The bottom lives counter should be decreased by 1 and player 2 should receive invincibility frames. |  |
| 4.18g | Attempt to shoot aliens with player 1 | Firing red lasers with player 1 that collide with an alien object | After 3 hits, the alien should be killed and player 1’s score should be incremented by 100. |  |
| 4.18h | Attempt to shoot aliens with player 2 | Firing blue lasers with player 2 that collide with an alien object | After 3 hits, the alien should be killed and player 2’s score should be incremented by 100. | Failed |

##### Version 1 Results:

4.17:

A picture containing diagram

Description automatically generated

As seen here, the pygame window runs without error and the new features display correctly. The second player’s lives and score are positioned in the right place (the bottom half of the screen). The timer in the middle is also displayed correctly.

4.18a:

Video 4.18a

4.18b:

Video 4.18b

4.18c:

Video 4.18c

4.18d/4.18f:

Video 4.18d

4.18d/4.18e:

Video 4.18e

4.18g:

Video 4.18g

4.18h:

Video 4.18h

As seen in video 4.18h, test 4.18h failed because there is no collision detection between player 2’s lasers and the aliens. I know now this is because there is only one alien collision detection loop and it cycles through sprites in the group ‘laser’ not the group ‘laser2’ – the lasers that the second player fires. This can be solved by adding another for loop which cycles through the laser2 spritegroup.

#### Version 2

For this version I will fix the logic error in the previous version. Once this is done, this section will be completed.

Play procedure (fixed):

1. **def** play():
2. ...
3. **while** play\_game:
4. ...
5. **elif game\_state == 5:**
7. *# Alien hit detection for player 2*
8. **for** n **in** laser2:
9. **for** alien **in** aliens:
10. **if pygame.sprite.collide\_rect(n, alien):**
11. n.kill()
12. alien.\_\_setattr\_\_("lives", alien.\_\_getattribute\_\_("lives") - 1)
13. **if** alien.\_\_getattribute\_\_("lives") <= 0:
14. score2 += 100
15. ...

I have now updated the alien hit detection for the two-player game to include laser2 hit detection. When an alien is killed by these lasers, player 2’s score should increment by 100.  
This is the reason why I use the variable ‘score2’ on line 14 instead of ‘score’.

##### Version 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.18h | Attempt to shoot aliens with player 2 | Firing blue lasers with player 2 that collide with an alien object | After 3 hits, the alien should be killed and player 2’s score should be incremented by 100. | Success |

##### Version 2 Results:

4.18h:

Video 4.18h-2

This video shows the blue lasers are now working and can collide with alien objects. The test was also successful because the correct score counter was increased (the bottom score counter was increased by 100).  
With this bug fixed, this stage of developing the two-player mode is complete. I can now consult my stakeholders to see how they think this version of the game can be improved or modified.

### 3.2.4.4 Feedback from Stakeholders on Two-Player Mode

After showing a prototype version of the ‘versus’ mode to my stakeholders I interviewed them.   
I first asked what they thought was good about this section of the game, and many seemed to think that the way scores and lives are tracked separately for the different players was implemented well. They also liked how the second player’s ship is a different colour to the first (blue instead of red), and fires lasers that are also a different colour but behave the same.  
After asking how the game could be improved many wanted a proper implementation of an end victory screen, similar to the win screen in the single player mode. My stakeholders clarified that the end screen should clearly show which player won and display and appropriate message. This is not currently the case in the game.

#### Version 3

For this version I will implement the win screen system for the two-player mode as mentioned above. This should correctly display which player won once the timer ends. The game can also end by one of the players being reduced to 0 lives.

Play() procedure (updated game\_state 6):

1. **def** play():
2. ...
3. **while** play\_game: *# Game loop*
4. ...
5. ***# Versus mode gameplay***
6. **elif** game\_state == 5
8. **if** lives == 0 **or** lives\_b == 0 **or** game\_timer <= 0:
9. game\_state = 6
11. *# Versus mode end screen*
12. **elif** game\_state == 6:
13. keys = pygame.key.get\_pressed()
15. **if lives\_b == 0:**
16. *# Plr 1 victory*
17. screen.fill("red")
18. message.update("Red wins!", font2, 0.2, True)
19. message.draw(screen)
20. **elif lives == 0:**
21. *# Plr 2 victory*
22. screen.fill("blue")
23. message.update("Blue wins!", font2, 0.2, True)
24. message.draw(screen)
25. **else:**
26. **if** score > score2:
27. *# Plr 1 victory*
28. screen.fill("red")
29. message.update("Red wins!", font2, 0.2, True)
30. **message.draw(screen)**
32. **elif** score2 > score:
33. *# Plr 2 victory*
34. screen.fill("blue")
35. **message.update("Blue wins!", font2, 0.2, True)**
36. message.draw(screen)
37. **else**:
38. *# Draw*
39. screen.fill("black")
40. **message.update("Draw", font2, 0.2, True)**
41. message.draw(screen)
43. laser.empty() *# Deletes all sprites on screen*
44. laser2.empty()
45. **enemies.empty()**
46. badlaser.empty()
47. aliens.empty()
49. **if** keys[pygame.K\_RETURN]:
50. **start\_delay = 30**
51. select = 0
52. score = 0
53. lives = 3
54. inv\_frames = 0
55. **inv\_frames\_b = 0**
56. game\_state = 5 *# This will be changed to 0 when the menu is added. For now it resets the versus game.*
57. ...

If the game\_state variable is equal to 5, the code checks if either player has lost all their lives or if the game timer has reached zero. If any of these conditions are met, the game\_state variable is set to 6, which triggers the end screen for the versus mode.

In the elif block for the game\_state variable equal to 6, the code checks if player 2 has lost all their lives. If so, the screen is filled with red colour and a victory message is displayed for player 1. Similarly, if player 1 has lost all their lives, the screen is filled with blue colour and a victory message is displayed for player 2.   
If neither player has lost all their lives, the code compares their scores to determine the winner. If player 1 has a higher score, the victory message is displayed for player 1 and the screen is filled with red color. If player 2 has a higher score, the victory message is displayed for player 2 and the screen is filled with blue colour. If both players have the same score, the game is considered a draw, and the screen is filled with black colour.  
The message sprite is used to display all messages for this part.

Finally, the code empties all the sprites on the screen and resets various game variables, such as the score, lives, and game state. This resets the game to the two-player game mode and starts a new round of gameplay.

##### Version 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 4.19a | Checking the correct victory screen displays when player 2 loses all their lives | Running the game and reducing player 2 to 0 lives | Player 1 win message: “red wins” should appear. |  |
| 4.19b | Checking the correct victory screen displays when player 1 loses all their lives | Running the game and reducing player 1 to 0 lives | Player 2 win message: “blue wins” should appear. |  |
| 4.20a | Testing that the correct victory screen displays when the timer reaches 0 and player 1 has more points | Player 1 having higher score when timer hits 0 | Player 1 win message: “red wins” should appear. |  |
| 4.20b | Testing that the correct victory screen displays when the timer reaches 0 and player 2 has more points | Player 2 having higher score when timer hits 0 | Player 2 win message: “blue wins” should appear. |  |
| 4.21 | Testing that the correct victory screen displays when the timer reaches 0 and players are tied on points | Players having the same score when timer hits 0 | Draw message should appear because neither player has won. |  |

##### Version 3 Results:

4.19a:

Video 4.19a

4.19b:

Video 4.19b

4.20a:

Video 4.19a

4.20b:

Video 4.20b

4.21:

Video 4.21

The videos show that all of the tests for version 3 of the two-player mode have passed. The correct win screens are displayed for the situation and the background is the correct colour.  
With this feature complete, I will now move on to the next section of development – the scoring system for the single player game.

## 3.2.5 Stage 5: Building the Scoring System

Currently, my game has a player score variable, but this is not saved when the player completes the game. I will change this by saving the players score to the database use a highscores module I will build. Scores will be saved under a separate table than usernames and passwords (the Highscores table instead of the Users table).

### Code for the Scoring System

#### Imported modules

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

I have imported sqlite3 in order to be able to access and edit the local database. This will allow me to save scores in a new table. The validation module has been imported in order to validate a score record before it is inserted into the database. This will use the is\_valid\_score() function that was built and tested earlier.  
‘dates’ is a small module with one function used for getting the current date. This module imports datetime in order to do this.

#### Version 1

Get\_date() function:

1. **def** get\_date():
2. **return** datetime.today().strftime("%d/%m/%Y")

This simple function in dates.py gets today’s date and returns it.

Enter\_score() function:

1. **def** enter\_score(name, score, date):
2. val\_d = validation.is\_valid\_date(date)
3. val\_s = validation.is\_valid\_score(score)
4. **if val\_d and val\_s:**
5. con = sqlite3.connect("LoginScores.db")
6. **try**:
7. con.execute('''insert into Highscores (Name, Score, Date) values (?, ?, ?)''',
8. (name, score, date))
9. **con.commit()**
10. con.close()
11. **return** "Entered successfully"
12. **except** Exception **as** ex:
13. con.close()
14. **return ex**
16. **else**:
17. **return** "Not valid"

This function will be used to insert a highscore record into the database. It takes the parameters: name, score and date and validates them before entering the record. This ensures that the code is robust as validation is always important.

Create\_h\_table() procedure:

1. **def** create\_h\_table():
2. con = sqlite3.connect("LoginScores.db")
3. con.execute("PRAGMA foreign\_keys = 1")
4. con.execute('''CREATE TABLE IF NOT EXISTS Highscores
5. **(ID INTEGER PRIMARY KEY AUTOINCREMENT,**
6. Name VARCHAR NOT NULL,
7. Score INT NOT NULL,
8. Date TEXT(16) NOT NULL,
9. FOREIGN KEY (Name) REFERENCES Users(Username));''')
10. ***# add default data***
11. **for** i **in** range(5):
12. enter\_score("------", 0, dates.get\_date())
13. con.commit()
14. con.close()

The create\_h\_table() procedure creates the Highscores table in the database. The Fields are Name, Score and Date. Name is marked as a foreign key because it is the primary key field of the Users table. In this table, the primary key is ID, which increments automatically.   
When the table is first created, the default data is also entered. This is just 5 blank names, each with 0 score.

##### Version 1 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Test** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 5.1 | Attempt to create the highscores table | Running create\_h\_table() | The table should be created and the default records should be entered. |  |
| 5.2a | Attempt to enter a valid score | Enter\_score() name = “test\_user1” score = 1000 date = “01/01/2001” | This record should be entered into the database. ‘Entered successfully’ should be returned. |  |
| 5.2b | Attempt to enter an invalid score | Enter\_score() name = “test\_user1” score = -1 date = “01/01/2001” | This record should not be entered into the database. ‘invalid’ message should be returned. |  |
| 5.2c | Attempt to enter with an invalid date | Enter\_score() name = “test\_user1” score = -1 date = “30/02/2001” | This record should not be entered into the database. ‘invalid’ message should be returned. |  |

##### Version 1 Results:

5.1:

Table

Description automatically generated

The table was created successfully, .and the correct default data was entered into this table.

5.2a:

Graphical user interface, text, application, chat or text message

Description automatically generated

Table

Description automatically generated

As shown in this evidence, the correct data was entered into the database and the success message was also returned.

5.2b:

Graphical user interface, text, application, chat or text message

Description automatically generated

5.2c:

Graphical user interface, text, application

Description automatically generated

The attempts to put invalid data into the highscore table did not succeed, meaning the enter\_user() function is working correctly.   
This means that I can begin developing additional functions for the HighscoresData module.

#### Version 2

Reset\_scores() procedure:

1. **def** reset\_scores():
2. con = sqlite3.connect("LoginScores.db")
3. cursor = con.cursor()
4. cursor.execute("DROP TABLE Highscores")
5. **create\_h\_table()**

This procedure deletes the entire Highscores table and recreates it again using the previously built create\_h\_table() procedure.  
I have made this function because it will be used in the admin section to reset the scores back to the default values.

Get\_names() function:

1. *# Returns a list of names in descending score order*
2. **def** get\_names():
3. names = []
4. con = sqlite3.connect("LoginScores.db")
5. **cursor = con.cursor()**
7. cursor.execute("SELECT \* FROM Highscores ORDER BY Score DESC")
8. records = cursor.fetchall()
9. **for** row **in** records:
10. **names.append(row[1])**
12. cursor.close()
13. con.close()
14. **return** names

Get\_scores() function:

1. *# Returns the list of scores in descending score order*
2. **def** get\_scores():
3. scores = []
4. con = sqlite3.connect("LoginScores.db")
5. **cursor = con.cursor()**
7. cursor.execute("SELECT \* FROM Highscores ORDER BY Score DESC")
8. records = cursor.fetchall()
9. **for** row **in** records:
10. **scores.append(row[2])**
12. cursor.close()
13. con.close()
14. **return** scores

These two new functions are used to return a list of the names and scores in descending order of score. I have done this in order to be able to use this later to display a leaderboard of the highest scores.   
This will be done later, in the development of the game’s menus stage of development and testing.

##### Version 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Test** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 5.3 | Attempt to reset the scores | Running reset\_scores() | The table should be recreated, and the default records should be entered. |  |
| 5.4a | Attempt to use get\_scores() to return name list. | Running get\_scores() after entering some test records into the table. | The scores should be returned in descending order in a list. |  |
| 5.4b | Attempt to use get\_names() to return name list. | Running get\_names() | The names should be returned in descending order (of score) in a list. |  |

##### Version 2 Results:

5.3:

Application

Description automatically generated with medium confidence

The table was successfully reset. Once again, the correct default data is initially inserted.

5.4a/ 5.4b:

Text

Description automatically generated

This evidence shows that the get\_scores() and get\_names() functions work and can be used to return the fields in descending order of score.   
This means that all of the functions in the HighscoresData module have been developed and sufficiently tested.  
Since there is no reason to collect stakeholder feedback for this section (since it is back-end and there does not yet exist a corresponding front-end for this section), I will now move forward to the next stage of development. I will build the menus for the game.

## 3.2.6 Stage 6: Building the Game Menus

For this section I will be programming the menus for the game. This will include the game’s main menu, the menu to display the highscores leaderboard, and a menu for settings. The main menu will be the menu the user first sees when they launch the game and will have options that allow the user to start playing both single-player or two-player, and viewing the settings menu or highscore menu.  
The settings menu will contain any settings that the user will be able to adjust. These will be saved to a text file so that settings can be still be loaded upon exiting and relaunching the game.  
The highscores menu will display a leaderboard for the scores arranging in descending order of score, with the players name next to the corresponding score. This will allow the user to view scores from within the game program itself. This will use the already built functions from stage 5 – the scoring system.

### Code for Game Menus

#### Version 1

In this version, I will focus on coding the main menu. I will create a new class that will allow me to generate all of the options that the user will see on the main menu.

Option class()

1. *# Buttons on the main menu*
2. **class** Option(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, variant, wd, ht):
4. super().\_\_init\_\_()
5. **self.cycle = 0**
6. self.timer = 0
7. self.toggle = True
8. self.wd = wd
9. self.ht = ht
10. **if variant == "play":**
11. self.type = "play"
12. self.image\_sprites = [pygame.image.load("graphics/menu/option\_play.png"),
13. pygame.image.load("graphics/menu/option\_play1.png"),
14. pygame.image.load("graphics/menu/option\_play2.png"),
15. **pygame.image.load("graphics/menu/option\_play3.png"),**
16. pygame.image.load("graphics/menu/option\_play4.png")]
18. self.image = pygame.transform.scale(self.image\_sprites[0], (wd / 3, ht / 15))
19. self.rect = self.image.get\_rect(center=(wd/2, ht/2.5))
21. **elif** variant == "settings":
22. self.type = "settings"
23. self.image\_sprites = [pygame.image.load("graphics/menu/option\_settings.png"),
24. pygame.image.load("graphics/menu/option\_settings1.png"),
25. **pygame.image.load("graphics/menu/option\_settings2.png"),**
26. pygame.image.load("graphics/menu/option\_settings3.png"),
27. pygame.image.load("graphics/menu/option\_settings4.png")]
29. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 2.3, self.ht / 15))
30. **self.rect = self.image.get\_rect(center=(wd/2, ht/2.5 \* 1.3))**
31. **elif** variant == "versus":
32. self.type = "versus"
33. self.image\_sprites = [pygame.image.load("graphics/menu/option\_versus.png"),
34. pygame.image.load("graphics/menu/option\_versus1.png"),
35. **pygame.image.load("graphics/menu/option\_versus2.png"),**
36. pygame.image.load("graphics/menu/option\_versus3.png"),
37. pygame.image.load("graphics/menu/option\_versus4.png")]
39. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 1.65, self.ht / 14.8))
40. **self.rect = self.image.get\_rect(center=(wd/2, ht/2.5 \* 1.6))**
41. **elif** variant == "highscores":
42. self.type = "highscores"
43. self.image\_sprites = [pygame.image.load("graphics/menu/option\_highscores.png"),
44. pygame.image.load("graphics/menu/option\_highscores1.png"),
45. **pygame.image.load("graphics/menu/option\_highscores2.png"),**
46. pygame.image.load("graphics/menu/option\_highscores3.png"),
47. pygame.image.load("graphics/menu/option\_highscores4.png")]
49. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 1.9, self.ht / 17))
50. **self.rect = self.image.get\_rect(center=(wd/2, ht/2.5 \* 1.9))**
51. **else**:
52. self.type = "exit"
53. self.image\_sprites = [pygame.image.load("graphics/menu/option\_exit.png"),
54. pygame.image.load("graphics/menu/option\_exit1.png")]
55. **self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 4, self.ht / 20))**
56. self.rect = self.image.get\_rect(center=(wd/2, ht/2.5 \* 2.3))
58. **def** animate(self, select):
59. **if** self.type == "play":
60. **if select == 0:**
61. self.image = pygame.transform.scale(self.image\_sprites[self.cycle], (self.wd / 3, self.ht / 15))
62. **else**:
63. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 3, self.ht / 15))
64. **if** self.type == "settings":
65. **if select == 1:**
66. self.image = pygame.transform.scale(self.image\_sprites[self.cycle], (self.wd / 2.3, self.ht / 15))
67. **else**:
68. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 2.3, self.ht / 15))
69. **if** self.type == "versus":
70. **if select == 2:**
71. self.image = pygame.transform.scale(self.image\_sprites[self.cycle], (self.wd / 1.65, self.ht / 14.8))
72. **else**:
73. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 1.65, self.ht / 14.8))
74. **if** self.type == "highscores":
75. **if select == 3:**
76. self.image = pygame.transform.scale(self.image\_sprites[self.cycle], (self.wd / 1.9, self.ht / 16))
77. **else**:
78. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 1.9, self.ht / 17))
79. **elif** self.type == "exit":
80. **if select == 4:**
81. self.image = pygame.transform.scale(self.image\_sprites[1], (self.wd / 4, self.ht / 20))
82. **else**:
83. self.image = pygame.transform.scale(self.image\_sprites[0], (self.wd / 4, self.ht / 20))
85. **def animate\_cycle(self):**
86. **if** self.toggle:
87. self.cycle += 1
88. **else**:
89. self.cycle -= 1
90. **if self.cycle >= 4:**
91. self.toggle = False
92. **if** self.cycle <= 1:
93. self.toggle = True
95. **def update(self, select):**
96. self.animate(select)
97. self.timer -= 1
98. **if** self.timer <= 0:
99. self.timer = 8
100. **self.animate\_cycle()**

This code defines a class called Option, which creates buttons on the main menu of a game using the Pygame library. The buttons have different variant attributes including play, settings, versus, highscores, and exit.

Each button has a list of images to allow it to be animated. These images are stored the ‘menu’ folder in graphics.

The animate method is responsible for changing the button's image to give it an animation effect when the user hovers over it. It does this by accessing the image\_sprites list and changing the button's image to the corresponding variation based on the button's variant and whether it is currently selected.  
The animate\_cycle function is responsible for cycling through the variations of the button's image by incrementing or decrementing the index of the image\_sprites list.  
The update function is responsible for calling the animate function and updating the button's timer. This timer attribute is used to make the animation progress once every 8 frames instead of every frame.   
I have done it this way in order to slow down the animation to look more normal.

Play() procedure (updated):

1. **def** play():
2. ...
3. text\_surface = font1.render("SPACE GAME", True, (180, 10, 10))
4. text\_rect = text\_surface.get\_rect(center=(width / 2, height / 8))
6. ...
7. options = pygame.sprite.Group()
9. buttons = ["play", "settings", "versus", "highscores", "exit"]
11. **for** i **in** buttons:
12. options.add(Option(i, width, height))
14. start\_delay = 30
16. ...
17. **while** play\_game: *# Game loop*
18. **for** event **in** pygame.event.get():
19. **if** event.type == pygame.QUIT:
20. **pygame.quit()**
21. exit()
22. **if** event.type == pygame.KEYDOWN **and** game\_state == 0:
23. **if** event.key == pygame.K\_UP:
24. **if** select > 0:
25. **select -= 1**
26. **else**:
27. select = 4
29. **if** event.key == pygame.K\_DOWN:
30. **if select < 4:**
31. select += 1
32. **else**:
33. select = 0
35. **if event.type == pygame.KEYDOWN and (game\_state == 1):**
36. **if** event.key == pygame.K\_ESCAPE:
37. game\_state = 10
38. text\_delay = 40
39. **if** event.type == pygame.KEYDOWN **and** (game\_state == 10):
40. **if event.key == pygame.K\_ESCAPE and text\_delay <= 0:**
41. game\_state = 1
42. **if** event.key == pygame.K\_BACKSPACE **and** text\_delay <= 0:
43. game\_state = 0 *# return to the main menu from pause screen*
45. ***# Main Menu***
46. **if** game\_state == 0:
47. keys = pygame.key.get\_pressed()
48. menu\_text = font2.render("MENU", False, (200, 200, 200), (0, 0, 0))
49. menu\_rect = menu\_text.get\_rect(center=(width / 2, height / 4))
50. **screen.fill((0, 0, 0))**
51. screen.blit(menu\_text, menu\_rect)
52. screen.blit(text\_surface, text\_rect)
53. options.update(select)
54. options.draw(screen)
56. **if** (keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]) **and** select == 0 **and** start\_delay <= 0: *# Play game*
57. boss\_active = False
58. dif = "NORMAL"
59. **if** dif == "EASY":
60. **lives = 5**
61. **elif** dif == "NORMAL":
62. lives = 3
63. **else**:
64. lives = 2
66. i = 0
67. text\_delay = 0
68. level = 1
69. game\_timer = 2700 *# 45 seconds*
70. **level\_text = font2.render("", False, (255, 255, 255), (0, 0, 0))**
71. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))
72. score = 0
73. game\_state = 4
74. player.sprite.position.x = height / 2 *# Reset ship pos*
75. **player.sprite.position.y = width / 10**
76. alien\_cooldown = 0
78. **if** (keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]) **and** select == 1 **and** start\_delay <= 0:
79. *# Takes to settings screen*
80. **game\_state = 3**
82. **if** (keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]) **and** select == 2 **and** start\_delay <= 0:
83. *# Takes to versus mode*
84. game\_timer = 3600
85. **game\_state = 7**
86. start\_delay = 60
87. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))
88. score2\_rect = score2\_surface.get\_rect(center=(width / 2, height / 1.5))
90. **if (keys[pygame.K\_RETURN] or keys[pygame.K\_SPACE]) and select == 3 and start\_delay <= 0:**
91. *# Will take to highscores screen*
92. game\_state = 8
94. **if** (keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]) **and** select == 4 **and** start\_delay <= 0:
95. **pygame.quit()**
96. *# Logs off the user*
97. **return** ()
98. start\_delay -= 1
99. ...

The play() procedure has now been updated to include the generation of a main menu buttons using the ‘options’ spritegroup. Before the game loop, the ‘buttons’ list and a for loop are used to do this.  
The game now initialises to game\_state = 0. This is done in order to immediately take the user to the main menu.  
From here, user input can be used to navigate the options and select one. If the user selects the ‘play’ option, they will be taken to the single-player game state. If they select the ‘quit’ option, the game will close etc.  
Some options currently don’t have function, such as the settings option since the settings section has not been completed.  
The variable ‘select’ is used to store the value of the menu option that is currently selected by the user on the main menu. For example, a value of 0 would be the 1st option at the top.

##### Version 1 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Test** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 6.1a | Attempt to run the main menu | Running play() | The main menu should appear with the options and text visible |  |
| 6.1b | Testing if select animation works correctly | Having an option selected on the main menu | The currently selected option should have animated arrows next to it (except the ‘log off’ button should be highlighted instead). |  |
| 6.2a | Attempt to move selected option upwards | Pressing up arrow multiple times with game running | The currently selected option should be updated to the one above it. If it is already at the top, it should ‘loop around’ and select the bottom option |  |
| 6.2b | Attempt to move selected option downwards | Pressing down arrow multiple times with game running | The currently selected option should be updated to the one below it. If it is already at the bottom, it should ‘loop around’ and select the top option |  |
| 6.3a | Attempt to use the ‘play’ option | Pressing enter while ‘play’ is selected | The user should taken to a level display screen showing level 1, then the single-player game |  |
| 6.3b | Attempt to use ‘quit’ button | Pressing enter while ‘quit’ is selected | This should exit the game |  |

##### Version 1 Results:

6.1a:

Text

Description automatically generated

The pygame window still runs without error. This test was successful since all of the options have been generated with the correct images, in the order expected.

6.2a/6.1b:

Video 6.2a

6.2b/6.1b:

Video 6.2b

Both of these videos showcase that the way the user can scroll up or down the option is working as expected. The animated arrows also correctly indicate the current option that is selected.

6.3a:

Video 6.3a

6.3b

Video 6.3b  
  
Since I don’t plan to add any other major features to the main menu, this section of developing the game menus is tested and completed. I will now begin developing the settings menu.

#### Version 2

I will first the back-end side for the settings menu since the front-end side will not be able to function at all without it. This will make testing for this section simpler.  
For the settings, I will use a new module called settings.py.   
This will contain all of the functions to do with editing, loading or saving the settings for the game. These settings are stored in a text file rather than a database because the data stored is simple and there is no need for many structured fields.

Get\_setting() function:

1. *# Dictionary for each line of the text file representing a setting*
2. st\_dict = {"WIDTH": 0, "HEIGHT": 1, "DIFFICULTY": 2, "COLOUR": 3}

5. ***# Returns the value for a given setting***
6. **def** get\_setting(st):
7. **global** st\_dict
9. **try**:
10. **st = st.upper()**
11. lines = []
12. item = ""
14. **with** open("saved\_settings.TXT") **as** f:
15. **for line in f:**
16. lines.append(line.strip())
18. target = lines[st\_dict[st]]
19. i = target.find(":") + 2
20. **while i < len(target):**
21. item += target[i]
22. i += 1
23. **return** item
24. **except** KeyError:
25. **return "Setting not found"**

This defines a dictionary called ‘st\_dict’ which currently contains four keys representing the settings: "WIDTH", "HEIGHT", "DIFFICULTY", and "COLOUR". Each key has a corresponding integer value ranging from 0 to 3.

The get\_setting() function takes a string argument ‘st’ representing a setting and returns its corresponding value from a text file named ‘saved\_settings.TXT’.

The function linearly searches for the target setting by using the dictionary and retrieves its corresponding line from the "lines" list. The function then extracts the value associated with the setting by searching for a colon (:) character in the line and adding the subsequent characters to the "item" string until the end of the line is reached.

If the given setting is not found in the dictionary, the function returns the string ‘Setting not found’. A try except clause is used to ensure that the function is robust.

Save\_setting() procedure:

1. *# Changes a specific setting to a given value*
2. **def** save\_setting(st, data):
3. st = st.upper()
4. lines = []
6. **with** open("saved\_settings.TXT") **as** f:
7. **for** line **in** f:
8. lines.append(line.strip())
10. **target = lines[st\_dict[st]]**
11. i = target.find(":") + 2
12. n = 0
13. new\_item = ""
14. **while** n < i:
15. **new\_item += target[n]**
16. n += 1
17. new\_item += data
18. lines[st\_dict[st]] = new\_item
20. **with open("saved\_settings.TXT", "w") as f:**
21. **for** items **in** lines:
22. f.write(items + "**\n**")

This function takes two paramets: ‘st’ representing the setting to be changed, and ‘data’ representing the new value to be set for this particular setting.

Load\_defaults() procedure:

1. **def** load\_defaults():
2. *# Resets text file to default settings*
3. open("saved\_settings.TXT", "w").close()
4. **with** open("saved\_settings.TXT", "a") **as** f:
5. **f.write("Window width: " + "960" + "\n")**
6. f.write("Window height: " + "540" + "**\n**")
7. f.write("Difficulty: " + "Normal" + "**\n**")
8. f.write("Colourblind mode: " + "False")

This simple procedure opens the settings text file and writes the default values for every setting. This will reset the text file to these defaults regardless of what its contents already are.

##### Version 2 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Test** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 6.4 | Attempt to load default settings to settings text file | Running load\_defaults() | The default values for every setting should be written to the text file. (width: 960, height: 540 etc.) |  |
| 6.5a | Attempt to save a setting | Running save\_setting() with st = “difficulty” and data = “Hard” | The difficulty setting in the text file should be set to “Hard” |  |
| 6.5b | Attempt to load an existent setting | Running get\_setting() with st = “difficulty” | The string “Hard” should be returned. |  |
| 6.5c | Attempt to load a non-existent setting | Running get\_setting() with st = “test” | “setting not found” should be returned |  |

##### Version 2 Results:

6.4:

Graphical user interface, text, application

Description automatically generated

The correct default settings were saved to the text file.

6.5a:

Graphical user interface, text, application

Description automatically generated

The ‘Difficulty’ in the text files was successfully changed to ‘Hard’. This shows that the save\_setting() function works as expected and can edit the settings text document.

6.5b:

Text

Description automatically generated

The function returns the correct value for the “Difficulty” setting.

6.5c:

A screenshot of a computer

Description automatically generated with medium confidence

As expected, the program returns the “setting not found” message when attempting to search for a setting that does not exist in the text document.

#### Version 3

Since the tests for the back-end section of the settings have passed, I can now develop the front-end game menu that will display the current settings and allow the user to change them.   
I will also quickly develop the small module called ‘colour\_changer.py’ which will have one function that will be used to change the colour of an image. This will be used to properly implement the colourblind mode throughout the project.

Change\_hue() procedure:

1. **def** change\_hue(image, n):
2. pixels = pygame.PixelArray(image)
3. **for** x **in** range(image.get\_width()):
4. **for** y **in** range(image.get\_height()):
5. **rgb = image.unmap\_rgb(pixels[x][y])**
6. color = pygame.Color(\*rgb)
7. h, s, l, a = color.hsla
8. color.hsla = (int(h) + n) % 360, int(s), int(l), int(a)
9. pixels[x][y] = color
10. **del pixels**

This procedure will create a pixel array from an image given in the first parameter. The code then iterates through each pixel in the image, gets the RGB value of the pixel and converts this to a HSV colour value and adds the value n to it. The HSV value is then converted back. I have done it this way in order to allow me to apply a colour hue to the entire image since every pixel will be changed.

Settings() class:

1. *# Buttons and animations for the settings file*
2. **class** Settings(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, variant, wd, ht):
4. super().\_\_init\_\_()
5. **self.wd = wd**
6. self.ht = ht
7. self.delay = 100
8. **if** variant == "difficulty":
9. self.type = "dif"
10. **self.image\_list = [pygame.image.load("graphics/menu/setting\_diff.png"),**
11. pygame.image.load("graphics/menu/setting\_diff1.png"),
12. pygame.image.load("graphics/menu/setting\_diff2.png"),
13. pygame.image.load("graphics/menu/setting\_diff3.png")]
15. **self.image = pygame.transform.scale(self.image\_list[0], (wd / 1.6, ht / 17))**
16. self.rect = self.image.get\_rect(midleft=(wd / 12, ht / 2.5 \* 0.8))
17. **elif** variant == "resolution":
18. self.type = "res"
19. self.image\_list = [pygame.image.load("graphics/menu/setting\_res.png"),
20. **pygame.image.load("graphics/menu/setting\_res1.png"),**
21. pygame.image.load("graphics/menu/setting\_res2.png"),
22. pygame.image.load("graphics/menu/setting\_res3.png"),
23. pygame.image.load("graphics/menu/setting\_res4.png")]
25. **self.image = pygame.transform.scale(self.image\_list[0], (wd / 1.2, ht / 17))**
26. self.rect = self.image.get\_rect(midleft=(wd / 12, ht / 2.5 \* 1.2))
27. **elif** variant == "colourblind":
28. self.type = "colour"
29. self.image\_list = [pygame.image.load("graphics/menu/setting\_colour.png"),
30. **pygame.image.load("graphics/menu/setting\_colour\_b.png"),**
31. pygame.image.load("graphics/menu/setting\_colour1.png"),
32. pygame.image.load("graphics/menu/setting\_colour1b.png")]
33. self.image = pygame.transform.scale(self.image\_list[0], (wd / 1.6, ht / 17))
34. self.rect = self.image.get\_rect(midleft=(wd / 12, ht / 2.5 \* 1.6))
35. **get = get\_setting("colour")**
36. **if** get == "True":
37. self.toggle = True
38. **else**:
39. self.toggle = False
40. **self.ignore = False**
41. **elif** variant == "controls":
42. self.type = "controls"
43. self.image1 = pygame.image.load("graphics/menu/controls.png")
44. self.image = pygame.transform.scale(self.image1, (wd/2, ht/6))
45. **self.rect = self.image.get\_rect(midleft=(wd/12, ht/1.2))**
47. **def** action(self, r, c, d):
48. keys = pygame.key.get\_pressed()
49. **if** self.type == "dif":
50. **if r == 0:**
51. temp\_list = ["Easy", "Normal", "Hard"]
52. c = c % 3
53. self.image = pygame.transform.scale(self.image\_list[c+1], (self.wd / 1.6, self.ht / 17))
54. **if** (keys[pygame.K\_SPACE] **or** keys[pygame.K\_RETURN]) **and** d < 1:
55. **save\_setting("difficulty", temp\_list[c])**
56. **else**:
57. self.image = pygame.transform.scale(self.image\_list[0], (self.wd / 1.6, self.ht / 17))
58. **elif** self.type == "res":
59. **if** r == 1:
60. **temp\_list = ["800", "960", "1440", "1920"]**
61. temp\_ht = ["600", "540", "810", "1080"]
62. c = c % 4
63. self.image = pygame.transform.scale(self.image\_list[c+1], (self.wd / 1.2, self.ht / 17))
64. **if** keys[pygame.K\_SPACE] **or** keys[pygame.K\_RETURN]:
65. **save\_setting("width", temp\_list[c])**
66. save\_setting("height", temp\_ht[c])
67. **else**:
68. self.image = pygame.transform.scale(self.image\_list[0], (self.wd / 1.2, self.ht / 17))
69. **elif** self.type == "colour":
70. **if r == 2:**
71. **if** self.toggle:
72. self.image = pygame.transform.scale(self.image\_list[3], (self.wd / 1.6, self.ht / 17))
73. **if** (keys[pygame.K\_SPACE] **or** keys[pygame.K\_RETURN]) **and** **not** self.ignore:
74. save\_setting("colour", "False")
75. **self.toggle = not self.toggle**
76. self.ignore = True
77. **else**:
78. self.image = pygame.transform.scale(self.image\_list[1], (self.wd / 1.6, self.ht / 17))
79. **if** (keys[pygame.K\_SPACE] **or** keys[pygame.K\_RETURN]) **and** **not** self.ignore:
80. **save\_setting("colour", "True")**
81. self.toggle = **not** self.toggle
82. self.ignore = True
83. **if** **not** keys[pygame.K\_SPACE] **and** **not** keys[pygame.K\_RETURN]:
84. self.ignore = False
85. **else:**
86. **if** self.toggle:
87. self.image = pygame.transform.scale(self.image\_list[2], (self.wd / 1.6, self.ht / 17))
88. **else**:
89. self.image = pygame.transform.scale(self.image\_list[0], (self.wd / 1.6, self.ht / 17))
91. **def** update(self, row, col, delay):
92. self.action(row, col, delay)

This is the code I will use to generate the button objects that appear in the settings menu. Like the main menu, these are animated based on what the user is currently selecting.  
The variant attribute is used to specify which setting is being represented (e.g., ‘difficulty’)  
Depending on the variant, the constructor loads different images and transforms them to the appropriate size.  
This class also has an action method that takes three parameters r, c, and d. This method handles the behaviour of the buttons when they are clicked. Depending on the type of setting being represented, the method changes the image displayed on the button and saves the corresponding setting based on the user input.  
Lastly, the class has an update method that calls the action method with the current row, column, and delay values. This method is called in the main game loop to continuously update the buttons' behaviour based on the user's input.

SettingMarker() class:

1. *# Marker to show the currently active setting*
2. **class** SettingMarker(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, row, wd, ht):
4. super().\_\_init\_\_()
5. **self.row = row**
6. self.wd = wd
7. self.ht = ht
8. self.image1 = pygame.image.load("graphics/menu/marker.png")
9. self.image = pygame.transform.scale(self.image1, (self.wd / 26, self.ht / 20))
10. **self.rect = self.image.get\_rect(center=(self.wd / 2, (self.ht / 6) \* self.row + self.ht / 4))**
12. **def** move(self):
13. **if** self.row == 0:
14. col\_dict = {"EASY": 0.4, "NORMAL": 0.52, "HARD": 0.65}
15. **self.rect.centerx = self.wd\*col\_dict[get\_setting("difficulty").upper()]**
16. **else**:
17. col\_dict = {"800": 0.39, "960": 0.53, "1440": 0.68, "1920": 0.83}
18. self.rect.centerx = self.wd\*col\_dict[get\_setting("width")]
20. **def update\_colour(self):**
21. self.image1 = pygame.image.load("graphics/menu/marker.png")
22. **if** get\_setting("colour") == "True":
23. self.image = change\_hue(self.image1, 320)
24. self.image = pygame.transform.scale(self.image1, (self.wd / 26, self.ht / 20))
26. **def** update(self, change):
27. self.move()
28. **if** change:
29. self.update\_colour()

This class will be used to generate markers that indicate to the user the setting that is already active. An object the SettingsMarker() class will be in a different location on the screen depending on the value set for the ‘row’ parameter in the constructor.  
The update\_colour() method updates the colour of the marker based on the colour setting. It loads the marker image, applies a hue change if the colour setting is ‘True’.

Play() procedure (updated):

1. **def** play():
2. ...
3. width = int(get\_setting("width"))
4. height = int(get\_setting("height"))
5. ...
6. settings\_text = font2.render("SETTINGS", False, (200, 200, 200), (0, 0, 0))
7. settings\_rect = settings\_text.get\_rect(center=(width / 4, height / 8))
9. cog\_image = pygame.image.load("graphics/cog.jpg")
10. cog\_surf = pygame.transform.scale(cog\_image, (width / 10, height / 8))
11. back\_image = pygame.image.load("graphics/menu/back.png")
12. back\_surf = pygame.transform.scale(back\_image, (width / 10, height / 10))
13. **back\_surf.set\_colorkey("black")**
15. options = pygame.sprite.Group()
16. settings = pygame.sprite.Group()
17. marker = pygame.sprite.Group()
19. buttons = ["play", "settings", "versus", "highscores", "exit"]
20. set\_buttons = ["difficulty", "resolution", "colourblind", "controls"]
22. **for** i **in** range(2):
23. **marker.add(SettingMarker(i, width, height))**
25. **for** i **in** buttons:
26. options.add(Option(i, width, height))
27. **for** i **in** set\_buttons:
28. **settings.add(Settings(i, width, height))**
30. dif = get\_setting("difficulty").upper()
32. **if** dif == "EASY":
33. **lives = 5**
34. lives\_b = 5
35. **elif** dif == "NORMAL":
36. lives = 3
37. lives\_b = 3
38. **else:**
39. lives = 2
40. lives\_b = 2
42. *# initialise variables*
44. game\_state = 0
45. select = 0
46. set\_row = 0
47. set\_col = 0
48. **start\_delay = 30**
49. set\_delay = 10
50. ...
51. **while** play\_game: *# Game loop*
52. **for** event **in** pygame.event.get():
53. **if event.type == pygame.QUIT:**
54. pygame.quit()
55. exit()
56. **if** event.type == pygame.KEYDOWN **and** game\_state == 0:
57. **if** event.key == pygame.K\_UP:
58. **if select > 0:**
59. select -= 1
60. **else**:
61. select = 4
63. **if event.key == pygame.K\_DOWN:**
64. **if** select < 4:
65. select += 1
66. **else**:
67. select = 0
69. **if** event.type == pygame.KEYDOWN **and** game\_state == 3:
70. **if** event.key == pygame.K\_LEFT:
71. set\_col -= 1
72. **if** event.key == pygame.K\_RIGHT:
73. **set\_col += 1**
74. **if** event.key == pygame.K\_UP **and** set\_row > 0:
75. set\_row -= 1
76. set\_col = 0
77. **if** event.key == pygame.K\_DOWN **and** set\_row < 2:
78. **set\_row += 1**
79. set\_col = 0
80. **if** event.type == pygame.KEYDOWN **and** (game\_state == 1):
81. **if** event.key == pygame.K\_ESCAPE:
82. game\_state = 10
83. **text\_delay = 40**
84. **if** event.type == pygame.KEYDOWN **and** (game\_state == 10):
85. **if** event.key == pygame.K\_ESCAPE **and** text\_delay <= 0:
86. game\_state = 1
87. **if** event.key == pygame.K\_BACKSPACE **and** text\_delay <= 0:
88. **game\_state = 0**
90. *# Main Menu*
91. **if** game\_state == 0:
92. keys = pygame.key.get\_pressed()
93. **menu\_text = font2.render("MENU", False, (200, 200, 200), (0, 0, 0))**
94. menu\_rect = menu\_text.get\_rect(center=(width / 2, height / 4))
95. screen.fill((0, 0, 0))
96. screen.blit(menu\_text, menu\_rect)
97. screen.blit(text\_surface, text\_rect)
98. **options.update(select)**
99. options.draw(screen)
101. hs\_rows.empty()
102. hs\_rows = update\_scores(hs\_rows, width, height)
104. **if** (keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]) **and** select == 0 **and** start\_delay <= 0: *# Play game*
105. saved = False
106. boss\_active = False
107. dif = get\_setting("difficulty").upper()
108. **if dif == "EASY":**
109. lives = 5
110. **elif** dif == "NORMAL":
111. lives = 3
112. **else**:
113. **lives = 2**
115. i = 0
116. text\_delay = 0
117. level = 1
118. **game\_timer = 2700 *# 45 seconds***
119. level\_text = font2.render("", False, (255, 255, 255), (0, 0, 0))
120. score\_rect = score\_surface.get\_rect(center=(width / 2, height / 10))
121. score = 0
122. game\_state = 4
123. **player.sprite.position.x = height / 2 *# Reset ship pos***
124. player.sprite.position.y = width / 10
125. alien\_cooldown = 0
127. **if** (keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]) **and** select == 1 **and** start\_delay <= 0:
128. ***# Takes to settings screen***
129. game\_state = 3
130. set\_delay = 10
131. timer\_change = False
132. ...
134. start\_delay -= 1
136. ...
137. **elif** game\_state == 3:
138. ***# Settings screen***
139. keys = pygame.key.get\_pressed()
140. colour = False
141. set\_delay -= 1
143. ***# Menu input***
144. settings.update(set\_row, set\_col, set\_delay)
145. **if** keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]:
146. colour = True
148. **if keys[pygame.K\_BACKSPACE] or keys[pygame.K\_ESCAPE]:**
149. game\_state = 0
150. set\_row = 0
151. set\_col = 0
152. **if** set\_row == 1 **and** (keys[pygame.K\_SPACE] **or** keys[pygame.K\_RETURN]):
153. **show\_message("Restart required", "Restarting program", 1)**


157. *# Drawing settings screen GUI*
158. **screen.fill((0, 0, 0))**
159. settings.draw(screen)
160. marker.update(colour)
161. marker.draw(screen)
162. screen.blit(settings\_text, settings\_rect)
163. **screen.blit(back\_surf, (width / 1.2, height / 16))**
164. screen.blit(cog\_surf, (width / 2, height / 16))

For the play() procedure, the difficulty is now loaded rather than always being initially set to ‘NORMAL’. This is done using the get\_setting() function that was built earlier in development and tested. A similar thing is also applied to the window width and height, which are now also loaded from the settings text file.  
The settings and marker sprite groups are also created before the game loop, because the number of settings is not dynamic.

In line 136 of play(), the state of the keyboard keys is obtained for the settings menu, which will be used to take input from the user. Line 137 sets a variable called colour to False, which will be used to keep track of whether the user has pressed a certain key. This is done in order to only update the colour of markers on player input, rather than updating them every frame which would be unnecessary and use more computational resources.  
The game's settings are updated based on user input, and checks whether the user has pressed the enter key or space bar (lines 142-143). If so, colour is set to True. If the user presses the backspace key or the escape key (lines 145-146), the game returns to the main menu.  
Lines 149-153 check whether the user has selected the option to change window resolution, and if so, a message is displayed indicating that the program is restarting. The program quits, and then restarts by calling a restart() function with a parameter called name.

For drawing the GUI for the Settings screen, the screen is filled with a black colour and then the various settings are drawn onto the screen including the makers.

##### Version 3 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Test** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 6.6a | Attempt to open settings menu | Selecting “settings” button from the game’s main menu | The settings menu should load with settings, markers and miscellaneous images displayed correctly. |  |
| 6.6b | Attempt to close settings menu | Pressing backspace while in settings menu | User should be taken back to viewing the main menu |  |
| 6.7a | Testing moving the currently selected setting | Pressing arrow keys whilst on settings menu | The highlighting setting (in white) should move, similar to the main menu. |  |
| 6.7b | Testing changing a setting | Selecting the “easy” setting for difficulty | The red marker should be updated to be above “easy” and the settings text file should be updated. |  |
| 6.7c | Testing if the “colourblind mode” functions correctly | Changing the colourblind mode setting to on | The colour of the marker should change to pink. |  |
| 6.7d | Attempt to disable the colourblind mode again | Changing the colourblind mode setting to off | The colour of the marker should change back to red. |  |

##### Version 3 Results:

6.6a:



The settings menu loads without error and looks as I expected. Non-functional images such as the back icon were loaded correctly.

6.6b:



The program successfully returned to the viewing main menu state.

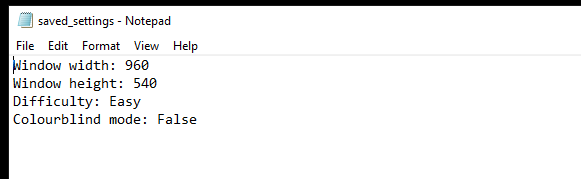
6.7a:



The option currently selected by the user can change row and column correctly.

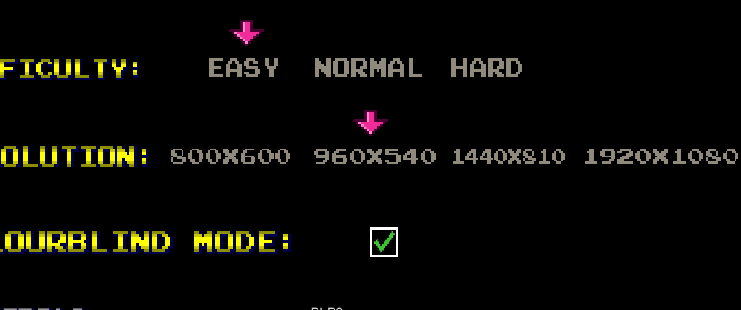
6.7b:





Both the marker, and the settings stored in the text file were updated correctly.

6.7c:



The colours of the markers are updated successfully to be given a pink hue. This shows the change\_colour() function and the settings menu state are both working.

6.7d:



The colour is changed back to the original red without issue. Therefore, the colourblind setting can be toggled without any logic errors occurring.

#### Version 4

Since the previous version for the game menus is working normally, I will add the final two menus needed to complete the code. These menus are the highscore menu, which will be a simple screen to display the highscores using the existing scoring system, and the timer menu for versus which will allow the user to set the time for the two-player game before it starts.

HighscoreRow() class:

1. *# Text that displays the top 5 player's names with corresponding score*
2. **class** HighscoreRow(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, num, wd, ht, name, score):
4. super().\_\_init\_\_()
5. **self.num = num**
6. self.name = name
7. self.score = score
8. y = (ht/3 + num\*ht/10)
9. self.font = pygame.font.Font("graphics/fonts/ARCADE\_R.ttf", round(wd / 24))
10. **self.image = self.font.render(f"{num+1} {name} {score}", False, (200, 200, 200))**
11. self.rect = self.image.get\_rect(midleft=(wd/8, y))
13. **def** update(self):
14. **if** self.num == 0:
15. **self.image = self.font.render(f"{self.num+1} {self.name} {self.score}!", False, (250, 200, 10))**
16. **else**:
17. self.image = self.font.render(f"{self.num+1} {self.name} {self.score}", False, (200, 200, 200))

This class will be used to generate the objects used to display the text (Number, player name and, score) on the highscores menu. If the text is at the top of the leaderboard, it will be shown in a different colour and with an exclamation mark at the end.

Update\_scores() function:

1. **def** update\_scores(hs\_rows, width, height):
2. name\_list = get\_names()
3. score\_list = get\_scores()
5. **for i in range(5):**
6. hs\_rows.add(HighscoreRow(i, width, height, name\_list[i], score\_list[i]))
8. **return** hs\_rows

I have made this function in order to update the scores that are displayed using HighscoreRow objects. It calls the functions get\_names() and get\_scores() to retrieve the names and scores of the players. Then it creates a HighscoreRow object for each of the top 5 players and adds it to the list of HighscoreRow objects. Finally, it returns the list of HighscoreRow objects.

Play() procedure updated:

1. **def** play():
2. ...
3. **elif** game\_state == 8:
4. *# Highscores screen*
5. **keys = pygame.key.get\_pressed()**
6. menu\_text = font2.render("HIGHSCORES", False, (200, 200, 200), (0, 0, 0))
7. menu\_rect = menu\_text.get\_rect(center=(width / 2, height / 7))
9. **if** keys[pygame.K\_BACKSPACE] **or** keys[pygame.K\_ESCAPE]:
10. **game\_state = 0**
12. screen.fill("black")
13. screen.blit(menu\_text, menu\_rect)
14. hs\_rows.draw(screen)
15. **hs\_rows.update()**
16. **...**

This game state for the highscores menu will now draw and update the highscore rows sprite group. There is also text displayed at the top of the screen using the menu\_text variable.

Play() procedure (versus timer screen):

1. **elif** game\_state == 7:
2. *# Versus mode setup*
3. keys = pygame.key.get\_pressed()
4. **if** keys[pygame.K\_UP] **and** **not** timer\_change **and** game\_timer < 7200:
5. **game\_timer += 300**
6. timer\_change = True
7. **if** keys[pygame.K\_DOWN] **and** **not** timer\_change **and** game\_timer > 300:
8. game\_timer -= 300
9. timer\_change = True
10. **if not (keys[pygame.K\_UP] or keys[pygame.K\_DOWN]):**
11. timer\_change = False
13. **if** keys[pygame.K\_BACKSPACE] **or** keys[pygame.K\_ESCAPE]:
14. game\_state = 0
16. aliens.empty()
17. dif = get\_setting("difficulty").upper()
19. **if** dif == "EASY":
20. **lives = 5**
21. lives\_b = 5
22. **elif** dif == "NORMAL":
23. lives = 3
24. lives\_b = 3
25. **else:**
26. lives = 2
27. lives\_b = 2
29. score = 0
30. **score2 = 0**
31. timer\_surf = font2.render(str(game\_timer // 60), True, (20, 200, 20), (0, 0, 50)).convert\_alpha()
32. player1.sprite.rect.center = (width / 10, height / 4)
33. player2.sprite.rect.center = (width / 10, height / 1.25)
35. **screen.fill("black")**
36. screen.blit(timer\_surf, timer\_rect)
37. message.update("Set timer", font2, 0.14, True)
38. message.sprite.rect.centery = height/4
39. message.draw(screen)

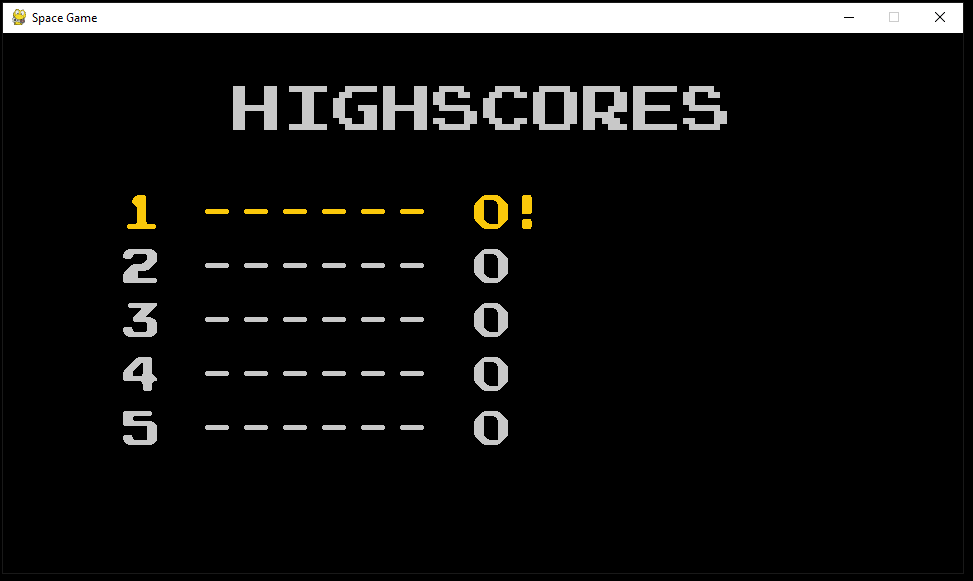
This state will allow the user to setup the timer before entering the two-player section of the game.   
I have done this in order to ensure this part of the game has some customisability for the user by allowing them to change how long the two-player game will last.   
The timer can be changed by user input in increments of 300 frames (5 seconds). This will by displayed on screen using timer\_surf.

##### Version 4 Testing:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Test** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 6.8a | Attempt to open highscores menu | Selecting “highscores” button from the game’s main menu | The highscores menu should load (with default scores displayed as text |  |
| 6.8b | Testing highscores menu with test records | Entering test scores into the highscores table and opening the highscores menu | The highscores menu should load five highest scores from the database and display them in descending order. |  |
| 6.9a | Attempt to display the two-player setup menu | Selecting the “versus mode” option before | The timer should appear before being taken to the two-player game. |  |
| 6.9b | Testing if the timer can be changed | Inputting up or down on timer setup screen | The timer should be incremented or decremented by 5. |  |
| 6.9c | Attempt to reduce timer below minimum value | Decreasing the timer many times | The timer should stop being decreased at 5 seconds. |  |
| 6.10 | Checking the changed timer actually applies to the game | Increasing the time to the maximum of 120 and playing the two-player game | The timer for the game should start at 120 seconds instead of 60 |  |
| 6.11 | Testing that the difficulty settings are applied in gameplay | Checking number of lives displayed in gameplay | Since difficulty has been set to “easy”, the number of lives each player gets should be 5, instead of 3. |  |

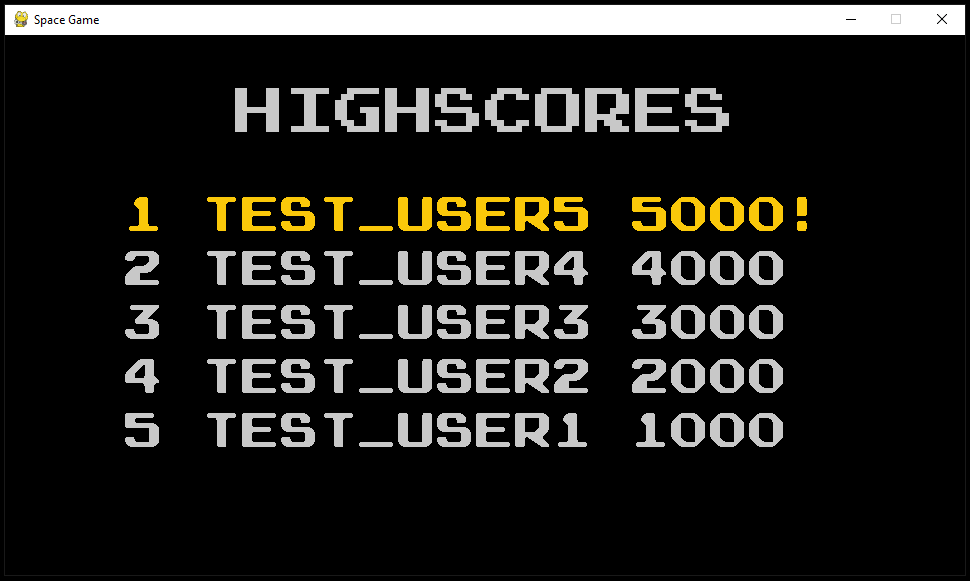
##### Version 4 Results:

6.8a:



As shown in this screenshot, the highscores menu displays correctly (with the default scores loaded to the database).

6.8b:



This shows that the highscores menu works as expected. The correct username is shown next to the corresponding score for that record.

6.9a:



The versus timer menu appears as expected with the timer value displayed in the centre.

6.9b:



The timer is successfully changed by five seconds

6.9c:



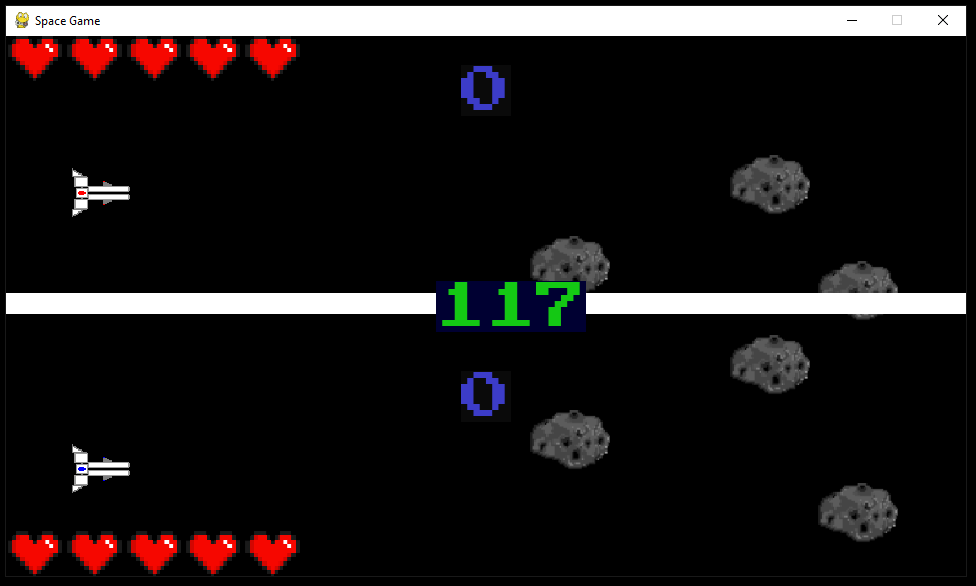
The user is not able to reduce the timer below the minimum value of 5 seconds.

6.9d:



The user is also not able to increase the starting value for the game timer above the maximum value of 120 seconds.

6.10/6.11:



The timer for the game started counting from 120, the value which the user changed it to. This shows the final test for the timer menu is successful. The number of lives has also been changed to 5 because the difficulty has been changed to easy. This shows the settings menu is working correctly and actually has an effect on gameplay.

With this section tested and working, I will now link all parts of the solution together for the final review and corrective actions in 3.3.

## 3.3 Final Review, Improvements and Corrective Actions

For this section I will be linking all parts of the project together that haven’t already been linked.   
I will also collect feedback from stakeholders.

### 3.3.1 Linking Sections Together

#### Linking Admin Panel to Highscores

Updated AdminControl.py:

1. **import** HighscoresData

4. **def** reset\_hs():
5. **HighscoresData.reset\_scores()**
6. show\_message("Done", "Highscores reset!", 1)

The function to reset the highscores using the admin window should now have functionality. HighscoresData is imported for the reset\_scores() procedure that I finished when I was building the scoring system.

#### Linking Game to Scoring system

Updated play() procedure:

1. **def** play(name):
2. ...
3. **while** play\_game:
4. ...
5. **elif game\_state == 2:**
6. *# Game Over screen*
7. **if** **not** saved:
8. enter\_score(name, score, get\_date())
9. saved = True
10. **...**
11. **elif** game\_state == 9:
12. *# Win screen*
13. keys = pygame.key.get\_pressed()
14. ...
15. **if not saved:**
16. enter\_score(name, score, get\_date())
17. saved = True
18. **if** keys[pygame.K\_RETURN]:
19. game\_state = 0
20. **start\_delay = 30**

The play procedure now takes the username as a parameter, allowing it to save the score with the player’s name. This is done using the enter\_score() function that was also completed during the building of the scoring system. The variable ‘saved’ is set to true after entering the score in order to prevent it from being entered multiple times whilst the player is in the ‘game over’ or ‘win’ game state (once every frame). This would hinder the performance of the program as well as adding illegitimate records to the database.

#### Linking Game to Login Section

Updated LoginWindow class:

1. **class** LoginWindow(tk.Tk):
2. ...
3. **def** log\_in(self):
4. username = self.entry1.get()
5. **password = self.entry2.get()**
6. **if** search(str(username), str(password), "Users"):
7. tk.messagebox.showinfo(title='', message="Welcome " + username)
8. LoginWindow.destroy(self)
9. game.setup()
10. **game.play(username)**
11. ...

For the class used to create login windows, I have updated the log\_in() methods to start the game as well as closing the login window.

Updated play() procedure:

1. **def** play(name):
2. ...
3. **while** play\_game: *# Game loop*
4. ...
5. ***# Main Menu***
6. **if** game\_state == 0:
8. **if** (keys[pygame.K\_RETURN] **or** keys[pygame.K\_SPACE]) **and** select == 4 **and** start\_delay <= 0:
9. pygame.quit()
10. ***# Logs off the user***
11. login2 = login.LoginWindow()
12. login2.mainloop()
13. **return** ()
14. start\_delay -= 1
15. ...

The main menu “log off” button is now functioning. This will now create a new LoginWindow() object and exits the game through return.

#### Linking all Settings to Game

Since the difficulty and window size have already been linked to the main game loop, I will focus on implementing the colourblind mode into some game sprites.

Updated Pickup() class:

1. **class** Pickup(pygame.sprite.Sprite):
2. ...
3. **def** update\_colour(self, width, height):
4. **if** get\_setting("colour") == "True" **and** self.type == "heart":
5. **self.image = change\_hue(self.surface, 280)**
6. self.image = pygame.transform.scale(self.surface, (width/32, height/20))
7. ...

I have used the change\_hue() function which I completed earlier in order to add an additional method to the Pickup() class. This method will change the colour of the heart pickups to be more easily visible if colourblind mode is enabled.   
  
Updated Lasers() class:

1. **class** Lasers(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, x, y, wd, ht, sound):
3. ...
4. **if** get\_setting("colour") == "True":
5. **self.image = change\_hue(self.surface, 320)**
6. self.image = pygame.transform.scale(self.surface, (self.wd / 40, self.ht / 160))
7. ...

Similarly, the Lasers() class has been updated to, upon initialisation, change the colour of the object to be more visible if the colourblind mode setting is enabled.

### 3.3.2 Feedback from Stakeholders

I allowed my stakeholders to access a final build of the project, including the login system, game, and score. None of them had any major issues with most of the project in its current state but many had something that they are dissatisfied to not see.  
The game I have made has no sound. There is no music or sound effects for any sections of the game, causing the game to fall short of their expectations for what would be in the final release.

I believe the inclusion of sound/music in my game would be extremely beneficial considering it would not take long to implement and would make my stakeholders much more satisfied with the game.

#### Updated Imports for the Game

Sprites.py:

1. **from** pygame **import** mixer

Game.py:

1. **import** pygame.mixer\_music

Pygame includes a built-in section called ‘mixer’ which allows for audio filed to be loaded and played while a program is running. This is the module I will use in order to implement sound into my game.

#### Updated Sprites

Updated Lasers() class:

1. **class** Lasers(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, x, y, wd, ht, sound):
3. super().\_\_init\_\_()
4. self.wd = wd
5. **self.ht = ht**
6. self.surface = pygame.image.load("graphics/laser.png").convert\_alpha()
7. self.image = pygame.transform.scale(self.surface, (wd / 40, ht / 160))
9. **if** sound:
10. **mixer.set\_num\_channels(10)**
11. mixer.Channel(1).set\_volume(0.5)
12. mixer.Channel(1).play(pygame.mixer.Sound('audio/sound\_shoot.wav'))
13. ...

Upon initialisation, a laser object will now play a short sound file (sound\_shoot.wav). This has been done in order to have a shooting sound effect that plays whenever the player or an enemy fire a laser.   
I have also introduced the ‘sound’ parameter in the constructor which can allow sound to be enabled or disabled for the object.

#### Updated Game

Updated play() procedure:

1. **def** play(name):
2. pygame.init()
3. mixer.init()
4. ...
5. **while play\_game: *# Game loop***
6. ...
7. *# Gameplay*
8. **elif** game\_state == 1:
9. ...
10. **mixer.music.set\_volume(1)**
11. ...
12. *# Level transition screen*
13. **elif** game\_state == 4:
14. screen.fill((0, 0, 0))
16. **if** **not** bg\_changed:
17. mixer.music.load("audio/music\_1.mp3")
18. mixer.music.set\_volume(0)
19. mixer.music.play()
21. **if** level == 1:
22. mixer.music.load("audio/music\_2.mp3")
23. mixer.music.set\_volume(0)
24. mixer.music.play()
25. **...**
26. *# Versus mode gameplay*
27. **elif** game\_state == 5:
29. mixer.music.set\_volume(0.8)
30. **...**
31. **elif** game\_state == 7:
32. *# Versus mode setup*
33. ...
34. mixer.music.load("audio/music\_3.mp3")
35. **mixer.music.set\_volume(0)**
36. mixer.music.play()
38. *# Pause screen*
39. **elif** game\_state == 10:
40. **pygame.mixer\_music.set\_volume(0)**
41. ...

Level one, the final two levels and the two-player mode now all have their own music which is loaded using mixer.music. The volume is also adjusted depending on the game state, which the pause screen (game\_state = 10) completely muting the music.  
Since the game now has music and sound effects, this should be sufficient in order to meet the stakeholders’ expectations for it.

After showing my stakeholders this new version, which has sound completely implemented, they preferred it to the version without sound and music. Because of this, I can now say that the final review and improvement for the project in development and testing is complete.  
This now means that I can compile the project in order to create an executable file that can be run without installing python or additional modules.

Here is the compiled game:

Graphical user interface, text, application, email

Description automatically generated

I will now move on to the next stage for the report – Evaluation.

Chapter Four: Evaluation

## 4.1 Introduction

My overall aim for this project was to create a game that would allow a single player or two-players to control a spaceship and shoot enemies in order to gain score, which is displayed on screen.  
For the game, some of the key objectives were that the game was entertaining to play, it has a feature that can save the user’s score to a database, and that the user is also able to change a variety of settings for the game.

## 4.2 Testing to inform evaluation

Despite having tested all of my modules separately during the development and testing stage of this project, it is still important that I test the system as a whole, including destructive, scenario and beta testing. I will do this according to my test plans in part 2.7 in the design section.

### Beta Testing

I have asked my stakeholders to download a copy of the game. This includes the game’s .exe file as well as the audio and graphics folders. I have emailed my stakeholders the necessary files as well as the following message:  
  
  
Attached is a copy of the game. In addition to trying out the game how you like could you please attempt the following things in no particular order:

* Starting the single player mode
* Moving around and shooting
* Winning the single player mode
* Changing the difficulty
* Setting a highscore and viewing it in-game
* Trying out the two-player mode
* Changing the window size
* Enabling colourblind mode

Make sure you run the game on a computer with Windows 10 installed. If you encounter any bugs in the game, take a screenshot, and send it to me.

After this I asked my stakeholders to fill out a google form with the following questions:

1. Did you have any issues with carrying out the tasks?
2. Was the animation during gameplay smooth enough?
3. Which difficulties did you try out and what did you think of them?
4. Did you find the game too easy/hard at any points?
5. Was the UI for the game menus easy to navigate?
6. What was the performance of the game like?
7. Were you able to play the game after resizing the window?

I have requested my stakeholders to use Windows 10 OS when running the compiled game, because it will only run on windows 10.   
I have asked my stakeholders to carry out a range of tasks and I have deliberately not told my stakeholders how to carry out the tasks, as this will allow me to test whether or not my UI is intuitive.

### 4.2.1 Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Test** | **Description** | **Test data** | **Expected outcome** | **Comment** |
| 7.1 | Attempt to load the game | Run game.exe on Windows 10 OS | Application will load and run |  |
| 7.2 | Attempt to close the window by clicking the X in top right corner. | Clicking X | The game should close immediately |  |
| 7.3a | Attempt to resize the game window in settings | Changing the window resolution in the settings menu | The window should be resized and elements in the window should adjust accordingly |  |
| 7.3b | Attempt to resize the game window | Dragging the sides of the game window | Nothing should happen because the window should not be resizable this way |  |
| 7.4 | Testing the application on different devices (all with windows 10 OS) | Running the game executable file on a laptop as well as running the game on a desktop PC. | The game should run correctly on both as long as windows 10 operating system is installed. |  |
| 7.5a | Attempt an SQL injection | Inputting the string “User OR 1 =1” into both entry boxes | An invalid details error message should appear |  |
| 7.5b | Attempt an SQL injection 2 | Inputting “; DROP TABLE Users” into username | An invalid details error message should appear |  |

#### Results:

7.1:

Graphical user interface, application

Description automatically generated

After logging in:

Text

Description automatically generated

These screenshots show that both the login system and the game can still run.

7.3a:

Text

Description automatically generated

A picture containing text, night sky

Description automatically generated

The game window can be resized, and the sprites are adjusted correctly.

7.3b:

Text

Description automatically generated

The game window size could not be changed by dragging the sides.

7.4:  
Add pictures of laptop/desktop here

7.5a:

Graphical user interface, text, application

Description automatically generated

7.5b:

Graphical user interface, application

Description automatically generated

The SQL injection did not work and the Users table still exists. This means that it was handled correctly.

Graphical user interface, application

Description automatically generated  
The Users table did not get removed.

### 4.2.2 Feedback from Stakeholders

### Beta Testing Results:

#### Question 1 – Did you have any issues with carrying out the tasks?

Lucas Jones and Francis Kip said they had no issue with completing the specified tasks. Lucas Jones additionally stated that the UI was “simple and easy to use”.   
Initially Kurk Milo had been unsure of the controls for the game, but after seeing the controls help in the settings menu, he figured out how to complete the tasks.  
Patrice Hayes stated that she could not figure out the controls for the main menu for some time, so was unable to navigate to the settings menu.

#### Question 2 – was the animation during gameplay smooth enough?

All of the stakeholders stated that the animation during both the single player and two-player games were smooth and the framerate for the game was high enough. No one had any issues for this part.

#### Question 3 – What did you think of the difficulties?

Francis Kip, Kurk Milo, and Patrice Hayes all found all of the difficulty options to accurately represent what they are called. For example, Kurk stated that “the easy difficulty was quite a bit easier than normal” and Francis stated, “playing on hard difficulty made the game much harder to complete”.  
However, Lucas Jones thought that “the game was easier than I expected on hard difficulty”.

#### Question 4 – Did you find the game too easy/hard at any points?

None of my stakeholders weren’t able to complete the game on any difficulty but Patrice Hayes stated that “the final boss enemy is too hard and took me multiple attempts”.   
My other stakeholders did not find the game too easy or too hard in any of the levels.

#### Question 5 – Was the UI easy to navigate?

As discussed before, most of my stakeholders were able to successfully navigate the UI in order to complete the tasks I set them. In addition, Kurk Milo particularly liked the simplicity and style of the main menu, which clearly shows the currently selected option with animated arrows.

#### Question 6 – What was the performance of the game like?

None of my stakeholders had any issue with the performance of the game. This is not surprising since the game is relatively simple and 2D. The game is also compiled instead of run using an interpreter, allowing for higher performance.  
All of my stakeholders were satisfied with the games framerate and Patrice Haye in particular said that “the game did not stutter at any point”.

#### Question 7 – Was the game playable after resizing the window?

Most of my stakeholders had no trouble playing the game after resizing the window and said that the game scaled correctly and fit the new window size.  
However, Kurk Milo said that after selecting the ‘1920x1080’ window resolution for the game it became bugged, and the sides of the window were being cut off.  
This is the screenshot he provided:



After asking what hardware he was playing the game on and what resolution he attempted to play on, I found out that he ran was playing the game displayed on a 1280x1024 monitor and tried to change the resolution to 1920x1080. As seen in the screenshot, this causes the sided of the game to become cropped off, making it impossible to see the health counter and objects at the sides of the window.  
It is important to note that this issue only happened after the resolution was changed from the default.

## 4.3 Evaluation

I am now going to compare the success criteria I created in the analysis section with the final system and justify whether the criteria were met or not.

### Essential Requirements

#### Main menu for the game

A main menu with several options is displayed which allow the user to select any button.

When the user logs in, the main menu for the game appears. This has already been tested and proven to work in test 6.1a, where the main menu can be clearly seen loaded correctly in a screenshot.  
Therefore, the system fully meets this requirement, and the main menu can be used to navigate to the different sections 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.

When the user selects the settings menu from the main menu, they are taken to the settings menu as tested and shown in the testing for version 3 of stage 6 – Building the game menus.   
However, the settings menu is missing the ability for the user to change the audio volume for the game. Every over setting mention has been implemented and functions.  
Therefore, the system partially succeeds at this requirement.

#### Window size is changeable

Game window width and height can be changed. Sprites’ size and position should scale based on window width and height.

The system fully meets this requirement as shown in the evaluation testing, in particular test 7.3a and 7.3b show that the window can be resized to a 4:3 aspect ratio and the sprites size and position scales accordingly. This requirement is also backed up by my stakeholders’ beta testing as all of my stakeholders were able to change the window size from the settings menu.

#### Number of lives displayed during gameplay

The UI should contain a number of hearts that indicate the number of lives the player has left during the game. In 2-player mode, each player’s number of lives must be displayed separately.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

As shown in these screenshots, the number of lives for both the single player game and the two-player mode display as required.   
Therefore, this requirement has been fully met.

#### Controllable ships by the players

For the single player game, there should be one ship that can be moved around the screen using the WASD keys. In 2-player mode, one ship should be controlled with the WASD keys and the other controlled with the arrow keys.

This requirement has been fully met because there is sufficient evidence to show that the players movement is correct in both game modes. This evidence includes the tests result 4.3 and tests 4.18 to highlight the single player and two-player movement respectively.

#### Program must not crash but show error messages

The program must be robust and, when faced with unexpected data or input, reject the data and show a suitable error message instead of just crashing.

For login:

Graphical user interface, application

Description automatically generated

For admin:

Graphical user interface, application, Word, Teams

Description automatically generated

Graphical user interface, application

Description automatically generated

As shown in these screenshots, all sections of the system will show error messages instead of crashing when faced with entered data that is not valid. This was the entire purpose of build the two general modules – 3.2.1.1 (Error handling) and 2.2.1.3 (Validation).

Because of this, it is evident that the system fully meets this requirement for displaying error messages rather than crashing.

#### Intuitive UI / simple design

The user should be able to utilize the system without needing prior knowledge or guidance. The function of a button should be clear by its label and position. Buttons should also look simple and the text should not be too small.

For the beta testing in 4.2, I told my stakeholders to complete a number of tasks within the system, but I did purposely not give them instructions on how to do them. The results of this beta test were that while most were able to navigate the game and do the tasks effectively, not all of my stakeholders could manage this, and the UI is not immediately clear at points. Therefore, the UI for the game could have been made to be more intuitive.

Hence, this requirement is only partially met.

#### High-scores leader board

The top 5 scores achieved must be shown in the highscores screen next to the player who got each score. The text should also be clear and readable.

Text

Description automatically generated

This screenshot shows the highscores menu for the game, which can be accessed at any time by the user from the main menu. My stakeholders and I believe that this menu fits the specified requirements for displaying the top 5 scores, the player’s name as well as being clear.

Therefore, I believe that this requirement has been fully met.

#### The game should run smoothly

The game should cap the framerate at a constant 60 frames per second. There should be no stuttering while playing the game and animations should look smooth.

For Question 6 of 4.2.2, I surveyed my stakeholders to see what they thought of the game’s performance. All of my stakeholders agreed that the game’s animations look smooth and there was no stuttering during gameplay at all. Additionally, the game is simple and 2D so it should not have performance issues even on lower end hardware.  
Therefore, this requirement has been fully met.

### Desirable Requirements

#### Login screen before the game

A login window should exist 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 should be hashed to improve security.

The entire login system and database have already been tested in stage 2 of development and testing.  
Graphical user interface, application

Description automatically generated

Text

Description automatically generated

As shown by these screenshots, the login system will take the user to the game’s main menu if existing user details are entered.

Therefore, this desirable requirement can be considered to be fully met.

#### 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. When the user runs the application again, the colour they selected previously should be loaded.

This has already been tested and proved to be functioning correctly in tests 3.10 and 3.11. These show that the user can easily change the colour using a dropdown menu and a colour picker window. This colour is also saved and loaded when the window is closed and reopened.

Therefore, this requirement has been met fully.

#### Minimal bugs in the game

The game should function completely as intended. Any unintentional visual effects should also be kept to a minimum.

When my stakeholders beta tested the game, they did not find any major bugs or glitches in it.  
However, there is one bug that is in the final release of the game – the window resize from the settings can cause parts of the game to become cropped depending on the monitor it is being displayed on.  
This was outlined in depth in Question 7 of the stakeholders’ beta test results.

Hence, this requirement is only partially successful.

## 4.5 Limitations and Maintenance

A limitation with my database, which is used for both the scores and the user details, is that it is a local database. For my project, this means that if a user has an account and sets a highscore, it could only be accessed from that one device they were playing on when they set that score.   
This could have been improved by hosting the database online, so it could be accessed by any user with a working internet connection. Doing this would have allowed there to be one highscores table that contains the data for any user that set a score whilst having an internet connection. This would let a user to see highest scores set by other people on different devices.

A maintenance issue with the login system is if a user forgets their password, as there is no option for them to reset it.   
For the admin panel, another maintenance issue exists concerning the ability to reset scores. While the admin can reset the entire highscores table, there is no option to remove just a single score. This could be problematic for the future as the system would have to be updated in order to allow for a single score to be deleted.

Since I have built the login section as separate from the game section, if a user just wanted to launch the game without going through the login section this would be easy to implement.  
  
It would also be relatively easy to add new features to my game, as I have used a modular approach and Object-Oriented Programming. My code is also commented, describing what a particular part of the program does. This would be beneficial if a future developer wanted to edit or add to my existing code because the comments would make it easier to understand the program.  
  
An advantage of the game I have distributed is its performance. The game runs quickly since I have compiled it into a single executable file before distributing it. This has the advantage of the user not having to install an interpreter in order to run the code. However, this compiled version will only work on a specific OS – windows 10. This gives the disadvantage of the game not being portable to a lot of devices such as computers running off of MAC or Linux operating systems.

Improvements to the maintainability of the game itself include many values (such as player movement speed and size of spites) being hard coded but adjusting to the window size. This could have been improved by giving the game more settings in the settings, allowing for values such as player speed and size to be changed precisely. In addition, the keys taken for inputs are also hardcoded and fixed, making it not possible for the user to change the key bindings for the game.  
Another improvement that could be made to the game would be giving the user more customisability options for the game’s appearance. For example, a feature could be added to allow the user to change the style of the background that appears during gameplay.

If I was to develop this project again, I would incorporate these improvements, as well as porting the game to multiple platforms rather than just Windows 10 PCs.