For my project I am going to use PyCharm and the python module pygame in order to program my game and to create my graphical user interface.

In my network section I have used json in order to string and destring data that is being sent over the server.

My game is a recreation of the classic arcade game Pacman. In the game Pacman traverses the maze in an attempt to collect all of the points and avoid the artificial intelligence ghosts which are chasing him. Each ghost has a different personality this means they will each have different AI goals. In my game I will also have a multiplayer feature where players are able to play across a network with their friends.

Player Class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **RETURNS** |
| background\_load | photo\_name, image\_x\_length, image\_y\_length | Loads in an image you want as background and scales it to whatever size you want. | background |
| text | font, size, colour, message, background\_colour, screen, x\_position, y\_position | Creates custom text message on screen. | **NONE** |
| button | start\_x\_position, increment\_x, start\_y\_position, increment\_y, dark\_colour, light\_colour, screen, font, font\_size, font\_colour, message, game\_state | Creates text on screen in a box that mimics a button so that when you hover over it the box will change colour and when clicked it will go to the appropriate section. | **NONE** |
| close\_game\_event | **NONE** | When the esc key or close window button is pressed the game closes. | **NONE** |
| menu\_draw\_screen | **NONE** | Draws the background and buttons onto the menu screen | **NONE** |
| singleplayer\_event\_manager | **NONE** | Runs singleplayer.event\_manager | **NONE** |
| singleplayer\_draw\_screen | **NONE** | Runs singleplayer.draw\_screen | **NONE** |
| multiplayer\_event\_manager | **NONE** | Runs multiplayer.event\_manager | **NONE** |
| multiplayer\_draw\_screen | **NONE** | Runs Multiplayer\_draw\_screen | **NONE** |
| draw\_lives | x\_value, increment\_value, player, screen | Draws the amount of lives the player has left as an image of pacman on the screen | **NONE** |
| movement | player | Moves the player if they press depending on the key that they press. | **NONE** |
| end\_program | **NONE** | Sets game loop to false and closes the game | **NONE** |
| log\_in\_screen | **NONE** | Creates an instance of the login screen class and once the player logs in takes them to the menu | **NONE** |
| leaderbaord\_draw\_screen | **NONE** | Handles the leader board | **NONE** |

Singleplayer class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| draw\_lives | x\_value, increment\_value, player, screen | It draws an icon of Pacman on the screen that represents the number of lives remaining | **NONE** |
| event\_manager | player, all\_sprites | Manages all evens that occur during the singleplayer mode for instance if a player wanted to move right | **NONE** |
| draw\_screen | screen, game\_background, all\_sprites | Draws the pacman background onto the screen then also draws options like lives and score onto screen | **NONE** |
| reset | **NONE** | Reset all the ghosts and pacman to their original positions and starting variables | **NONE** |
| reset\_game | **NONE** | Reset all ghosts and pacman to their original start values | **NONE** |

Pacman Class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| update | **NONE** | Updates pacmans position and ensures he stays within the squares in the maze and updates the pacmans image depending on the direction. | **NONE** |
| move\_right | **NONE** | Updates direction to move right | **NONE** |
| move\_left | **NONE** | Updates direction to move left | **NONE** |
| move\_up | **NONE** | Updates direction to move up | **NONE** |
| move down | **NONE** | Updates direction to move down | **NONE** |
| reset | starting\_position | Resets to starting position and starting variable values | **NONE** |
| reset\_game | starting\_position | Entirely restarts all the pacman game variables and pacman location | **NONE** |
| sinlgeplayer\_update | **NONE** | Controls all the singleplayer actions and events | **NONE** |
| multiplayer\_update | **NONE** | Controls all the multiplayer actions and events | **NONE** |
| move | **NONE** | Updates the players position and monitors if they’re going to hit into a wall. | **NONE** |
| force\_pause | **None** | Freezes the player in a position that’s in the centre of the cell. | **None** |
| get\_lives | **NONE** | Returns the amount of lives the player has left | lives |
| update\_lives | lives | Updates the amount of lives the player has left | **NONE** |
| get\_score | **NONE** | Get the players score | score |
| update\_score | score | Updates the players score | **NONE** |
| get\_level | **NONE** | Get the level the player is on | level |
| update\_level | level | Updates the players level | **NONE** |
| get\_timer | **NONE** | Gets the player timer variable | timer |
| update\_timer | timer | Updates the timer variable | **NONE** |
| get\_is\_alive | **NONE** | Returns is\_alive variable | is\_alive |
| update\_is\_alive | is\_alive | Updates the is\_alive variable | **NONE** |
| get\_level\_over | **NONE** | Returns level\_over variable | level\_over |
| update\_level\_over | level\_over | Updates level\_over variable | **NONE** |
| get\_is\_paused | **NONE** | Gets is\_paused variable | is\_paused |
| update\_is\_paused | is\_paused | Updates is\_paused variable | **NONE** |
| get\_orientation | **NONE** | Gets orientation variable | orientation |
| update\_orientation | orientation | Updates orientation variable | **NONE** |
| get\_has\_started | **NONE** | Gets has\_started variable | has\_started |
| update\_has\_started | has\_started | Update has\_started variable | **NONE** |
| get\_update\_direction | **NONE** | Get update\_direction variable | update\_direction |
| update\_update\_direction | update\_direction | Updates update\_direction variable | **NONE** |
| get\_direction | **NONE** | Gets direction variable | direction |
| update\_direction | direction | Updates direction variable | **NONE** |
| get\_pixel\_position | **NONE** | Gets pixel\_position variable | pixel\_position |
| update\_pixel\_position | pixel\_position | Updates pixel\_position variable | **NONE** |

Ghost Class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| update | **NONE** | Updates ghost image depending on variables and direction also implements movement to the shortest path | **NONE** |
| search | cell\_to\_check, end\_location | Recursively loops through the maze checking adjacent positions till it finds the end location | **NONE** |
| movement\_function | visited\_positions | Loops from the end location to the start location in an attempt to find the shortest path. | shortest\_path |
| direction\_calculator | visited\_positions, direction | Calculates the direction needed to move to the next position | direction |
| orange\_ghost\_movement | direction\_input | Provides the movement for the orange ghost who attempts to get two cells ahead of Pacman and cut his off. | position |
| closest\_distance\_calculator | position\_variable, list | Calculates which point in an array of points is closes to your end point | position |
| adjacent\_search | cell\_to\_check, end\_location | Searches each node that’s adjacent and returns all the possible positions in a list. | priority\_queue |
| reset | starting\_position | Resets game variables that have been altered during the course of the game as well as the ghost’ ‘position | **NONE** |
| is\_running | **NONE** | If the ghost has been eaten by Pacman it finds the shortest path back to the ghost home | **NONE** |
| reset\_game | starting\_position | Resets the game variables and ghosts’ positions but also the lists. | **NONE** |
| singleplayer\_update | **NONE** | Has all the necessary checks and updates for the singleplayer version of pacman | **NONE** |
| multiplayer\_update | **NONE** | Has all the necessary checks and updates for the multiplayer version of pacman | **NONE** |
| ghost\_essentials | **NONE** | Has all the events and check and image updates that occur both in the singleplayer and multiplayer versions of pacman | **NONE** |
| force\_pause | **None** | Freezes the ghost in a position that’s in the centre of the cell. | **None** |
| image\_animator | **NONE** | Switches between ghosts images depending on orientation and If they’re alive or frightened | **NONE** |
| get\_change\_mode | **NONE** | Gets change\_mode variable | change\_mode |
| update\_change\_mode | change\_mode | Updates change\_mode variable | **NONE** |
| get\_is\_frightened | **NONE** | Gets is\_frightened variable | is\_frightened |
| update\_is\_frightened | is\_frightened | Updates is\_frightened variable | **NONE** |
| get\_is\_alive | **NONE** | Returns is\_alive variable | is\_alive |
| update\_is\_alive | is\_alive | Updates the is\_alive variable | **NONE** |
| get\_ghost\_clock | **NONE** | Gets ghost\_clock variable | ghost\_clock |
| update­ ghost\_clock | ghost\_clock | Updates ghost\_clock variable | **NONE** |
| get\_start\_pos | **NONE** | Gets start\_pos variable | start\_pos |
| update\_ start\_pos | start\_pos | Updates start\_pos variable | **NONE** |
| get\_end\_pos | **NONE** | Gets end\_pos variable | end\_pos |
| update\_ end\_pos | end\_pos | Updates end\_pos \_pos variable | **NONE** |
| get\_old\_direction | **NONE** | Gets old\_direction variable | old\_direction |
| update\_ old\_direction | old\_direction | Updates old\_direction variable | **NONE** |
| get\_direction | **NONE** | Gets direction variable | direction |
| update\_direction | **direction** | Updates direction variable | **NONE** |
| get\_pixel\_position | **NONE** | Gets pixel\_position variable | pixel\_position |
| update\_pixel\_position | **Pixel\_position** | Updates pixel\_position variable | **NONE** |

Maze class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| create\_walls | maze, outside\_maze | Assigns a grid position to each wall in the maze and returns the value as a 2D array aswell as the positions of all the dots in the game | walls\_list, maze\_pixel\_lis, dots\_list |
| reset | **NONE** | Resets all the walls lists as well as all the special dots and regular dots lists. | **NONE** |
| random\_fruit | dots\_list, fruit\_positions, fruit\_pictures | For singleplayer mode. Has a chance of randomly putting fruits into the maze | **NONE** |

I have a game settings file where all settings for the game are stored and can be easily changed such as the colours screen width and length, Pacmans starting position, fonts etc.

I also have a main file where all the code is run from it also deals with the game state and allows the game to appropriately progress.

Multiplayer Class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| Event\_manager | Player, all\_sprites | Runs the entire pacman multiplayer game. | **NONE** |
| Player\_zero\_function | Input\_player, position, ghost, ghost\_identifier | Runs the shortest path algorithm and direction\_calculator algorithm to send to the server. | **NONE** |
| Player\_one\_function | Input\_player, ghost, ghost\_identifier | Updates all of the ghosts variables depending on the data received from the server. | **NONE** |
| Reset | Player\_id | Rsets all the positions and variables for the ghosts and pacman including their direction | **NONE** |
| Reset\_game | Player\_id | Entirely reset all the players , ghosts and maze back to default | **NONE** |
| Pacman\_dead\_check | Ghost | Checks to see if either of the players have collided with any ghosts. | **NONE** |
| Dot\_check | player | Checks if either player has eaten a pac dot or a special dot | **NONE** |
| Draw\_screen | Screen, background\_image | Controls what is seen on the screen for the entire pacman game | **NONE** |

Network Class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| Client | Connection, player | Threaded client that occurs when a player joins the network. Data is both sent and received here. | **NONE** |

Client\_network Class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| Connect\_to\_server | **NONE** | Connects client to the server and returns their starting position and variables | Client.recv(1024).decode() |
| Send data | Data | Sends data to the server and then also recturns data | Client.recv(1024).decode() |
| Send\_data\_no\_return | Data | Send data to the server but doesn’t receive any data back | **NONE** |
| Return\_data | **NONE** | Returns the players data after they connect to the server | data |

Login Class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| Login | **NONE** | Checks if the user meets the requirements to be logged into the game against the game database | **NONE** |
| Register\_account | **NONE** | Registers a new user if their account meets requirements and saves their data to a database | **NONE** |
| Register | **NONE** | A new register window and allows the user to enter the username and password they want for the account | **NONE** |

Sprite\_object\_class class:

|  |  |  |  |
| --- | --- | --- | --- |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **OUTPUT** |
| Update\_sprite | Images\_to\_update, speed, variable\_x\_size, variable\_y\_size | Loops through multiple images in order to gives animation illusion. Also makes the size of these images smaller in order to fit in the cells | **NONE** |
| Orientation\_images | Direction, right\_images, left\_images, up\_images, down\_images, speed, variable\_x\_size, variable\_y\_size, orientation | Depending on the orientation it displays a different image to the users screen then uses the update sprite function in order to make this image look animated. It also changes the orientation variable depending on the direction | **NONE** |
| Wall\_check | My\_direction, wall\_list, player\_rect | Gets the players pixel position and when they hit a wall it stops the player | My\_direction, pixel\_position |

* Talk about each file

A few key structures that I have in my game include:

Ghost.Priority\_queue – This is a priority queue that is used in the ghosts’ class in the shortest path algorithm the queue is ordered from largest to smallest and the smallest values are taken out and used first in order to find the shortest path.

Two-dimensional array – The maze for the game is represented as a two-dimensional array where the 1s represent all the walls in the maze and the 0s represents all the spaces.

Ghost.visited - A list that stores all the previously visited places.

Dots\_list – An array with an x and y position of every dot that it is in the maze at that time.

My game uses inheritance between the ghost and pacman classes as they share functions

My game also used a database in order to store the score for all players. Once a player has died in their game their score is automatically saved to the database table and if they want to view the leader board, they are able to do so at any time.

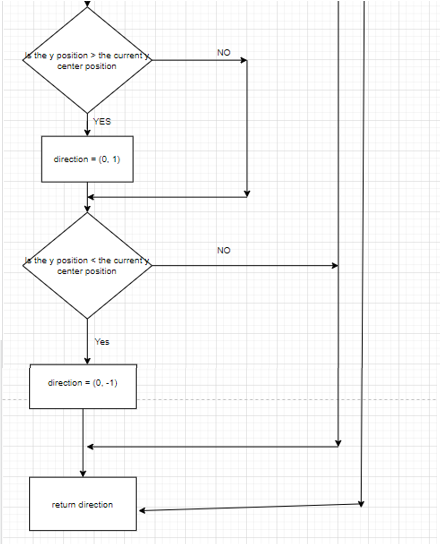
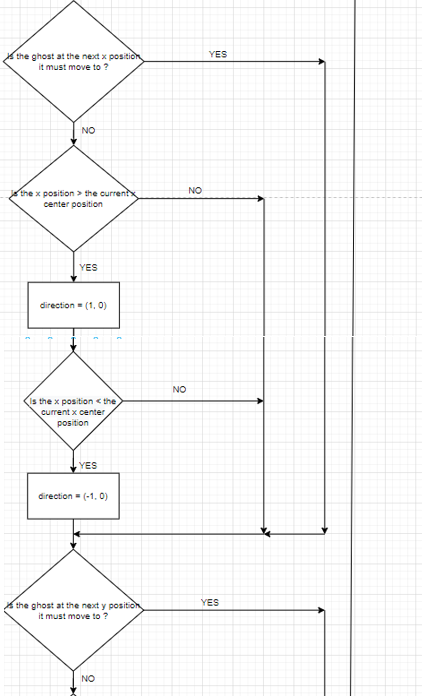
Diagram

Description automatically generatedA\* Shortest Path Algorithm

Shortest Path algorithm:

Diagram

Description automatically generatedMy project uses the A\* shortest path in order to find the shortest path to the player from the ghost and then move to them. The variable cell to check hold the distance of that cell from the end position, the position coordinates and the coordinates of the position it came from. An iterative function is then called and initially checks if this position is the end position if is the end position and the length of the priority queue is empty then it will add that value to the visited list otherwise it will add every value from the priority queue to the visited list. If the cell isn’t equal to the end location then the program will run the adjacent\_search subprocess which checks every cell next to the cell that you are checking ( e.g up, down, left and right) and add the position to the priority queue along with the distance from the end position and the position It came from. The priority queue is then sorted in size order from largest distance from the end position to smallest distance from the end position. If the closest position is already in priority queue the remove it from priority queue, then take the cell\_to\_check(or position) and add this to the visited list. Then take the position with the smallest distance and rerun the algorithm on that position.



Direction Calculator subprocess:

This subprocess is used by the ghost in order to calculate the direction that they need to move in order to catch the player. The subprocess works by first checking if the length of the visited list is == 1. If so, then the ghost must be at its end location. This is used by the ghosts when they’ re in their frightened mode in order to move in random directions in the corner, its also used by the blue ghost in order to move in random directions around the super dot/ pellet his end\_pos variable is reassigned here. If the length of the visited list is then greater than or equal to two then it will attempt to update the direction. If the player is currently in the middle of the cell and depending on if the cell is to his left right up or down his direction will update accordingly.

Diagram

Description automatically generated

The movement function sub process :

The purpose of this sub process is to work backwards from the end position and find all of the nodes/ cells that it has visited an adding them to a separate list. This list is then used by the direction calculator in order to move in the correct direction. It starts first at the end position and then check what cell it came from, this cell must be the shortest path to the end then it continues until it has found the start position.

Diagram

Description automatically generated

Adjacent Search Sub process:

The aim of this subprocess is to find all the cells that are adjacent. If these cells are valid(e.g they aren’t a wall and they haven’t been visited before) then they are added to a priority queue list along with their distance away from the end node in order to calculate estimate the shortest path. This distance is then used later in order to order the priority queue.

Database:

The database file for the game is very simple as It only contains a single entity user.

user( user\_id, username, password, highscore)

The user\_id is the primary key that is used so each player can be uniquely identified.

**Class** **Diagram:**

**Diagram

Description automatically generated**

Class Purposes:

Player

The player class is first instantiated in the main file. It controls the login section of the game. It creates an instance of the Login Class and runs that, then once the user has successfully logged in instances of both Singleplayer and Multiplayer are made. The main purpose of the player class is control what occurs when during the entirety of using the game from login to close. It switches between various even managers and draw screen functions in order to make sure the right evens and correct images are on the screen. The various functions run their appropriate class functions such as singleplayer\_event\_manager running singleplayer.event\_manager. The change in functions is achieved through a variable called state which changes depending on the button input from the user. The player class also controls the leader board where a database request is sent in order to find the top 10 players (if possible) and order them. The player class also hold useful functions for writing on the screen and creating buttons.

Singleplayer

The singleplayer class creates an instance of Pacman which is the player, and various instances of the ghosts (e.g red\_ghost, blue\_ghost, black\_ghost, orange\_ghost) the class also creates an instance of maze which is used during the game. The event manager method contains all the logic for the actual game, it controls what occurs when. Then the draw\_screen method controls what is on the screen at each time and ensure the correct image and sprites are displayed on the screen depending on the situation. Two methods reset and reset game are used, when the player has died to send all the ghosts and the player to their starting position and when the game is over in order to reset all variables and positions. Finally, the class takes in the player class as a parameter in order to update the state variable.

Multiplayer

The multiplayer class creates two instances of pacman (players one and two) it also makes various instances of ghost as in the singleplayer class one for each coloured ghost. It also makes an instance of the client class which contains functions responsible for sending and receiving data to the server. The event manager in this class is not only responsible for game events but also sending data to the server to be updated. The two players are allocated a number by the server 0 and 1. Only player 0 will calculate the shortest path and the ghost position, he will then send this data to the server to be updated which is then taken in by the other player and the data is updated. Player 0 also sends data such as their level etc. The draw\_screen method controls what both players see at any point during the game depending on various variables.

Pacman

The pacman class is an object that all players play as. This class controls the player movement so that when they press a key it moves them in the appropriate direction. As pacman is an acceleration-based game the player is always moving in one direction. Depending on if the game is multiplayer or singleplayer the class will act differently as certain checks have to be made elsewhere in the multiplayer such as if either player has eaten a Pac dot or if there are any dots left. The class also has various reset methods to restart all the variables. This class holds all of pacmans images.

Ghost

The ghost class contains all the algorithm in order to find pacman. Just like the pacman class there are different methods called depending on if the game is singleplayer or multiplayer. Majority of the complex algorithms in this class have been explained earlier.

Moving\_object

This is the superclass of the pacman and ghost class. It contains methods that are used by both classes. Update\_sprites to make the image look animated, orientation\_images to make sure the image is correct to your orientation and a wall check for to see if they have hit a wall or not.

Maze

Upon initialisation a 2D maze is created consisting of 0s and ones in order to represent walls and spaces. The create walls method is used to create a walls list, a list of all the available spaces in the maze and a list of all the positions of the dots. These lists are then used by other classes to represent the dots. The reset method returns all variables to their starting condition and the random fruit method is used only by the singleplayer game once the maze has been made in order to have a chance of random fruit being placed randomly in the maze per level.

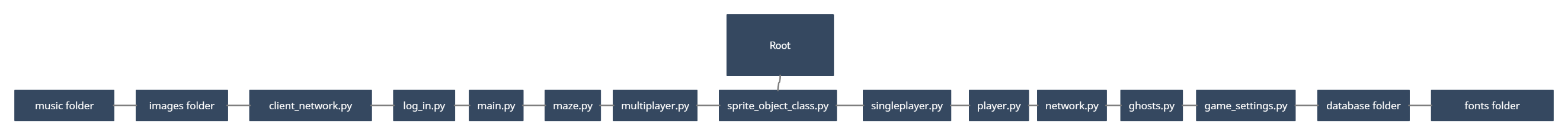
Client

The methods in this class are all used by a client when they communicate with the server. Upon initialisation the class connects to the server and return data.

Login

This class is responsible for creating the login window and allow a player to sign in and also register. The method login is used in order to compare against the data stored in the database and see if the username and password they have entered is for a user and if it is correct. The register\_account method sends the users username and password to the database In order to compare against data there and if the username is unique then the account is created and the data is saved to the database. The register method creates a smaller register window where the user can register an account.

**File Structure**



**User Interface:**

The user interface in my game has been designed based on the needs that were outlined in analysis, they can be summarised as:

* Simple interface that is easy to use by both young and old users.
* Clearly labelled large buttons each with a single purpose.
* Must be graphical.
* Use clear and simple English to not confuse younger users.

Here is how I believe I have met these criteria. First my user interface is graphical, it uses lots of bright colours and images. I have purposefully made it like this due to the large variety in the target age range. Therefore, the font I have decided to use was arial bold in order to make the text stand out I also made the font size very large, the smallest being around 20pts. All of the font on the screen is written to contrast the background so the text stands out more. Furthermore, whenever text was a necessity (e.g buttons) then I made it adamant that I used single phrases such as singleplayer or multiplayer to make the game overall easier to use.

A picture containing graphical user interface

Description automatically generatedThe following diagrams show the design for each window of the game starting with the login screen.

Button made a different colour to everything else on the screen with a question mark to indicate it can be pressed

Title has been coloured a different colour and made bold and large so the user knows to log in here

Large colourful button with one word for clear user input

Graphical user interface

Description automatically generated>Register Screen

Title has been coloured a different colour and made bold and large so the user knows to log in here

Large colourful button with one word for clear user input

>Menu Screen

A picture containing text, slot machine, scoreboard

Description automatically generated

Large buttons with vibrant colours and one word to catch readers eye

A screenshot of a computer screen

Description automatically generated with low confidence>Singleplayer Screen **Figure 1.1**

Buttons become darker when you hover over them to indicate they can be clicked

Large clear text to show lives, level and score

A picture containing text

Description automatically generated>Paused Screen Singleplayer

Large buttons with vibrant colours and one word to catch readers eye

Buttons become darker when you hover over them to indicate they can be clicked

>Multiplayer Waiting Screen

A screen shot of a calculator

Description automatically generated with low confidence

Large bold text on screen to indicate to users theyre waiting for a second player to join

Large clear text to show lives, level my score and player twos score

A picture containing text

Description automatically generated>Multiplayer Screen

This is the screen displayed for a multiplayer game

>Multiplayer Paused

A screen shot of a calculator

Description automatically generated with low confidence

Bold text to indicate the amount of time remaining.

Text to tell you the other player has paused the game

Large clear buttons that stand out

>Multiplayer Paused 2

A screenshot of a computer screen

Description automatically generated with low confidence

Large clear buttons that stand out.

Large test to inform what button to press to resume the game.

>Leader board Screen

Table

Description automatically generated

Large numbers to show each user position.

Large buttons with single phrases for simplicity.

Players usernames in black to contrast alternating background.

Users scores in order for largest to smallest.

Buttons become lighter or darker when hovered over.

Large text

>About Page Screen

Diagram

Description automatically generated

Large bold text opposite colour to screen

Large images

Button that is interactive