Contents

[Analysis 2](#_Toc64835055)

[Design 14](#_Toc64835056)

[Testing 42](#_Toc64835057)

[Evaluation 52](#_Toc64835058)

[Technical Solution 58](#_Toc64835059)

# Analysis

**Introduction**

Games are a famous way to keep us engaged and entertained as well as socialize with friends online. Thus, I have decided to undertake the arcade game Pacman for my project, a game which has not been as famous and popular as it once was many years ago. I would hope to use a large range of the original features from the original Pacman game but alter the AI that is used to control the ghost to make the game more challenging and engaging so users don’t know what to expect. To try and bring more life to the game I will be alerting certain features to make it more challenging and engaging. Machine learning ghosts will also mean that the game will become harder as it progresses as the ghosts become faster. A feature I plan to alter is lives functionality. I also plan to add various forms of random power-ups to make the game unpredictable, these power-ups will make the Pacman stronger in different ways. I would also hope to add a multiplayer network game mode to the game which was not in the original Pacman. I hope that this will make the game more competitive and will draw more people to the game as they can play against their friends. My final aim is to have randomly increasing Pacman maps, however, will be dependent on the time I have available as the Pacman maps have certain Tetris-like shapes to ensure there aren’t any dead-ends in the maze.

**Problem Definition**

For my project, I am hoping to recreate the arcade game Pacman as well as create AI for the 4 ghosts. Pacman is an old-school arcade game where the goal is to collect all of “Pac Dots” and escape all of the various coloured AI ghosts in the generated maze through various levels of difficulty but if the user loses all of their lives the game is over. As well as “Pac Dots” there are special power-up dots that Pacman can eat which will make the ghosts vulnerable.

As Pacman is a single-player arcade game it is limited in the number of players that can play at one given time. To overcome this boundary though I will have a networking multiplayer feature where the user Can play against other friends on the network. Also, because the game is single player the ghosts in the game will be required to be AI.

It would be essential that I add a GUI for my Pacman game. The GUI will first be presented when encouraging the user to sign in, then will move to a menu screen. The user will have an option to choose between One player and Two players. The use of GUI in the game will make it more user-friendly and engaging.

Once they click one of the game options and the game starts the user will use the arrow keys to navigate through a generated maze the main aim of the game is for the user to collect all of the “Pac Dots” whilst also trying to avoid all of the ghosts. The users “Pac Dots” will serve as their game points and this data will be stored in a database and will be used for the scoreboard. The user will be given a starting amount of 3 lives however they can gain more during the game and gain various powerups to make the game easier and more engaging.

If the user is to meet one of the ghosts, then they automatically lose a life, and they must start from the starting point again however all progress they have made in that level will be retained. Once they die the level, they reached the amount of “Pac Dots” they collected, and their overall time will all be saved to the game database.

There are various methods of AI to use for the ghosts in Pacman. Firstly, we need to know that each ghost moves differently. The red ghost (Blinky) will try to approach Pacman and will follow his exact location once found. The pink ghost (Pinky) will attempt to get ahead of Pacman and cut him off. The cyan ghost (Inky) will often just roam around a certain area and his movement is very unpredictable. And finally, the orange ghost (Clyde) just moves around randomly and usually steers clear of Pacman. I will need to develop a different AI for each ghost in the game.

The multiplayer aspect of my game will be not be guaranteed as it is dependent on time restrictions and various game complications.

**Background Research**

Pacman is originally a maze arcade game created back in 1980 in Japan by a Namco. Pacman is a maze arcade game where the player must control Pacman and navigate it through the maze attempting to collect all the “Pac-Dots” whilst avoiding the various coloured ghosts. If the player eats the large dots, then the ghosts change colour, and this allows Pacman to be able to eat them for extra points. Development for the game originally began in 1979 by Mr. Toru Iwatani.

Iwatani designed the game to engage both male and female users to games as at the time most games were tailored towards men with genres such as war and sports. He also designed Pacman in a way to attract younger users. The game begins with one of the ghosts (the red one) outside the ghost’s box whilst the other ghosts remain inside. The red ghosts will start to approach you almost immediately and will continue following you around the maze.

Then almost as soon as the game starts the second ghost (the pink one) will exit the ghost chamber and will attempt to go to where Pacman is heading rather than his actual positions right now. The blue ghosts will continue to remain in the ghost’s chamber for a longer time and he won’t exit until Pacman has managed to eat 30 of the “Pac-Dots”, his movement is dependent on the red ghost and Pacman as he predicts a target tile. Then finally the orange ghost leaves last and he does not do so until at least a third of the “Pac-Dots” have been consumed. This ghost has two different modes that it alternates during the game. One is if his distance is further than 8 tiles away then he will use the same algorithm as the red ghost. The other mode is when he is within 8 tiles of Pacman, he will try to reach the tile set for him in his scatter mode.

Once Pacman has eaten one of the larger dots then each ghost enters scatter mode this is when each ghost will head towards a specific corner outside the maze. Therefore, as this point cannot be accessed the ghosts will continue looping around attempting to get to that point until the effects of the larger dot rush out typically between 5 and 7 seconds before returning to chase again.

AI has been in the game of Pacman from the very start and is a vital part of the game for the ghosts to be challenging. We know that for the ghosts to work as AI we need both a data structure and a searching algorithm to process and find the shortest path towards the Pacman.

Upon researching previous versions of the Pacman game, I expectedly came across the original arcade game. This game is very well built and has a majority of the features that I would like to also implement into my Pacman game, however, some features aren’t in the original game that I would hope to add to my own. An example of a feature from the game is the AI algorithm for the ghosts. I would like to take the algorithm for the ghosts however I would like to change them slightly in order the make the game more challenging and engaging.

Another feature the game implements is the use of a leader board or scoreboard to hold high scores. I am going to add this feature to the menu section of my game. One feature that is not implemented in any version of Pacman that I researched was the ability to play multiplayer. This feature would enable players to have a cooperative mode and play over a network with their friends. Therefore, to increase my target audience, I will make the game multiplayer.

From this source we can see that Pacman is a graphical-based game therefore in my game I aim to replicate this feature.

Aside from the ghosts in the original game, I would like to alter some of the algorithms for the ghosts. First of all, the red ghost Blinky which uses maze solving to find the shortest path to the user and the orange ghost which approaches the cell that’s Infront of the Pacman will both remain the same however is the other two ghost algorithms that I would like to alter.

One new ghost I would like to add would be that instead of the Pink ghost I would have a Black ghost called Moe. I would like to implement a ghost that has long periods where he doesn’t follow Pacman at all and moves randomly throughout the map until periods of high intensity where the ghost will chase Pacman until a certain amount of time has passed before he reverts to randomly moving. This ghost hopes that Pacman will have periods where he is sandwiched between ghosts making the game altogether more difficult.

The other ghost I would like to change will be a Blue ghost named Sid. Sid works differently from any other ghost that has been in the game. He will find the Pac-Fruit nearest Pacman and he will go there and then begin moving in a random direction near the fruit. The aim of this ghost is to make it more challenging for users who are just used to grabbing the Pac-Fruits to pass all the levels the addition of this ghost makes the game more challenging. Then if there aren’t any Pac-Fruits left in the game the job of this ghost is “Completed” and thus he will just wander around the map until the game is finished and he will continue next level.

The final feature I would like to add that will differ from the original Pacman game would be the time the ghosts come out. To tailor for younger players and make the game more enjoyable and easier I will release the ghosts at different points during the game. The idea I have in mind is to only release two ghosts in the first round to allow users to play for longer. Once the user reaches the third level the Black ghost will be released and every level after that all the ghosts will be released after set requirements have been met.

Other features I will implement:

* Music
* Menu
* Low level graphics

**End User**

Pacman’s age rating is 3 years old and E for everyone which is because the game itself can be reduced to simply making moves left right up or down depending on your situation. Therefore, I will need to take this into account and make my GUI very simple and easy for even a 3-year-old to navigate around and play a game, therefore in my game I will have minimal buttons and each one of them will be large.

My game will be aimed at users both young and old male and female who simply what a game that is exciting and challenging but also addictive drawing them in to play and increase their high score. I will make my game easy to use and understand as well as having a help section to assist users who are not necessarily game addicts, I can do this by implementing GUI. I would hope that my game forces the user to improve their ability to plan as they try to predict the direction the ghosts are going to move to survive longer.

I would hope for my game to be fast-paced and engaging. It will be directed towards both people who do not have a lot of time and people with a lot of time as each round is fast paced. Recently arcade games have fallen out of favour with many people as console and pc games have slowly overtaken them in the last decade. Therefore, I would like my version of Pacman to be as similar to the actual game as possible whilst also improving aspects from the original version to make it more enjoyable.

In my game, I also hope to draw old-school fans from the original game that want to replay the game, but they would like a little bit of a different type of challenge. I hope to add little features to my game to draw these users to my game.

**Interview**

In order to visualise the need of potential users I decided to ask a 10-year-old and a 19-year-old questions on features I could implement to my game to make it easier for both to play

1. What features could I include to make the game more interactive?

10-year-old: I would like to see not a lot of words and bright colours with simple shapes.

19-year-old: shapes and short sentences would help but also If there was possibly a navigation help system in place that shows the user where to go if it is their firs time playing the game.

1. Have you ever played Pacman if so, would you recommend any additional features?

10-year-old: Yeh I have but the ghosts in the game are a bit difficult so I would like them to be easier and I would like to play the game with my friends from my school when we’re all at home.

19-year-old: Yeh I have played the game a lot and in my opinion the map begins to get a little repetitive after a couple of levels so if the map could be changing that would make the game better.

1. Do you believe the game is fast paced?

10-year-old: Yeh it is too hard for me to change direction and I always get stuck on level one.

19-year-old: Yeh I would have to agree that it is too fast for me. Often, I am stuck on level 2 because of how fast the ghosts move.

1. Would you like an extra network feature in order to play with your friends?

10-year-old: This is something I said earlier because I would really like to be able to play with my friends together so that we can beat the game.

19-year-old: My friends do not really play games all that often so hopefully if I tell them that Pacman has a multiplayer feature it might convince them to play the game.

1. Do like the number of ghosts in the game?

10-year-old: I do not like the ghosts they are too hard, and they always follow me around.

19-year-old: The ghosts are quite complex with the way they constantly follow you around all the time however I would like to see different ghosts added to the game in order to make it more fun.

**Interview Analysis**

From my interview I can see some of the essential features I will have to include into my game for my game to be used to both young and old users. The predictions I made were correct as it seems users would like to see:

* Multiplayer
* Help Option
* Larger Buttons
* Bright Colours
* Simple Shapes
* Easier game

I can also see that potential end users would like to game to remain the same as much as possible it seems that they want minor changes such as “randomising the size of map” in order to make the game less repetitive and “making the ghosts easier” so they can get further in the game, therefore, I will take both these statements into consideration when creating my game and hopefully keep the game as enjoyable as possible.

In order to make the game easier for the users, one idea I had was to limit the ghosts per level. This would involve things such as in the first level only releasing out the red and orange ghosts and then as the levels progress each ghost is released. The aim of this is to ensure users still have a chance to play the game nearer the start but then only skilled and dedicated players can reach the later levels.

I can also see that a multiplayer networking feature is something that is highly wanted in both older and younger users so that they can play with their friends from school whilst they are at home. I will ensure that I have this in my game now in order to capture players at the higher end of the age group as well as users from the lower end.

Finally, I can see that older users that have played the game before would like different ghosts to play with as they have already had enough of the old ghosts.

**Objectives:**

* Login system that only accepts the correct username and password combination from the user.
* Sign Up/ Register Option for users who do not have an account and would like to play the game.
* Music in the background of the game as well as special sounds for events such as picking up “Pac-Dots”.
* Graphics that will be loaded into the game and used for the Pacman, the four different ghosts and the “Pac Dots”.
* Leader board which will be accessible from the menu page and will show the top ten high scores as well as their gamertag.
* Multiplayer Network Option which allows friends to simultaneously play with their friends on the same map. Which involves:
  + Create a network sever.
  + Sending data over the network.
  + Receiving data over the network.
  + Updating player location depending on the data received.
* Simultaneous isolated multiplayer games
* Maze Solving this will be used in the AI for the ghosts and will enable to try and eat Pacman throughout the game. Which will use:
  + Pythagoras to find the shortest distance.
  + A\* shortest path algorithm.
* Various AI ghosts each using the same algorithm but having different destination cells. (e. g the red ghost will aim for the same cell as Pacman whilst another ghost may aim for the cell ahead of Pacman.
* Blue ghost that hovers around the large Pac dots
* Black ghost that has periods of chasing you followed by longer periods where he randomly moves around.
* Orange ghost which attempts to predict where you’re going by aiming for two spaces ahead of you.
* Point System which will be used to determine each user ranking in the overall scoreboard, it will be dependent on various factors.
* Game Pause which will stop the game at any point.
* Easy to use GUI with interactive buttons that change colour when you hover over them to allow both users of all ages to easily and efficiently navigate and understand the game.
* User can control Pacman and they can change his orientation depending on four keys. The keys will be pre-set to the up, down, left and right arrows. There may be an option for the player to change the keys to custom options.
* Display points and lives at the top of the screen.
* A ghost’s box that they roam in until they can exit.
* Pacman should be able to consume the ghosts once he has eaten a special fruit.
* Users score saved to the database.
* Graphics that will change for the ghosts once the “Pac Fruit” has been eaten this will last for around 10 seconds and once the ghosts are in this mode if they are eaten, they will die and will have to return back to their ghosts’ cube to come back to life and flickers when there’s 3 seconds left of being frightened.
* If the ghost Is eaten after their graphics will change to just eyes.
* Then once the player gains a certain number of points or a set amount of time passes the blue ghost will exit the chamber and attempt to chase Pacman.
* The black ghost will continue to remain in the ghost’s chamber for a longer time and he won’t exit until later levels.
* Then finally the orange ghost leaves last and he does not do so until at least level 4.
* Attempt to make the game progressively harder.
* Randomly generated Pacman maze, this will be dependent on if I have enough time.
* Multiple Levels to make game more difficult.

Source:

<https://gameinternals.com/understanding-pac-man-ghost-behavior>

**Diagram, schematic

Description automatically generated**

**Data Requirements**

**Login In / Register:**

Initially when a new user joins the game, they will be unable to play as they don’t have an account registered therefore, they will have to sign up for the game, the data they create here is then saved in the Game Database and is used to check a user’s username and password combination if it is correct or not. Then once the user has created an account or has logged into the game, they will be taken to the main menu page.

**Main Menu**

Here the user is presented with various choices. Each choice will be large and a simple shape in order to make it easier for younger players, the options will be:

Single player Mode:

Here the user has joined a single player mode. The game will generate a randomly sized map and the game will begin and will continue until the user has run out of lives and died. Once the game has finished the user’s data will be saved to the database and they will be notified if they set a personal or public high score and then they will be taken back to the menu section to play another game or close the application.

Multi-player Mode:

Here the user is prompted with a friends list and they can see all the people they have friended. If the user has friends that are online, then they can send them a request to be able to play the game online together and work together in order to defeat the ghosts. Once both users die then their data is saved to a game database for multiplayer games. Then they prompted if they set a personal or a public high score and they are taken to the main hub able to play again.

About Page:

This page will have lots of information about how the game works. Once the player clicks the button, they will be taken to a screen filled with various images and text about the ghost behaviours and how to move and pause the game etc.

Scoreboard:

If the user clicks this option, they are shown both the single- player and the multiplayer high scores for this game. The data for this is taken from the game database and then they are redirected to the main hub again.

**Main Complexity**

One main complexity I have in my game is my maze solving or shortest path algorithm. After doing research I have decided that the A\* Shortest path algorithm would be best for my game. The algorithm is much more efficient than Breadth-first search as It only searches the position that is closest to the end position. Here an A picture containing watch

Description automatically generatedexample of how the algorithm works.

Say for instance we had this graph. The algorithm is dependent on an accurate and reliable heuristic value which will be an approximate distance from the end position. This value should always be an underestimate. The value of the heuristic is added to the actual cost of the distance between two points. E.g if we were trying to reach point D and the heuristic value at B was 3 its overall score would be 5 and if the heuristic at C was 4 its overall score would then be 9. These values are then added to a priority queue along with the position that it came from in this instance it would be A then ordered by size then the lowest value is taken out and its adjacent nodes are then checked. This repeats over and over again until the end position is found. Once the end position is found the algorithm backtracks going to each position it came from before then arriving at the starting position, this is then the shortest path.

To calculate the heuristic value in my game I have decided to use the Manhattan distance calculator. The way this works is very simple and demonstrated by this diagram.

Chart, line chart

Description automatically generated

The distance calculate is essentially the length of the y distance between points + the length of the x difference between points, or (x2 – x1) + (y2-y1).

My aim is to use maze solving techniques in order to find the shortest path between the ghost and Pacman and travel down that path.

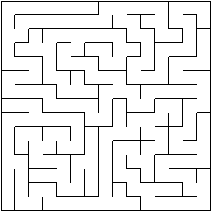
The black ghost Moe is a custom ghost. The algorithms used for this ghost will be the same as the algorithms used for the other ghosts however the intended square will be 2 places behind Pacman. The code will need to check if the intended square behind Pacman is either in the game or not and will act accordingly.

The final ghost will be called Sid and will be a ghost that hovers around the Pac-Fruits that give Pacman a special ability. I intend to do this by first calculating the shortest distance between Pacman and each of the Pac-Fruits then finding the closest fruit to Pacman. Then the ghost will move to that fruit and hover around that area for a set amount of time until the algorithm just repeats over and over. If there are not any Pac-Fruits around the ghost will just wander around the map until the next level.

Another complexity involving mazes will be the period where the ghosts are vulnerable. The ghosts are required to run towards a corner in the map and they will do so by taking the shortest path from their exact location to the corner whilst the effect of the Pac-Fruit is still in effect until they go back to their separate algorithms.

**Random Maps**

If I have enough time, I may implement a button which allows the user to play the game but with random maps.



To begin with, one of the difficulties for my game will be the maze generation I have to create. I researched various maze generation algorithms. The first I came across was a recursive backtracker, you start by choosing a random point in the field then randomly choose an adjacent cell if it hasn’t been visited yet you go to that cell and that becomes the new current cell. If all the cells adjacent have been visited, you go back and repeat the whole process until it has backed up to the starting point. It should generate a maze-like this.

However, the problem with using this sort of maze generation is that the maze generated in Pacman a wider and symmetrical with various ways to arrive at a single point. Most of the maze generation techniques available generate a single channel maze that is unsymmetrical. Therefore, I will need to create my maze generation algorithm that also ensures there will be a box for the ghosts in the centre.

MAP = [  
 [1, 1, 1, 1, 1, 1, 1, 1],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 0, 0, 0, 0, 0, 0, 0],  
 [1, 1, 1, 1, 1, 1, 1, 1]  
 ]

However, the conclusion I arrived at was very different from what I was aiming out to do. I deduced that the maze for Pacman is better described as a map as a maze typically has multiple dead ends and singular paths however this isn’t the same for Pacman as the entire map is connected without and ends, this ultimately changed the way my map generation would work. For this complexity I have decided to have a set 2D list in python that represents the map then when map generation begins the program will randomly select Tetris styled shapes to generate randomly. In the example to the left the 1 is used to represent the walls and the 0s represent the empty squares across the map.

**Sources:**

<https://www.google.com/search?q=weighted+undirected+graph&rlz=1C1CHBF_en-GBGB874GB875&sxsrf=ALeKk03mi3tM4DkO0DyNNlHJhhx-LMf2eQ:1588604080990&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjaoL6vu5rpAhWIN8AKHY-jBIIQ_AUoAXoECA8QAw&biw=681&bih=615#imgrc=TJ88n82xeyzHnM>

<http://weblog.jamisbuck.org/2010/12/27/maze-generation-recursive-backtracking>

<https://shaunlebron.github.io/pacman-mazegen/>

<https://www.google.com/search?q=pacman&rlz=1C1CHBF_en-GBGB874GB875&sxsrf=ALeKk01FiqWM3WCN3T2bC0oU1TYNT04lzg:1588605086045&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjr-d2Ov5rpAhXBoFwKHVVlA2kQ_AUoAXoECCIQAw>

# Design

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

In my network section, I have used JSON 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 which means they will each have different AI goals. In my game, I will also have a multiplayer feature where players can 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** |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **RETURNS** |
| 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 |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **RETURNS** |
| 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** |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **RETURNS** |
| 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 |
| **FUNCTION** | **PARAMETERS** | **PURPOSE** | **RETURNS** |
| 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 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 as well 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** |

**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 ghost’s variables depending on the data received from the server. | **NONE** |
| reset | player\_id | Resets 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, game\_over\_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, updated 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 returns 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. | **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 |

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, Pacman's starting position, fonts, etc.

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

In my network file data is sent to and from the players in a multiplayer game. Once the server starts running each player is allocated either a zero or a one depending on when they joined, each pair of people are also allocated a game\_id in order to uniquely identify each game. When player zero joins a copy of the initial positions list and variables is made and added to the game\_data dictionary. Once the first player joins the game, they are sent the variable waiting which will update to true when player two joins the game and when player two joins the game starts. When the game starts the players send over their own location and receive the location of the other player. The player allocated 0 also controls the ghost’s position and direction this data is then sent to the network and received by the other player.

**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 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 represent 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 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 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 can do so at any time.

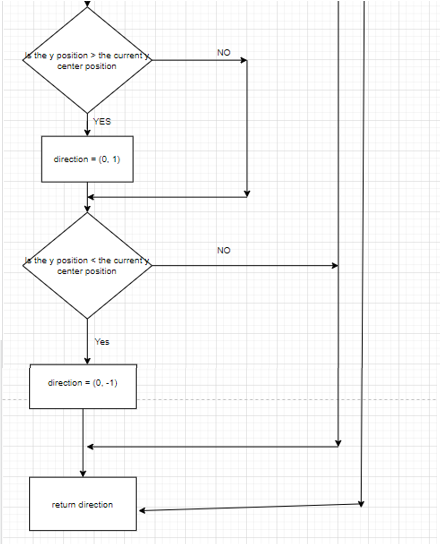
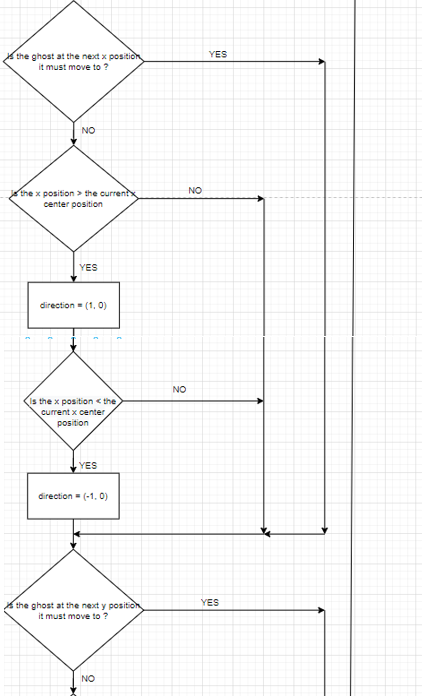
**A\* Shortest Path Algorithm**

**Diagram

Description automatically generated**

**Shortest Path algorithm:**

Diagram

Description automatically generatedMy project uses the A\* shortest path to find the shortest path to the player from the ghost and then move to them. The variable cell to check holds 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 the largest distance from the end position to the smallest distance from the end position. If the closest position is already in the priority queue the remove it from the 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 to calculate the direction that they need to move 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 to move in random directions in the corner, it’s also used by the blue ghost 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 backward from the end position and find all of the nodes/ cells that it has visited and adding them to a separate list. This list is then used by the direction calculator to move in the correct direction. It starts first at the end position and then checks 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:**

This subprocess aims to find all the adjacent cells. 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 to calculate estimate the shortest path. This distance is then used later 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 to control what occurs when during the entirety of using the game from login to close. It switches between various even managers and draws screen functions 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 to find the top 10 players (if possible) and order them. The player class also holds 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 a 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 ensures 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 to reset all variables and positions. Finally, the class takes in the player class as a parameter 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 for 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 initialization, a 2D maze is created consisting of 0s and ones 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 to have a chance of random fruit being placed randomly in the maze per level.

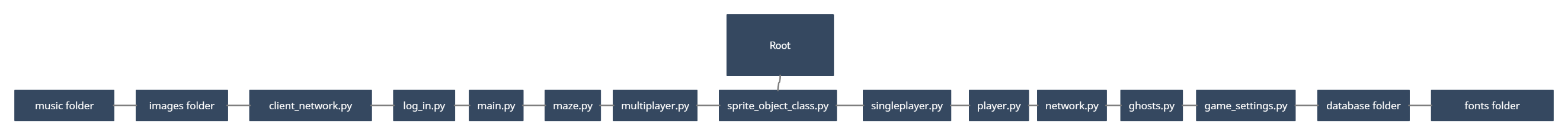
**Client:**

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 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 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.

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

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

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

# Testing

**Strategy**

The purpose of the tests of my program is so that I can be confident that all areas of my project function properly, areas that don’t function correctly will be fixed so they have the expected outcome. I will test my project by creating a large test table which will help in organizing the tests and ensure all are completed. Finally, to prove that I have indeed tested my code I have created a video with timestamps demonstrating the point at which each test is completed alongside my test table.

To test my login system, I will first attempt to enter valid and invalid data which are usernames that haven’t been used before and passwords that are incorrect. To test the registration section of the login system I will attempt both invalid and valid data as before. This could include usernames that have already been taken or usernames and passwords that are either too long or short. I can then test my database by making sure the user's account was created to show my database exists and is working. Finally, I will attempt to log in with the account I just created to show the system works.

I will demonstrate the GUI by clicking each of the bright buttons and hovering over them to show how they change colour to suggest you can interact with them.

To test the leader board page, I will be comparing the top ten users displayed on the page against the top 10 users I get from a SQL database query. To also test the leader board works I will be inputting data directly to the database and once I review testing the leader board to see if the leader board has been updated or not.

I will use my singleplayer and multiplayer sections to test a range of objectives. To test the level increase works I will be fully completing a set number of levels of the game. I will work my way up to level five to demonstrate how the difficulty of the game gradually progresses as more ghosts are released at later levels. I will be testing my in-game music by selecting the singleplayer mode and waiting for any music to play. My in-game graphics will be demonstrated through my traversal of either the multiplayer or singleplayer sections. I will test the game being paused in both the singleplayer and multiplayer sections to demonstrate it working and what happens when it is paused. I will also test the keys being able to control the player. I will test my game score through the singleplayer game mode. To test this section, I will move the player around collecting the different types of Pac dots, large Pac dots, and fruits and monitoring the score to see if it will increase or not. I will also test if the lives score and levels variables are displayed on the screen and if they increase accordingly.

To test if the database saves the user's data, I will be playing a game and getting a high score then checking the leader board and database to see if the data has been updated.

To test each ghost, I will show each ghost moving individually once by one to demonstrate the difference in their algorithms. I will be attempting to show the red ghost chasing the players' direct location, the blue ghost hovering around the large Pac dots if any remain, the black ghost who has intervals between when he chases the player and roams randomly and the orange ghost who predicts where the player is going and if possible, attempts to get two spaces ahead of them. Each ghost will also demonstrate the shortest path algorithm they are using to find the path they take. I will also test to ensure the ghosts remain in the ghost cage until the requirements for them to leave have been met. I will also test player collisions with ghosts when the ghosts aren’t frightened to see if the player dies. I will also test player collision with the ghost when the ghost is frightened which will test if the ghost image changes to eyes once he has been eaten and runs away back to the ghost cage.

Finally, I will test the multiplayer section of the game by playing across two separate computers and recording them to show the games are connected and also demonstrate how multiple isolated games can be run simultaneously.

I will perform a test to prove that my entire program works by following a set path, the path goes as followed:

I will first attempt to sign in with invalid details, I will then register a new account to show this section is working and show the database to show the new username and password was registered.

I will then proceed to sign into this new account and check the leader board to see all the high scores once again opening the database and running a SQL command to verify these are the top accounts. I will then play a singleplayer game to demonstrate the majority of the objectives.

Once I have finished with the singleplayer mode and set a high score I will check the leader board again to demonstrate how it updates.

I will then move onto the multiplayer section of the code opening two different games and showing how they are linked then opening another game and showing how that is independent of any other game.

Then I will have a separate video to further demonstrate the individual ghost moving mechanisms.

LOGIN SYSTEM TEST: <https://youtu.be/8M7WCXNeP6M>

SINGLEPLAYER GAME TEST: <https://youtu.be/bObs6P0TXiA>

SINGLEPLAYER PAUSE TEST: <https://youtu.be/FXtyvoLrufA>

MULTIPLAYER GAME TEST: <https://youtu.be/cQrqajnl6aA>

MULTIPLAYER GHOST DEATH: https://youtu.be/yhel9meSkwU

GHOST SHORTEST PATH TEST: <https://youtu.be/pntvLEjjYic>

CLOSEST PAC DOT TEST: <https://youtu.be/hH2E8_Sg6i8>

BLACK GHOST MOVEMENT TEST: https://youtu.be/xeIdaAL5g1o

ORANGE GHOST MOVEMENT TEST: https://youtu.be/0wCTO8-tfhM

Singleplayer Tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **1.01** | Empty username or password box returns an error | Username: “”  Password = “” | Error prompt indicating you must enter data | Error window appears | **NONE** | **0:03**  LOGIN SYSTEM TEST |
| **1.02** | Invalid usernames and prevent user log in | Username: notausername  Password:  Password | Error prompt indicating the username doesn’t exist in the system | Error window appears | **NONE** | **0:15**  LOGIN SYSTEM TEST |
| **1.03** | Invalid password prevents user log in | Username: admin  Password: notthecorrectpassword | Error window indicating the password is incorrect | Error window appears | **NONE** | **0:24**  LOGIN SYSTEM TEST |
| **1.04** | Register new account with pre-existing username | Username: admin  Password: newuser | Error prompt indicating the account already exists | Error window appears | **NONE** | **0:39**  LOGIN SYSTEM TEST |
| **1.05** | Register new account with invalid username conditions | Username: hi  Password: pacman | Error prompt indicating username too short | Error window appears | **NONE** | **0:50**  LOGIN SYSTEM TEST |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **1.06** | Register new account with invalid password conditions | Username: newuser  Password: pas | Error prompt indicating the password is too short | Error window appears | **NONE** | **0:59**  LOGIN SYSTEM TEST |
| **1.07** | Register a user with valid account details | Username: newuser  Password: password | Successful account creating prompt | Info window appears | **NONE** | **1:06**  LOGIN SYSTEM TEST |
| **1.08** | See if the registered data is saved to the database | SELECT \* From user WHERE username = ? | Account saved on database | Account on database | **NONE** | **1:16**  LOGIN SYSTEM TEST |
| **1.09** | Login with valid username and password details | Username: newuser  Password: pacman | Login window closed and the user is taken to the menu screen | User logged in successfully | **NONE** | **1:29**  LOGIN SYSTEM TEST |
| **1.10** | Button changes colour when hovered over | Mouse over button | Button becomes darker | Button becomes darker | **NONE** | **1:34**  LOGIN SYSTEM TEST |
| **1.11** | Button achieves the goal intended that’s written on the button | Leader board button clicked | Taken to leader board screen | Leader board screen appears. | **NONE** | **1:41**  LOGIN SYSTEM TEST |
| **1.12** | Leader board displays correct top 10 players from the database | “Select username, highscore FROM user ORDER By highscore DESC” | Correct 10 users displayed | Correct users displayed | **NONE** | **2:25**  LOGIN SYSTEM TEST |
| **1.13** | Does the menu button work | Menu button clicked | User taken back to the menu screen | User taken to menu screen | **NONE** | 0:02  SINGLEPLAYER GAME TEST |
| **2.00** | Does the singleplayer button work | Singleplayer button clicked | User taken to the singleplayer game | Singleplayer game starts | **NONE** | 0:09  SINGLEPLAYER GAME TEST |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **2.01** | Test that the game doesn’t start till the onscreen counter is finished. | Trying to move. | Unable to move and ghosts don’t move | All sprites remain in position | **NONE** | 0:16  SINGLEPLAYER GAME TEST |
| **2.02** | Collecting pac dots updates score and removes the from the screen | Moving into pac dots | Score increases and they removed from the screen | Score increased and they’re removed from screen | **NONE** | 0:18  SINGLEPLAYER GAME TEST |
| **2.03** | Game music and pac dots | Playing game | Music Plays | Music Plays | **NONE** | 0:25  SINGLEPLAYER GAME TEST |
| **2.04** | Blue ghost released once the player gains 200 points | Collecting 200 points | Blue ghost leaves ghost cage | Blue ghost leaves ghost cage | **NONE** | 0:30  SINGLEPLAYER GAME TEST |
| **2.05** | Collisions with ghost when they aren’t frightened results in player losing a life and the images that represents lives decrease by one | Player collides with ghost when the ghost isn’t frightened | Player restarts level and loses a life | Level restarts and the player loses a life | **NONE** | 0:26  SINGLEPLAYER GAME TEST |
| **2.06** | Collecting a large pac dots changes the ghosts state and makes the frightened | Moving into large pac dots | Ghost becomes frightened and runs and runs away | Ghost becomes frightened and runs to its corner | **NONE** | 0:41  SINGLEPLAYER GAME TEST |
| **2.07** | Frightened ghosts have a unique image | Eating large pac dot | Ghost turns a dark blue | Ghost turns dark blue | **NONE** | 0:41  SINGLEPLAYER GAME TEST |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **2.08** | Frightened ghosts will run to their unique corner with the shortest path | Eating large pac dot | Ghosts runs away | Ghost runs | **NONE** | 0:42  SINGLEPLAYER GAME TEST |
| **2.09** | Ghosts flicker in last 3 seconds of being frightened | Eating large pac dot | Ghosts flicker | Ghost flicker | **NONE** | 0:52  SINGLEPLAYER GAME TEST |
| **2.10** | Collision with ghost that is frightened results is the ghost’s death and the ghost runs to the ghost cage | Eating ghost when the ghost is frightened | Ghost dies | Ghost dies | **NONE** | **1:02**  SINGLEPLAYER GAME TEST |
| **2.11** | Dead ghosts will have an image of eyes and run to ghost cage using the shortest path to get there | Eating ghost when it is frightened | Ghost runs away as an eye image | Ghost runs away as an eye image | **NONE** | **1:02**  SINGLEPLAYER GAME TEST |
| **2.12** | Player is able to pause game | Clicking esc | Game paused | Game paused | **None** | **0: 10**  SINGLEPLAYER PAUSE TEST |
| **2.13** | Player is able to resume game | Clicking resume game buttons | Resume game | Game resumed | **None** | **0:13**  SINGLEPLAYER PAUSE TEST |
| **2.14** | Completing the level reset the maze and all entities | Eating all pac dots and large pac dots | Game reset | Game reset | **NONE** | **1:38**  SINGLEPLAYER GAME TEST |
| **2.15** | Black ghost comes out at level 3 | Reach level 3 | Black ghost begins to move randomly | Black ghost moves randomly | **NONE** | **2:51**  SINGLEPLAYER GAME TEST |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **2.16** | When the black ghost is chasing its image changes to a white ghost | Move around maze | Image changes | Image changes | **NONE** | **3:00**  SINGLEPLAYER GAME TEST |
| **2.17** | Orange ghost comes out at level 4 | Reach level 4 | Orange ghost begins to chase two spaces ahead of pacman | Orange ghost chases | **NONE** | **4:01**  SINGLEPLAYER GAME TEST |
| **2.18** | When the player loses all their lives the game freezes and the player has the option to the menu or to start again. | Die repeatedly to the ghosts | Game over heading appears alongside buttons. | Game over message appears alongside buttons. | **NONE** | **4:45**  SINGLEPLAYER GAME TEST |
| **2.19** | Players Score updates on leader board once they die | “Select username, highscore FROM user ORDER By highscore DESC” | Score updates on leaderboard | Score updates on loaderboard | **NONE** | **5:07**  SINGLEPLAYER GAME TEST |

Multiplayer Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **3.00** | Multiplayer button takes you to the multiplayer section | Clicking multiplayer button | Multiplayer game opens | Multiplayer game opened | **NONE** | **0:02**  MULTIPLAYER GAME TEST |
| **3.01** | When player connects to multiplayer there is a waiting screen | Join multiplayer game | “Waiting For Player 2…” message on screen | Waiting message on screen | **NONE** | **0:05**  MULTIPLAYER GAME TEST |
| **3.02** | When a second player joins the game it automatically starts | Second player join game | Game countdown occurs | Game countdown starts | **NONE** | **0:17**  MULTIPLAYER GAME TEST |
| **3.03** | When one player on his screen he also moves on player twos screen | Moving around | Players position is updated on both screens | Players position updated on both screens | **NONE** | **0:23**  MULTIPLAYER GAME TEST |
| **3.04** | When a player increases their score the score updates on both screen | Eats pac dots | Score updates on both screens | Score updates on both screens | **NONE** | **0:25**  MULTIPLAYER GAME TEST |
| **3.05** | When one of the players dies their total life decreases for both players. | Die | Image that represents lives decreases by one on both players screens | Image that represents lives decreases by one on both screens | **NONE** | **0:43**  MULTIPLAYER GAME TEST |
| **3.06** | All ghosts’ positions are updated on both screens and the positions and directions are identical | Move around and have ghost chase me | Ghost position is identical in both games | Ghost position is the same | **NONE** | **0:26**  MULTIPLAYER GAME TEST |
| **3.07** | When either player consumes a large pac dot the ghost becomes frightened and their image changes on both screens | Eat large pac dot | Ghost becomes frightened on both screens | Ghost is frightened on both screens | **NONE** | **0:49**  MULTIPLAYER GAME TEST |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **3.08** | Ghosts flicker on both screens when they have 3 seconds of being frightened left | Eat large pac dot | Image flickers on both screens | Flickers on both screens | **NONE** | **1:03**  MULTIPLAYER GAME TEST |
| **3.09** | When either player collides with a ghost when its scared the ghost dies on both screens | Eat ghost when frightened | Ghost image changes on both screens | Ghost image changes on both screens | **NONE** | **0:12**  MULTIPLAYER GOHST DEATH |
| **3.1** | Frightened ghosts will run to their unique corner with the shortest path on both screens | Eating large pac dot | Ghosts runs away on both screens | Ghost runs on both screens | **NONE** | **0:49**  MULTIPLAYER GAME TEST |
| **3.11** | When one player pauses the game, it is paused for both players | Click esc to pause game | Both players have a paused screen with different messages | Both players have a paused screen with different messages. | **NONE** | **1:06**  MULTIPLAYER GAME TEST |
| **3.12** | Games are made independently of each other and other players can play a unique game simultaneously to your game | Open another game with two more players | Games are unique | Games are unique | **NONE** | **1:54**  MULTIPLAYER GAME TEST |
| **3.13** | Dead ghost run to the ghost cage using the shortest path | Eating ghost when frightened | Ghost runs to ghost cage | Ghost runs to ghost cage | **NONE** | **0:12**  MULTIPLAYER GOHST DEATH |

Ghost Shortest Path Test:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test Purpose** | **Test Data** | **Expected Outcome** | **Actual Outcome** | **Changes** | **Timestamp of test and appropriate video** |
| **4.00** | Ghost is able to find shortest path between two points | Two random points in the maze | Ghost traverse shortest path | Ghost traverses’ shortest path | **NONE** | **0:00 – 1:24**  GHOST SHORTEST PATH TEST |
| **4.01** | When ghost is frightened it runs to its corner using the shortest path | Red and Blue ghost as examples. | Ghost takes shortest path | Ghost takes shortest path | **NONE** | **1:26 – end**  GHOST SHORTEST PATH TEST |
| **4.02** | Blue ghost is able to find the closest pac do to pacman and travel the shortest distance there | Blue ghost and pacman | Travels shortest distance to large dot | Shortest distance travelled. | **NONE** | **ENTIRE VIDEO**  CLOSEST PAC DOT TEST |
| **4.03** | Back ghost has unique behaviour and travels shortest distance | Black ghost and pacman | Black ghost travels shortest path and changes its image when it is chasing the player | Black ghost travels shortest path and changes its image when it is chasing the player | **NONE** | **ENTIRE VIDEO**  BLACK GHOST MOVEMENT TEST |
| **4.04** | Orange ghost has unique behaviour that chases two spaces ahead of player using the shortest path algorithm. | Orange ghost and player | Travels the shortest path to cut off the player | Shortest path travelled. | **NONE** | **ENTIRE VIDEO**  ORANGE GHOST MOVEMENT TEST |

In my test I have decided to only show the frightened shortest path for two ghosts, the red and blue this is because all the ghosts use the same shortest path algorithm in order to get to the predetermined squares so essentially the outcome of one ghost is identical to the other ghosts the end result would just be arriving at a different square. Finally, this is the same for when the ghost has died and Is no longer alive as the end position is the only difference. In my videos and testing I opted to show the shortest path algorithm works as a whole at different positions in order to show it will work regardless of the end and start location.

# Evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| Objective | Met ? | Comment | Test Reference |
| Login system that only accepts the correct username and password combination from the user. | Yes | The login system will allow a user to log into the system as long as their username and password are correct and have previously been registered. If the user enters a username that hasn’t been registered into the system, then an error message informing them that the username doesn’t exist pops up on the other hand if their password is incorrect then they’re brought an error screen saying incorrect password.  Objective met as the user is able to sign into the game as long as their details are correct. | **TEST No: 1.01**  **TEST No: 1.02**  **TEST No: 1.03**  **TEST No: 1.09** |
| Sign Up/ Register Option for users who do not have an account and would like to play the game. | Yes | If users attempt to register an account with a username that’s already been registered, then they are told so in a error message box the same happens if their username is too short or long or if their password is too short. Otherwise, their account will be registered.  This objective has been met as users are able to create a new account. | **TEST No: 1.04**  **TEST No: 1.05**  **TEST No: 1.06**  **TEST No: 1.07** |
| Music in the background of the game as well as special sounds for events such as picking up “Pac-Dots”. | Yes | Once the level starts the music starts playing and every time he dies, restarts or un pauses the level the music will restart.  Objective met as players have in game music in the singleplayer mode. | **TEST No: 2.03** |
| Graphics that will be loaded into the game and used for the Pacman, the four different ghosts and the “Pac Dots”. | Yes | N/A | **Figure 1.1** |
| Objective | Met ? | Comment | Test Reference |
| Leader board which will be accessible from the menu page and will show the top ten high scores as well as their gamertag. | Yes | The leader board is constantly updating and will display the top 10 users if available.  Objective met as players are able to view the top high scores in the game. | **TEST No: 1.12** |
| Multiplayer Network Option which allows friends to simultaneously play with their friends on the same map. | Yes | The players are able to play across a network. Once one player joins, they have a waiting for player two screen and the game won’t begin until another player joins then when they join the game runs the same as a singleplayer game but with two players.  Objective has been met as two players are able to play an entire pacman game together across a network. | **TEST No: 3.0 – 3.12** |
| Maze Solving this will be used in the AI for the ghosts and will enable to try and eat Pacman throughout the game. | Yes | The A\* shortest path algorithm provides the shortest path between positions in the maze.  Objective met as the ghost is able to traverse the shortest path between two positions. | **TEST No: 4.0** |
| Various AI ghosts each using the same algorithm but having different destination cells. | Yes | Each ghost is different as the red ghost chases your exact location, the blue ghost the closest large pac dot, the black ghost has intervals of peace and chasing you and the orange ghost attempts to cut you off and aims for the cell two spaces ahead of you.  Objective met as the ghosts all have different behaviour patterns. | **TEST No: 4.00 – 4.04** |
| Objective | Met ? | Comment | Test Reference |
| Point System | Yes | As the player collects pac dots their score increases.    This objective has been met as there is a clear point system in the game. | **TEST No: 2.02**  **TEST No: 3.04** |
| Game Pause | Yes | If the player clicks the esc button the game will pause both in multiplayer and singleplayer.  Objective met as users can pause. | **TEST No: 2.12**  **TEST No: 2.13**  **TEST No: 3.11** |
| Easy to use GUI with interactive buttons | Yes | The buttons are large and colourful and also change colour as you hover over them.  Objective has been met as buttons are easy to use and change colour. | **TEST No: 1.10**  **TEST No: 1.11**  **TEST No: 1.13**  **TEST No: 2.00**  **TEST No: 3.00** |
| Display points and lives at the top of the screen. | Yes | Objective met as points and score are shown constantly during the game. | **TEST No: 2.02**  **TEST No: 3.04** |
| Users score saved to the database and leader board updated. | Yes | Leader board updates once the singleplayer game is over.  Objective met as the users score is saved to database. | **TEST No: 2.19** |
| Graphics that will change for the ghosts once the “Pac Fruit” has been eaten and flickers when there’s 3 seconds left of being frightened. | Yes | Ghosts become frightened and run to their corners.  Objective met as their image also updates when they are frightened in singleplayer and multiplayer games. | **TEST No: 2.07**  **TEST No: 2.09**  **TEST No: 3.07**  **TEST No: 3.08** |
| Frightened ghosts use shortest path algorithm to run away |  | Ghosts run to their corners using the shortest path algorithm.  Objective met as ghosts take the shortest path to their corners | **TEST No: 2.08**  **TEST No: 3.1**  **TEST No: 4.00**  **TEST No: 4.01**  **TEST No: 4.02** |
| If the ghost Is eaten after their graphics will change to just eyes and run to the ghost cage using shortest path algorithm. | Yes | Ghost dies if the player eats the ghost once it is frightened.  Objective met as ghost image will change to eyes in both single and multiplayer games. | **TEST No: 2.11**  **TEST No: 3.09**  **TEST No: 3.13** |
| Objective | Met ? | Comment | Test Reference |
| Then once the player gains a certain number of points or a set amount of time passes the blue ghost will exit the chamber and attempt to chase Pacman. | Yes | Once the player gains 200 points or 20 seconds passes the blue ghost is released.  Objective achieved as the ghost leaves once a specific condition has been met. | **TEST No: 2.04** |
| The black ghost leaves the chamber at level 3. | Yes | Once the player reaches level 3 the black ghost is released.  Objective achieved as the ghost leaves once a specific condition has been met. | **TEST No: 2.15** |
| Then finally the orange ghost leaves last and he does not do so until at least level 4. | Yes | Once the player reaches level 4 the black ghost is released.  Objective achieved as the ghost leaves once a specific condition has been met. | **TEST No: 2.17** |
| Randomly generated Pacman maze, this will be dependent on if I have enough time. | No | N/A | **N/A** |
| Multiple Levels to make game more difficult. | Yes | Once the player collects all the dots in a level the next level automatically begins.  Objective met as the game has multiple levels in which more ghosts exit in order to make it more difficult. | **TEST No: 2.14** |

**Users Feedback**

**How easy is the system to use ?**

**“**The system is so easy to use I love it. Right from the login screen I saw a lot or clear and straightforward text and buttons like the register and log in. When I got to the menu page, I loved how all the buttons just stood out to me like it was all so easy to read and clear. Then when I clicked each button, they took me to the correct place.”

**What do you enjoy about the game ?**

“The singleplayer game was very fun to play I loved how it got more challenging later on once more ghosts left the chamber.” - 19-year-old

“I really like how colourful all the buttons are on the screen. My favourite feature was the multiplayer section, I loved being able to play with my friends” – 10-year-old.

**Does it meet the objectives ?**

“For me 100% . Nearly everything that I would want in a pacman game is here. Looking at the objective table above I can agree that all the objectives they said they met have met.”

**Any areas to improve ?**

“Overall, I think the game is wonderful and I know I would really enjoy playing a game like this, however, random mazes are a feature that I would have loved to see. I feel like even though the game is really enjoyable I think that adding random mazes would prevent the game from even becoming repetitive another suggestion I have is more customisation of the game of example I would love to have choices on the look of pacman and maybe there was a way you could make the game easier for younger players by making the ghosts easier possibly. The final feature I would really enjoy in the game would be a friends list or a way to play with specific friends rather than random players.

**Feedback analysis:**

Overall, the feedback that I have received from two potential end users has been mainly positive this has filled me with a lot of confidence that I have achieved my objectives and produced a good game. I was happy hear that the buttons made the system easy to use for all ages as that was a feature I really wanted to get right in the system. I agree with the areas to improve these were features that were initially meant to be in the game however due to time constraints they weren’t able to be implemented.

**Possible Extensions**

If I were given the chance to develop my game again there are some features/objectives that I would certainly love to add. The first feature would be random maze generation. Initially, I set out to add this feature however I didn’t have time to complete it. Initially, I set out to have the maze represented and pygame drew rectangles however this didn’t look good and I felt as though these graphics wouldn’t have met the needs of the end-users. Next time I would create a maze then randomly place Tetris-like shapes such as L, T I, and rectangles and squares like there are in the original Pac-Man maze. Then for each wall in the maze, I would represent it with a wall image and see if that would be better visually. This would make my game more unpredictable and less repetitive which could make users play for longer periods.

Another feature I would add to my game next time would be a setting/customization file and button on the screen. Right now, my game doesn’t have any form of customization I would like to implement a way for the player to be able to change the image of the ghosts and the image of the player, I would also add a way for the player to be able to change the game music.

Also, if I had more time, I would’ve provided a way for players to play with their friends and a friends list. I initially set out to do this and if I had more time It was a feature that I certainly would’ve included this feature would’ve made my game more enjoyable for the end-users as they can play specifically with who they want to and also see if their friends are online at any time.

Finally, the last objective I would do differently is my shortest path algorithm. Currently, I am using the A\* shortest path algorithm which is very efficient at finding the shortest path. It is better than Breadth's first search in that it doesn’t search the entire maze it only searches the paths that are the closest distance to the end. However, the algorithm relies on a heuristic value that is reliable and accurate however in the Pacman maze this can be difficult to achieve as some distances are further away however they lead to the shortest path which is something that should be impossible in the A\* method. In my algorithm, I took this into account to make the ghosts take the shortest path however next time I would prefer to use a different algorithm like BFS despite its time limitations.

# Technical Solution

**main.py**

**from** **player** **import** \*

player = Player()

**while** player.game\_loop:

player.clock.tick(fps)

# MENU SCREEN OPTIONS

player.close\_game\_event()

**if** player.state == "log\_in\_screen":

player.log\_in\_screen()

**if** player.state == "menu":

player.menu\_draw\_screen()

**if** player.state == "singleplayer" **or** player.state == "gameover\_singleplayer":

player.singleplayer\_draw\_screen()

**if** player.state == "singleplayer":

player.singleplayer\_event\_manager()

**if** player.state == "multiplayer" **or** player.state == "gameover\_multiplayer":

player.multiplayer\_event\_manager()

player.multiplayer\_draw\_screen()

**if** player.state == "leaderboard":

player.leaderboard\_draw\_screen()

**if** player.state == "about":

player.about\_draw\_screen()

**if** player.state == "quit":

quit()

**player.py**

**from** **singleplayer** **import** \*

**from** **multiplayer** **import** \*

**from** **tkinter** **import** \*

**from** **log\_in** **import** \*

**import** **pygame**

**from** **pygame** **import** mixer

pygame.init()

**class** **Player**:

# INITIALIZE GAME

**def** **\_\_init\_\_**(self):

self.game\_loop = True

self.screen = None

self.singleplayer = None

self.multiplayer = None

self.login = None

self.clock = pygame.time.Clock()

self.state = "log\_in\_screen"

self.grey\_screen = None

self.menu\_background = self.background\_load(os.path.join(image\_folder, "menu\_screen.png"), screen\_width, screen\_length)

self.game\_background = self.background\_load(os.path.join(image\_folder, "maze.png"), screen\_width - width\_buffer, screen\_length - length\_buffer)

self.game\_over\_image = self.background\_load(os.path.join(pacman\_images, "game\_over.png"), **350**, **150**)

self.coin\_image = pygame.image.load(os.path.join(image\_folder, "coins.jpg"))

self.dot\_image = pygame.transform.scale(self.coin\_image, (**5**, **5**))

self.special\_dot\_image = pygame.transform.scale(self.coin\_image, (**20**, **20**))

self.fruit\_images = [pygame.image.load(os.path.join(fruit\_images, "a.png")), pygame.image.load(os.path.join(fruit\_images, "b.png")), pygame.image.load(os.path.join(fruit\_images, "c.png")), pygame.image.load(os.path.join(fruit\_images, "d.png")), pygame.image.load(os.path.join(fruit\_images, "e.png")), pygame.image.load(os.path.join(fruit\_images, "f.png")), pygame.image.load(os.path.join(fruit\_images, "g.png")), pygame.image.load(os.path.join(fruit\_images, "h.png"))]

self.wasd = self.background\_load(os.path.join(about\_images, "wasd.png"), **250**, **250**)

self.udlr = self.background\_load(os.path.join(about\_images, "udlr.jpg"), **200**, **200**)

self.red\_ghost\_image = self.background\_load(os.path.join(ghost\_images, "redghost\_right\_1.png"), **30**, **30**)

self.blue\_ghost\_image = self.background\_load(os.path.join(ghost\_images, "blueghost\_right\_1.png"), **30**, **30**)

self.black\_ghost\_image = self.background\_load(os.path.join(ghost\_images, "blackghost\_right\_1.png"), **30**, **30**)

self.white\_ghost\_image = self.background\_load(os.path.join(ghost\_images, "whiteghost\_right\_1.png"), **30**, **30**)

self.orange\_ghost\_image = self.background\_load(os.path.join(ghost\_images, "orangeghost\_right\_1.png"), **30**, **30**)

self.esc = self.background\_load(os.path.join(about\_images, "esc.png"), **100**, **100**)

# VARIABLES

# EVENT MANAGER

**def** **close\_game\_event**(self):

##### close\_game\_event #######

# Parameters : None

# Return Type : None

# Purpose :- Close game if close button pressed

##########################

**for** each\_event **in** pygame.event.get():

**if** each\_event.type == pygame.QUIT:

# IF THE PLAYER CLICKS THE CLOSE BUTTON

self.game\_loop = False

# END LOOP

**def** **end\_program**(self):

##### end\_program #######

# Parameters : None

# Return Type : None

# Purpose :- End program

##########################

self.game\_loop = False

# END LOOP

**def** **menu\_draw\_screen**(self):

##### menu\_draw\_screen #######

# Parameters : None

# Return Type : None

# Purpose :- Display the menu screen as well as buttons for the user

##########################

self.screen.blit(self.menu\_background, (**0**, **0**))

self.button(**100**, **131**, **300**, **35**, LIGHT\_RED, RED, self.screen, menu\_font, **20**, BLACK, "SINGLEPLAYER", "singleplayer")

self.button(**257**, **131**, **368**, **35**, WHITE, GRAY, self.screen, menu\_font, **20**, BLACK, "MULTIPLAYER", "multiplayer")

self.button(**40**, **131**, **460**, **35**, LIGHT\_YELLOW, YELLOW, self.screen, menu\_font, **20**, BLACK, "LEADERBOARD", "leaderboard")

self.button(**370**, **131**, **550**, **35**, LIGHT\_GREEN, GREEN, self.screen, menu\_font, **20**, BLACK, "ABOUT", "about")

# DRAW ON MENU SCREEN

pygame.display.flip()

**def** **log\_in\_screen**(self):

##### log\_in\_screen #######

# Parameters : None

# Return Type : None

# Purpose :- Control the player logging in

##########################

log\_in\_screen = Tk()

self.login = Login(log\_in\_screen)

# CREATE NEW INSTANCE OF LOG IN

log\_in\_screen.protocol("WM\_DELETE\_WINDOW", self.end\_program())

# IF THE PLAYER CLOSES THE GAME WINDOW

self.login.screen.mainloop()

# LOOP THE SCREEN

**if** self.login.logged\_in:

# IF THE PLAYER LOGGED IN

self.screen = pygame.display.set\_mode((screen\_width, screen\_length))

pygame.display.set\_caption(game\_title)

pygame.display.set\_icon(pygame.image.load(os.path.join(image\_folder, "pacman\_image.png")))

self.singleplayer = singleplayer(self)

self.multiplayer = multiplayer(self)

self.state = "menu"

self.game\_loop = True

self.grey\_screen = pygame.Surface((screen\_width, screen\_length)).convert\_alpha()

self.grey\_screen.fill((**0**, **0**, **0**, **150**))

# CREATED WINDOW SCREEN

**def** **singleplayer\_event\_manager**(self):

##### singleplayer\_event\_manager #######

# Parameters : None

# Return Type : None

# Purpose :- Runs singleplayer event manager method

##########################

self.singleplayer.event\_manager()

# DRAWING MAZE

**def** **singleplayer\_draw\_screen**(self):

##### singleplayer\_draw\_screen #######

# Parameters : None

# Return Type : None

# Purpose :- Runs singleplayer draw screen method

##########################

self.singleplayer.draw\_screen(self.screen, self.game\_background, self.game\_over\_image)

**def** **multiplayer\_event\_manager**(self):

##### multiplayer\_event\_manager #######

# Parameters : None

# Return Type : None

# Purpose :- Runs multiplayer event manager method

##########################

self.multiplayer.event\_manager()

**def** **multiplayer\_draw\_screen**(self):

##### multiplayer\_draw\_screen #######

# Parameters : None

# Return Type : None

# Purpose :- Runs multiplayer draw screen method

##########################

self.multiplayer.draw\_screen(self.screen, self.game\_background, self.game\_over\_image)

**def** **leaderboard\_draw\_screen**(self):

##### leaderboard\_draw\_screen #######

# Parameters : None

# Return Type : None

# Purpose :- Display leaderboard of the top 10 players

##########################

y\_pos\_odd = **162**

y\_pos\_even = **202**

self.text(menu\_font, **50**, WHITE, "TOP SCORES", BLACK, self.screen, screen\_width//**2**, **40**)

# HEADING

pygame.draw.rect(self.screen, WHITE, (**150**, **60**, **240**, **10**))

# UNDERLINE

pygame.draw.rect(self.screen, WHITE, (**0**, **100**, screen\_length, **40**))

pygame.draw.rect(self.screen, GRAY, (**0**, **140**, screen\_length, **40**))

pygame.draw.rect(self.screen, WHITE, (**0**, **180**, screen\_length, **40**))

pygame.draw.rect(self.screen, GRAY, (**0**, **220**, screen\_length, **40**))

pygame.draw.rect(self.screen, WHITE, (**0**, **260**, screen\_length, **40**))

pygame.draw.rect(self.screen, GRAY, (**0**, **300**, screen\_length, **40**))

pygame.draw.rect(self.screen, WHITE, (**0**, **340**, screen\_length, **40**))

pygame.draw.rect(self.screen, GRAY, (**0**, **380**, screen\_length, **40**))

pygame.draw.rect(self.screen, WHITE, (**0**, **420**, screen\_length, **40**))

pygame.draw.rect(self.screen, GRAY, (**0**, **460**, screen\_length, **40**))

pygame.draw.rect(self.screen, WHITE, (**0**, **500**, screen\_length, **40**))

pygame.draw.rect(self.screen, BLACK, (**90**, **100**, **3**, **570**))

pygame.draw.rect(self.screen, BLACK, (**360**, **100**, **3**, **570**))

self.text(menu\_font, **24**, BLACK, "#", WHITE, self.screen, **34**, **122**)

self.text(menu\_font, **24**, BLACK, "USERNAME", WHITE, self.screen, **230**, **122**)

self.text(menu\_font, **24**, BLACK, "SCORE", WHITE, self.screen, **450**, **122**)

# DRAW ON SCREEN

**for** each\_number **in** range(**1**, **10**, **2**):

self.text(menu\_font, **24**, BLACK, "{}".format(each\_number), GRAY, self.screen, **34**, y\_pos\_odd)

y\_pos\_odd += **80**

**for** each\_number **in** range(**2**, **11**, **2**):

self.text(menu\_font, **24**, BLACK, "{}".format(each\_number), WHITE, self.screen, **34**, y\_pos\_even)

y\_pos\_even += **80**

database = sqlite3.connect("database/players.db")

# CONNECT TO DATABASE

db\_cursor = database.cursor()

# CREATE CURSOR

get\_players = 'SELECT username, highscore FROM user ORDER BY highscore DESC'

db\_cursor.execute(get\_players)

# GET ALL PLAYERS IN DATABASE AND ORDER FROM HIGHEST TO LOWEST

leaderboard\_players = db\_cursor.fetchall()

**if** len(leaderboard\_players) > **10**:

# IF THERE ARE MORE THAN 10 PLAYERS IN THE DATABASE

leaderboard\_players = leaderboard\_players[:**10**]

# GET THE TOP TEN

y\_pos\_odd = **162**

y\_pos\_even = **202**

**for** each\_player **in** range(**0**, len(leaderboard\_players), **2**):

self.text(menu\_font, **24**, BLACK, "{}".format(leaderboard\_players[each\_player][**0**]), GRAY, self.screen, **215**, y\_pos\_odd)

self.text(menu\_font, **24**, BLACK, "{}".format(leaderboard\_players[each\_player][**1**]), GRAY, self.screen, **440**, y\_pos\_odd)

y\_pos\_odd += **80**

**for** each\_player **in** range(**1**, len(leaderboard\_players), **2**):

self.text(menu\_font, **24**, BLACK, "{}".format(leaderboard\_players[each\_player][**0**]), WHITE, self.screen, **215**, y\_pos\_even)

self.text(menu\_font, **24**, BLACK, "{}".format(leaderboard\_players[each\_player][**1**]), WHITE, self.screen, **440**, y\_pos\_even)

y\_pos\_even += **80**

self.button(**30**, **131**, **570**, **35**, WHITE, GRAY, self.screen, menu\_font, **20**, BLACK, "MENU SCREEN", "menu")

self.button(**380**, **131**, **570**, **35**, GRAY, WHITE, self.screen, menu\_font, **20**, BLACK, "QUIT", "quit")

# DRAW PLAYERS ON SCREEN

pygame.display.flip()

**def** **about\_draw\_screen**(self):

##### about\_draw\_screen #######

# Parameters : None

# Return Type : None

# Purpose :- Screen to show players how to play the game

##########################

self.screen.fill(WHITE)

self.screen.blit(self.udlr, (**290**, **20**))

self.screen.blit(self.wasd, (**11**, **0**))

self.text(menu\_font, **25**, BLACK, "To move around click these keys", WHITE, self.screen, **280**, **200**)

self.screen.blit(self.red\_ghost\_image, (**11**, **240**))

self.text(menu\_font, **15**, BLACK, "The red ghost will follow you around the maze chasing your exact position.", WHITE, self.screen, **280**, **250**)

self.screen.blit(self.blue\_ghost\_image, (**11**, **300**))

self.text(menu\_font, **15**, BLACK, "The blue ghost will hover around the closest large pac dot to your location.", WHITE, self.screen, **280**, **310**)

self.screen.blit(self.black\_ghost\_image, (**11**, **360**))

self.text(menu\_font, **13**, BLACK, "The black ghost will move around randomly for 10 seconds then turn to the white ghost.", WHITE, self.screen, **290**, **370**)

self.screen.blit(self.white\_ghost\_image, (**11**, **420**))

self.text(menu\_font, **14**, BLACK, "The white ghost will chase your exact location for 15 seconds", WHITE, self.screen, **250**, **430**)

self.text(menu\_font, **14**, BLACK, " then turn to the black ghost again.", WHITE, self.screen, **190**, **450**)

self.screen.blit(self.orange\_ghost\_image, (**11**, **480**))

self.text(menu\_font, **15**, BLACK, "The orange ghost attemps to cut you off aiming two spaces ahead", WHITE,self.screen, **250**, **490**)

self.screen.blit(self.esc, (**0**, **520**))

self.text(menu\_font, **20**, BLACK, "Press the esc key to pause the game", WHITE, self.screen, **250**, **550**)

self.button(**400**, **131**, **570**, **35**, WHITE, GRAY, self.screen, menu\_font, **20**, BLACK, "MENU SCREEN", "menu")

# DRAW ON SCREEN

pygame.display.flip()

**def** **movement**(self, player):

##### movement #######

# Parameters : None

# Return Type : None

# Purpose :- Controls what happens when user presses certain keys

##########################

key\_pressed = pygame.key.get\_pressed()

# GET THE KEY PRESSED

**if** key\_pressed[pygame.K\_RIGHT] **or** key\_pressed[pygame.K\_d]:

# IF THE PLAYER PRESSES D OR RIGHT

player.move\_right()

# MOVE RIGHT

**elif** key\_pressed[pygame.K\_LEFT] **or** key\_pressed[pygame.K\_a]:

# IF THE PLAYER PRESSES LEFT OR A

player.move\_left()

# MOVE LEFT

**elif** key\_pressed[pygame.K\_UP] **or** key\_pressed[pygame.K\_w]:

# IF THE PLAYER PRESSES W OR UP

player.move\_up()

# MOVE UP

**elif** key\_pressed[pygame.K\_DOWN] **or** key\_pressed[pygame.K\_s]:

# IF THE PLAYER PRESSES DOWN OR S

player.move\_down()

# MOVE DOWN

**elif** key\_pressed[pygame.K\_ESCAPE]:

# IF THE PLAYER PRESSES ESCAPE TO PAUSE THE GAME

player.update\_is\_paused(True)

# PAUSE GAME

**def** **draw\_lives**(self, x\_value, increment\_value, screen, player\_lives, player\_heart\_image):

##### draw\_lives #######

# Parameters :- x\_value:int, increment\_value:int, screen:Canvas, player\_lives:int, player\_heart\_image:Image

# Return Type :- None

# Purpose :- Draws pacman image on screen depending on the amount of lives the player has left

##########################

# FUNCTION THAT DRAWS SMALL PACMAN IMAGES THAT REPRESENT THE AMOUNT OF LIVES THE PLAYER HAS LEFT

**for** eachLife **in** range(**0**, player\_lives):

# FOR EACH LIFE THAT THE PLAYER HAS

screen.blit(player\_heart\_image, (x\_value, **2**))

x\_value += increment\_value

# DRAW THE PACMAN IMAGE ON THE SCREEN AND INCREASE THE X POSITION OF THE IMAGE

**def** **background\_load**(self, photo\_name, image\_x\_length, image\_y\_length):

##### background\_load #######

# Parameters : photo\_name:String, image\_x\_length:int, image\_y\_length:int

# Return Type : background:Image

# Purpose :- Loads in an image and changes image size

##########################

# LOADS IN THE BACKGROUND PHOTO

background = pygame.image.load(photo\_name)

background = pygame.transform.scale(background, (image\_x\_length, image\_y\_length))

**return** background

**def** **text**(self, font, size, colour, message, background\_colour, screen, x\_position, y\_position):

##### text #######

# Parameters : font:Font, size:int, colour:Tuple, message:String, background\_colour:Tuple, screen:Canvas, x\_position:int, y\_position:int

# Return Type : None

# Purpose :- Writes text on screen

##########################

font\_variable = pygame.font.Font(font, size)

message\_variable = font\_variable.render(message, True, colour, background\_colour)

text\_rectangle = message\_variable.get\_rect()

text\_rectangle.center = (x\_position, y\_position)

screen.blit(message\_variable, text\_rectangle)

# DRAW TEXT ON SCREEN

**def** **button**(self, start\_x\_position, increment\_x, start\_y\_position, increment\_y, dark\_colour, light\_colour, screen, font, font\_size, font\_colour, message, game\_state):

##### button #######

# Parameters : start\_x\_position:int, increment\_x:int, start\_y\_position:int, increment\_y:int, dark\_colour:Tuple, light\_colour:Tuple, screen:Canvas, font:Font, font\_size:int, font\_colour:Tuple, message:String, game\_state:String

# Return Type : None

# Purpose :- Creates a clickable button on the screen

##########################

mouse = pygame.mouse.get\_pos()

mouse\_clicked = pygame.mouse.get\_pressed()

**if** start\_x\_position + increment\_x > mouse[**0**] > start\_x\_position **and** start\_y\_position + increment\_y > mouse[**1**] > start\_y\_position:

pygame.draw.rect(screen, light\_colour, (start\_x\_position, start\_y\_position, increment\_x, increment\_y))

self.text(font, font\_size, font\_colour, message, light\_colour, screen, start\_x\_position + int(increment\_x / **2**), start\_y\_position + int(increment\_y / **2**))

**if** mouse\_clicked[**0**] == **1**:

self.state = game\_state

**else**:

pygame.draw.rect(screen, dark\_colour, (start\_x\_position, start\_y\_position, increment\_x, increment\_y))

self.text(font, font\_size, font\_colour, message, dark\_colour, screen, start\_x\_position + int(increment\_x / **2**), start\_y\_position + int(increment\_y / **2**))

# CREATE A BUTTON

**pacman.py**

**from** **sprite\_object\_class** **import** \*

**import** **pygame**

**from** **pygame** **import** mixer

pygame.init()

vector = pygame.math.Vector2

**class** **Pacman**(pygame.sprite.Sprite, moving\_object):

**def** **\_\_init\_\_**(self, starting\_position, maze, gamemode, gamemode\_class):

super().\_\_init\_\_()

self.heart\_image = pygame.image.load(os.path.join(pacman\_images, "pacman\_left\_1.png"))

self.heart\_image = pygame.transform.scale(self.heart\_image, (**15**, **15**))

self.right\_images = [pygame.image.load(os.path.join(pacman\_images, "pacman\_start\_image.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_right\_1.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_right\_2.png"))]

self.left\_images = [pygame.image.load(os.path.join(pacman\_images, "pacman\_start\_image.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_left\_1.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_left\_2.png"))]

self.up\_images = [pygame.image.load(os.path.join(pacman\_images, "pacman\_start\_image.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_up\_1.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_up\_2.png"))]

self.down\_images = [pygame.image.load(os.path.join(pacman\_images, "pacman\_start\_image.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_down\_1.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_down\_2.png"))]

self.death\_images = [pygame.image.load(os.path.join(pacman\_images, "pacman\_start\_image.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_1.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_2.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_3.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_4.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_5.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_6.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_7.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_8.png")), pygame.image.load(os.path.join(pacman\_images, "pacman\_death\_9.png"))]

# LOAD IN ANIMATION IMAGES

self.gamemode = gamemode

self.maze\_class = maze

self.image\_loop = **0**

self.image = self.right\_images[self.image\_loop]

self.image = pygame.transform.scale(self.image, (cell\_width-**5**, cell\_height-**5**))

self.image.set\_colorkey(BLACK)

self.rect = self.image.get\_rect()

self.rect.center = (starting\_position[**0**], starting\_position[**1**])

self.\_\_pixel\_position = vector(starting\_position[**0**] - cell\_width//**2**, starting\_position[**1**]-cell\_height//**2**)

self.\_\_direction = vector(**0**, **0**)

self.\_\_update\_direction = (**0**, **0**)

self.\_\_lives = **3**

self.\_\_score = **0**

self.\_\_level = **1**

self.\_\_orientation = None

self.\_\_timer = **0**

self.\_\_is\_alive = True

self.\_\_level\_over = False

self.\_\_gamemode\_class = gamemode\_class

self.\_\_is\_paused = False

self.\_\_has\_started = False

# PACMAN VARIABLES

**def** **update**(self):

##### update #######

# Parameters : None

# Return Type : None

# Purpose :- Updates the pacman

##########################

# UPDATE PACMAN FUNCTION

self.update\_pixel\_position(vector((self.rect.x // cell\_width) \* cell\_width, (self.rect.y // cell\_width) \* cell\_width))

# UPDATE PACMAN PIXEL POSITION

**if** self.gamemode == "singleplayer":

# IF THE GAMEMODE IS FOR SINGLEPLAYER

self.sinlgeplayer\_update()

# RUN UPDATE FUNCTION FOR SINGLEPLAYER

**if** self.gamemode == "multiplayer":

# IF THE GAMEMODE IS FOR SINGLEPLAYER

self.multiplayer\_update()

# RUN UPDATE FUNCTION FOR SINGLEPLAYER

# ANIMATIONS

**def** **move\_right**(self):

##### move\_right #######

# Parameters : None

# Return Type : None

# Purpose :- Update pacman direction to move right

##########################

# MOVE RIGHT

self.update\_direction((**1**, **0**))

# UPDATE DIRECTION

**def** **move\_left**(self):

##### move\_left #######

# Parameters : None

# Return Type : None

# Purpose :- Update pacman direction to move left

##########################

# MOVE LEFT

self.update\_direction((-**1**, **0**))

# UPDATE DIRECTION

**def** **move\_up**(self):

##### move\_up #######

# Parameters : None

# Return Type : None

# Purpose :- Update pacman direction to move up

##########################

# MOVE UP

self.update\_direction((**0**, -**1**))

# UPDATE DIRECTION

**def** **move\_down**(self):

##### move\_down #######

# Parameters : None

# Return Type : None

# Purpose :- Update pacman direction to move down

##########################

# MOVE DOWN

self.update\_direction((**0**, **1**))

# UPDATE DIRECTION

**def** **reset**(self, starting\_position):

##### reset #######

# Parameters : starting\_position:Tuple

# Return Type : None

# Purpose :- Resets position and game variables

##########################

self.rect.center = (starting\_position[**0**], starting\_position[**1**])

self.update\_pixel\_position(vector((self.rect.x // cell\_width) \* cell\_width, (self.rect.y // cell\_width) \* cell\_width))

self.update\_direction((**0**, **0**))

self.update\_update\_direction((**0**, **0**))

self.update\_is\_alive(True)

self.update\_orientation(None)

self.update\_timer(**0**)

self.update\_is\_paused(False)

# RESET ALL POSITIONS AND VARIABLES

**if** self.get\_level\_over():

# IF THE LEVEL IS OVER

**if** self.get\_lives() <= **4** **and** self.get\_level() >= **3**:

# IF THE PLAYER HAS LESS THAN 5 LIVES AND IS PAST LEVEL THREE

self.update\_lives(self.get\_lives() + **1**)

**print**("done")

# GIVE THEM ANOTHER LIFE

self.update\_level(self.get\_level() + **1**)

# INCREASE THE PLAYERS LEVEL

self.update\_level\_over(False)

# CHANGE LEVEL OVER VARIABLE BACK TO DEFAULT

**def** **reset\_game**(self, starting\_position):

##### reset\_game #######

# Parameters : starting\_position:Tuple

# Return Type : None

# Purpose :- Reset entire game back to beginning

##########################

self.reset(starting\_position)

self.image\_loop = **0**

self.update\_lives(**3**)

self.update\_score(**0**)

self.update\_level(**1**)

self.update\_has\_started(False)

# RESET ALL PLAYER VARIABLES BACK TO DEFAULT

**def** **sinlgeplayer\_update**(self):

##### singleplayer\_update #######

# Parameters : None

# Return Type : None

# Purpose :- Update logic for a singlepayer game

##########################

**if** self.get\_is\_alive():

# IF THE PLAYER IS ALIVE

self.orientation\_images(self.get\_direction(), self.right\_images, self.left\_images, self.up\_images, self.down\_images, **0.15**, **5**, **5**)

# UPDATE IMAGE DEPENDING ON ORIENTATION

**if** **not** self.get\_is\_paused() **and** self.get\_has\_started():

# IF THE PLAYER IS NOT PAUSED AND THE GAME HAS STARTED

self.move()

# MOVE THE PLAYER

**if** self.get\_pixel\_position() **in** self.maze\_class.dots\_list:

# IF THE PLAYER COLLIDES WITH A PACDOT

self.maze\_class.dots\_list.remove(self.get\_pixel\_position())

self.update\_score(self.get\_score() + **5**)

waka\_waka = mixer.Sound(os.path.join(music\_folder, "wakawaka.ogg"))

waka\_waka.set\_volume(**0.15**)

waka\_waka.play()

# PLAY MUSIC AND REMOVE THAT DOT FROM SCREEN

**if** self.get\_pixel\_position() **in** self.maze\_class.special\_dots\_list:

# IF THE PLAYER COLLIDES WITH A LARGE PAC DOT

self.maze\_class.special\_dots\_list.remove(self.get\_pixel\_position())

self.update\_score(self.get\_score() + **20**)

self.\_\_gamemode\_class.red\_ghost.update\_is\_frightened(True)

self.\_\_gamemode\_class.blue\_ghost.update\_is\_frightened(True)

self.\_\_gamemode\_class.black\_ghost.update\_is\_frightened(True)

self.\_\_gamemode\_class.orange\_ghost.update\_is\_frightened(True)

self.update\_timer(**0**)

# REMOVE THAT DOT FROM THE SCREEN AND MAKE ALL THE GHOST FRIGHTENED

**if** self.get\_pixel\_position() **in** self.maze\_class.fruit\_positions:

# IF THE PLAYER EATS A FRUIT

self.maze\_class.fruit\_images.remove(self.maze\_class.fruit\_images[self.maze\_class.fruit\_positions.index(self.get\_pixel\_position())])

self.maze\_class.fruit\_positions.remove(self.get\_pixel\_position())

self.update\_score(self.get\_score() + **15**)

# GIVE THEM 15 POINTS AND REMOVE FRUIT FROM SCREEN

**if** len(self.maze\_class.dots\_list) == **0** **and** len(self.maze\_class.special\_dots\_list) == **0**:

# IF THE PLAYER HAS EATEN ALL THE DOTS

self.update\_level\_over(True)

self.update\_has\_started(False)

self.update\_timer(**0**)

# THE LEVEL IS OVER

**def** **multiplayer\_update**(self):

##### multiplayer\_update #######

# Parameters : None

# Return Type : None

# Purpose :- Update logic for a multiplayer game

##########################

**if** **not** self.get\_is\_paused() **and** self.get\_has\_started():

# IF THE PLAYER IS NOT PAUSED AND THE GAME HAS STARTED

self.move()

# MOVE

# UPDATE ANIMATIONS

**if** self.get\_is\_alive():

# IF THE PLAYER IS ALIVE

self.orientation\_images(self.get\_direction(), self.right\_images, self.left\_images, self.up\_images, self.down\_images, **0.15**, **5**, **5**)

# ANIMATE THEIR IMAGE DEPENDING ON ORIENTATION

**def** **move**(self):

##### move #######

# Parameters : None

# Return Type : None

# Purpose :- Move the player on the screen and check for walls

##########################

self.rect.center += vector(self.get\_update\_direction())

# MOVE THE PLAYER

**if** self.get\_direction() == vector(-self.get\_update\_direction()[**0**], -self.get\_update\_direction()[**1**]):

# IF THE PLAYER ATTEMPTS TO MOVE IN THE OPPOSITE DIRECTION

self.update\_update\_direction(self.get\_direction())

# UPDATE RIGHT AWAY

**if** (self.rect.center[**0**] - cell\_width // **2**) % cell\_width == **0** **and** (self.rect.center[**1**] - cell\_width // **2**) % cell\_height == **0**:

# IF THE PLAYER IS IN THE MIDDLE OF THE CELL

**if** self.get\_direction() != vector(**0**, **0**):

# IF THEIR DIRECTION IS NOT 0

tempA, tempB = self.wall\_check(self.get\_direction(), self.maze\_class.walls, self.rect)

self.update\_update\_direction(tempA)

self.update\_pixel\_position(tempB)

# RUN WALL CHECK METHOD AND UPDATE DIRECTION

**if** self.get\_update\_direction() == (**1**, **0**):

self.update\_orientation("right")

# UPDATE ORIENTATION

**elif** self.get\_update\_direction() == (-**1**, **0**):

self.update\_orientation("left")

# UPDATE ORIENTATION

**elif** self.get\_update\_direction() == (**0**, -**1**):

self.update\_orientation("up")

# UPDATE ORIENTATION

**elif** self.get\_update\_direction() == (**0**, **1**):

self.update\_orientation("down")

# UPDATE ORIENTATION

**def** **force\_pause**(self):

##### force\_pause #######

# Parameters : None

# Return Type : None

# Purpose :- Stops the player where it is and resets time

##########################

self.rect.center = (self.get\_pixel\_position()[**0**] + (cell\_width//**2**), self.get\_pixel\_position()[**1**] + (cell\_height//**2**))

self.update\_update\_direction((**0**, **0**))

self.update\_timer(**0**)

# FREEZE THE PLAYER IN PLACE

**def** **get\_lives**(self):

##### get\_lives #######

# Parameters : None

# Return Type : \_\_lives:int

# Purpose :- Get amount of lives

##########################

**return** self.\_\_lives

**def** **update\_lives**(self, new\_lives):

##### update\_lives #######

# Parameters : new\_lives:int

# Return Type : None

# Purpose :- Update \_\_lives variable

##########################

self.\_\_lives = new\_lives

**def** **get\_score**(self):

##### get\_score #######

# Parameters : None

# Return Type : \_\_score:int

# Purpose :- Get player score

##########################

**return** self.\_\_score

**def** **update\_score**(self, new\_score):

##### update\_score #######

# Parameters : new\_score:int

# Return Type : None

# Purpose :- Update player score

##########################

self.\_\_score = new\_score

**def** **get\_level**(self):

##### get\_level #######

# Parameters : None

# Return Type : \_\_level:int

# Purpose :- Get player level

##########################

**return** self.\_\_level

**def** **update\_level**(self, new\_level):

##### update\_level #######

# Parameters : new\_level:int

# Return Type : None

# Purpose :- Update player level

##########################

self.\_\_level = new\_level

**def** **get\_timer**(self):

##### get\_timer #######

# Parameters : None

# Return Type : \_\_timer:int

# Purpose :- Get player timer variable

##########################

**return** self.\_\_timer

**def** **update\_timer**(self, timer):

##### update\_timer #######

# Parameters : timer:int

# Return Type : None

# Purpose :- Update player timer variable

##########################

self.\_\_timer = timer

**def** **get\_is\_alive**(self):

##### get\_is\_alive #######

# Parameters : None

# Return Type : \_\_is\_alive:Boolean

# Purpose :- Get is\_alive

##########################

**return** self.\_\_is\_alive

**def** **update\_is\_alive**(self, alive):

##### update\_is\_alive #######

# Parameters : alive:Boolean

# Return Type : None

# Purpose :- Update is\_alive variable

##########################

self.\_\_is\_alive = alive

**def** **get\_level\_over**(self):

##### get\_level\_over #######

# Parameters : None

# Return Type : \_\_level\_over: Boolean

# Purpose :- Gets level\_over variable

##########################

**return** self.\_\_level\_over

**def** **update\_level\_over**(self, level\_over):

##### update\_level\_over #######

# Parameters : level\_over:Boolean

# Return Type : None

# Purpose :- Update level\_over variable

##########################

self.\_\_level\_over = level\_over

**def** **get\_is\_paused**(self):

##### get\_is\_paused #######

# Parameters : None

# Return Type : \_\_is\_paused:Boolean

# Purpose :- Get is\_paused variable

##########################

**return** self.\_\_is\_paused

**def** **update\_is\_paused**(self, is\_paused):

##### update\_is\_paused #######

# Parameters : is\_paused:Boolean

# Return Type : None

# Purpose :- Update is\_paused variable

##########################

self.\_\_is\_paused = is\_paused

**def** **get\_orientation**(self):

##### get\_orientation #######

# Parameters : None

# Return Type : \_\_orientation:String

# Purpose :- Get player orientation

##########################

**return** self.\_\_orientation

**def** **update\_orientation**(self, orientation):

##### update\_orientation #######

# Parameters : orientation:String

# Return Type : None

# Purpose :- Update orientation variable

##########################

self.\_\_orientation = orientation

**def** **get\_has\_started**(self):

##### get\_has\_started #######

# Parameters : None

# Return Type : has\_started:Boolean

# Purpose :- Get has\_started variable

##########################

**return** self.\_\_has\_started

**def** **update\_has\_started**(self, has\_started):

##### update\_has\_started #######

# Parameters : has\_started:Boolean

# Return Type : None

# Purpose :- Update has\_started variable

##########################

self.\_\_has\_started = has\_started

**def** **get\_update\_direction**(self):

##### get\_update\_direction #######

# Parameters :- None

# Return Type :- \_\_update\_direction:Tuple

# Purpose :- Get update direction variable

##########################

**return** self.\_\_update\_direction

**def** **update\_update\_direction**(self, update\_direction):

##### update\_update\_direction #######

# Parameters : update\_direction:Tuple

# Return Type : None

# Purpose :- Update update\_direction variable

##########################

self.\_\_update\_direction = update\_direction

**def** **get\_direction**(self):

##### get\_direction #######

# Parameters : None

# Return Type : \_\_direction:Tuple

# Purpose :- Get direction variable

##########################

**return** self.\_\_direction

**def** **update\_direction**(self, direction):

##### update\_direction #######

# Parameters : direction:Tuple

# Return Type : None

# Purpose :- Update direction variable

##########################

self.\_\_direction = direction

**def** **get\_pixel\_position**(self):

##### get\_pixel\_position #######

# Parameters : None

# Return Type : \_\_pixel\_position:Tuple

# Purpose :- Gets pixel\_position variable

##########################

**return** self.\_\_pixel\_position

**def** **update\_pixel\_position**(self, pixel\_position):

##### update\_pixel\_position #######

# Parameters : pixel\_position:Tuple

# Return Type : None

# Purpose :- Update pixel\_position variable

##########################

self.\_\_pixel\_position = pixel\_position

**ghosts.py**

**import** **random**

**from** **sprite\_object\_class** **import** \*

vector = pygame.math.Vector2

**class** **Ghost**(pygame.sprite.Sprite, moving\_object):

**def** **\_\_init\_\_**(self, right\_images, left\_images, up\_images, down\_images, starting\_position, pacman, colour, gamemode):

super().\_\_init\_\_()

# Initialize Sprite Class

self.right\_images = right\_images

self.left\_images = left\_images

self.up\_images = up\_images

self.down\_images = down\_images

self.frightened\_mode\_images = [pygame.image.load(os.path.join(ghost\_images, "frightenedmode\_1.png")), pygame.image.load(os.path.join(ghost\_images, "frightenedmode\_2.png"))]

self.flicker\_mode\_images = [pygame.image.load(os.path.join(ghost\_images, "frightenedmode\_1.png")), pygame.image.load(os.path.join(ghost\_images, "frightenedmode\_2.png")), pygame.image.load(os.path.join(ghost\_images, "ghostdeath\_1.png")), pygame.image.load(os.path.join(ghost\_images, "ghostdeath\_2.png"))]

self.right\_eye = [pygame.image.load(os.path.join(ghost\_images, "ge\_r.png"))]

self.left\_eye = [pygame.image.load(os.path.join(ghost\_images, "ge\_l.png"))]

self.up\_eye = [pygame.image.load(os.path.join(ghost\_images, "ge\_u.png"))]

self.down\_eye = [pygame.image.load(os.path.join(ghost\_images, "ge\_d.png"))]

self.white\_right = [pygame.image.load(os.path.join(ghost\_images, "whiteghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "whiteghost\_right\_2.png"))]

self.white\_left = [pygame.image.load(os.path.join(ghost\_images, "whiteghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "whiteghost\_left\_2.png"))]

self.white\_up = [pygame.image.load(os.path.join(ghost\_images, "whiteghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "whiteghost\_up\_2.png"))]

self.white\_down = [pygame.image.load(os.path.join(ghost\_images, "whiteghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "whiteghost\_down\_2.png"))]

self.image\_loop = **0**

self.image = self.right\_images[self.image\_loop]

self.image = pygame.transform.scale(self.image, (cell\_width - **4**, cell\_height - **4**))

self.image.set\_colorkey(WHITE)

self.rect = self.image.get\_rect()

self.rect.center = (starting\_position[**0**] + cell\_width // **2**, starting\_position[**1**] + cell\_height // **2**)

self.movement = []

self.priority\_queue = []

self.visited = []

self.pacman = pacman

self.\_\_pixel\_position = vector(starting\_position)

self.\_\_direction = (**0**, **0**)

self.\_\_colour = colour

self.\_\_old\_direction = False

self.\_\_end\_pos = None

self.\_\_start\_pos = None

self.\_\_ghost\_clock = **0**

self.\_\_is\_alive = True

self.\_\_is\_frightened = False

self.\_\_change\_mode = None

self.\_\_gamemode = gamemode

# GHOST VARIABLES

**def** **update**(self):

##### update #######

# Parameters : None

# Return Type : None

# Purpose :- Updates the ghost

##########################

**if** self.\_\_gamemode == "singleplayer":

# IF THE GAME MODE IS SINGLEPLAYER

self.singleplayer\_update()

# RUN THE SINLGEPLAYER UPDATE METHOD

**if** self.\_\_gamemode == "multiplayer":

# IF THE GAME MODE IS MULTIPLAYER

self.multiplayer\_update()

# RUN MULTIPLAYER METHOD

self.priority\_queue = []

self.visited = []

# EMPTY PRIORITY QUEUE AND VISITED LISTS

**def** **search**(self, cell\_to\_check, end\_location, walls):

##### search #######

# Parameters : cell\_to\_check:List, end\_location:Tuple, walls:List

# Return Type : None

# Purpose :- Uses A\* Search algorithm and recursion to find the path

##########################

**if** cell\_to\_check[**1**] == end\_location:

# IF THE CELL IS THE THE END LOCATION

**if** len(self.priority\_queue) == **0**:

# IF THERE ARE NO POSITIONS IN PRIORITY QUEUE

self.visited.append(cell\_to\_check[**1**])

# ADD THIS POSITION TO PRIORITY QUEUE

**else**:

# IF PRIORITY QUEUE IS NOT EMPTY

**for** each\_position **in** self.priority\_queue:

# FOR EACH POSITION IN THE PRIORITY QUEUE

self.visited.append(each\_position)

# ADD THIS POSITION TO VISITED

**else**:

# IF THIS POSITION IS NOT THE END POSITION

**for** each\_position **in** self.adjacent\_search(cell\_to\_check, end\_location, walls):

# RUN ADJACENT\_SEARCH METHOD AND FOR EACH POSITION YOU GET

self.priority\_queue.append(each\_position)

# ADD THAT POSITION TO PRIORITY QUEUE

self.priority\_queue.sort(reverse=True)

# SORT THE PRIORITY QUEUE IN SIZE ORDER

**if** cell\_to\_check **in** self.priority\_queue:

# IF THE POSITION WE HAVE CHECKED IS IN PRIORITY QUEUE

self.priority\_queue.remove(cell\_to\_check)

# REMOVE THIS POSITION FROM PRIORITY QUEUE

self.visited.append(cell\_to\_check)

# ADD THIS POSITION TO VISITED LIST

self.search(self.priority\_queue[-**1**], end\_location, walls)

# RUN THE FUNCTION AGAIN WITH THE CLOSEST POSITION

**def** **movement\_function**(self, visited\_positions):

##### movement\_function #######

# Parameters : visited\_position:List

# Return Type : None

# Purpose :- Backtracks visiting where each cell came from to find the shortest path

##########################

shortest\_path = []

position = visited\_positions[-**1**]

shortest\_path.append(position[**1**])

# ASSIGN VARIABLES

**while** position != visited\_positions[**0**]:

# WHILE THE POSITION IS NOT THE START POSITION

**for** each\_position **in** visited\_positions:

# FOR EACH POSITION IN THE VISITED POSITIONS LIST

**if** each\_position[**1**] == position[-**1**]:

# IF THE POSITION IS WHERE THE CELL CAME FROM

shortest\_path.append(each\_position[**1**])

# ADD THIS POSITION TO SHORTEST PATH

position = each\_position

# UPDATE POSITION VARIABLE

**return** shortest\_path

**def** **direction\_calculator**(self, visited\_positions, direction, position, maze):

##### direction\_calculator #######

# Parameters : visited\_positions:List, direction:Tuple, position:Tuple, maze:object

# Return Type : direction:Tuple

# Purpose :- Calculates the direction the ghost must move in order to get to the player

##########################

**if** len(visited\_positions) == **1**:

# IF THERE IS ONLY ONE POSITION IN VISITED POSITIONS

**if** self.\_\_colour == "blue" **and** self.get\_is\_frightened() **is** False **and** self.get\_is\_alive():

# IF THE BLUE GHOST IS NOT FRIGHTENED AND IS ALIVE

self.update\_end\_pos(self.closest\_distance\_calculator(position, self.pacman.maze\_class.special\_dots\_list))

# UPDATE THE END POS VARIABLE

**else**:

**return** self.random\_movement(direction, maze)

# OTHERWISE RETURN A RANDOM DIRECTION

**else**:

path = self.movement\_function(visited\_positions)

# CALCULATE THE PATH THE GHOST SHOULD TAKE

**if** len(path) >= **2**:

# IF THERE ARE MORE THAN TWO POSITIONS THERE

**if** (path[-**1**][**0**] + (cell\_width//**2**), path[-**1**][**1**] + (cell\_height//**2**)) == self.rect.center:

# IF THE GHOST IS IN THE CENTER OF THE CELL

**if** path[-**2**][**0**] + (cell\_width // **2**) != self.rect.center[**0**]:

# IF THE GHOST HAS TO MOVE IN THE X AXIS

**if** path[-**2**][**0**] + (cell\_width // **2**) > self.rect.center[**0**]:

# IF THE POSITION IS TO THE RIGHT

**return** **1**, **0**

# MOVE RIGHT

**elif** path[-**2**][**0**] + (cell\_width // **2**) < self.rect.center[**0**]:

# IF THE POSITION IS TO THE LEFT

**return** -**1**, **0**

# MOVE LEFT

**elif** path[-**2**][**1**] + (cell\_width // **2**) != self.rect.center[**1**]:

# IF THE GHOST HAS TO MOVE IN THE Y AXIS

**if** path[-**2**][**1**] + (cell\_width // **2**) > self.rect.center[**1**]:

# IF THE POSITION IS DOWN

**return** **0**, **1**

# MOVE DOWN

**elif** path[-**2**][**1**] + (cell\_width // **2**) < self.rect.center[**1**]:

# IF THE POSITION IS UP

**return** **0**, -**1**

# MOVE UP

**else**:

**return** direction

**else**:

**return** direction

**return** direction

**def** **orange\_ghost\_movement**(self, position, maze, pacman):

##### orange\_ghost\_movement #######

# Parameters : position:Tuple, maze:object, pacman:object

# Return Type : position:Tuple

# Purpose :- Find the cell two ahead of the player if possible

##########################

**if** pacman.get\_orientation() == "right":

# IF THE PLAYER IS FACING RIGHT

**if** (position[**0**]+**56**, position[**1**]) **in** maze **and** (position[**0**]+**56**, position[**1**]) != (**224**, **280**):

# IF TWO POSITIONS AHEAD OF THEM IS IN THE MAZE

**return** int(position[**0**]+**56**), int(position[**1**])

# RETURN TWO POSITIONS AHEAD OF THEM

**else**:

# OTHERWISE RETURN THEIR EXACT POSITION

**return** position

**if** pacman.get\_orientation() == "left":

# IF THE PLAYER IS FACING LEFT

**if** (position[**0**]-**56**, position[**1**]) **in** maze **and** (position[**0**]-**56**, position[**1**]) != (**280**, **280**):

# IF TWO POSITIONS AHEAD OF THEM IS IN THE MAZE

**return** int(position[**0**]-**56**), int(position[**1**])

# RETURN TWO POSITIONS AHEAD OF THEM

**else**:

**return** position

# OTHERWISE RETURN THEIR EXACT POSITION

**if** pacman.get\_orientation() == "up":

# IF THE PLAYER IS FACING UP

**if** (position[**0**], position[**1**]-**56**) **in** maze **and** (position[**0**], position[**1**]-**56**) != (**252**, **280**):

# IF TWO POSITIONS AHEAD OF THEM IS IN THE MAZE

**return** int(position[**0**]), int(position[**1**]-**56**)

# RETURN TWO POSITIONS AHEAD OF THEM

**else**:

**return** position

# OTHERWISE RETURN THEIR EXACT POSITION

**if** pacman.get\_orientation() == "down":

**if** (position[**0**], position[**1**]+**56**) **in** maze **and** (position[**0**], position[**1**]+**56**) != (**252**, **280**) **and** (position[**0**], position[**1**]+**56**) != (**224**, **280**) **and** (position[**0**], position[**1**]+**56**) != (**280**, **280**):

# IF TWO POSITIONS AHEAD OF THEM IS IN THE MAZE

**return** int(position[**0**]), int(position[**1**]+**56**)

# RETURN TWO POSITIONS AHEAD OF THEM

**else**:

**return** position

# OTHERWISE RETURN THEIR EXACT POSITION

**def** **random\_movement**(self, direction\_input, maze):

##### random\_movement #######

# Parameters : direction\_input:Tuple, maze:object

# Return Type : direction:Tuple

# Purpose :- Generates a random direction to move in

##########################

**if** (self.rect.center[**0**] - (cell\_width//**2**)) % **28** == **0** **and** (self.rect.center[**1**] - (cell\_height//**2**)) % **28** == **0**:

# IF THE GHOST IS IN THE CENTRE

**if** self.rect.center == (**294**, **294**):

# IF THE GHOST IS AT THE STARTING POSITION

**return** -**1**, **0**

# GO LEFT

**else**:

direction = random.randint(**1**, **4**)

# GENERATE A RANDOM NUMBER

**while** self.get\_old\_direction() == direction:

# WHILE THE DIRECTION IS THE SAME DIRECTION AS LAST TIME

direction = random.randint(**1**, **4**)

# GENERATE A RANDOM NUMBER

self.update\_old\_direction(direction)

# UPDATE OLD DIRECTION

**if** direction == **1** **and** (self.get\_pixel\_position()[**0**] + **28**, self.get\_pixel\_position()[**1**]) **in** maze.maze\_pixel\_list:

# IF THE NUMBER IS ONE AND THE PLAYER DOESNT HAVE A WALL TO THEIR RIGHT

**return** **1**, **0**

# GO RIGHT

**if** direction == **2** **and** (self.get\_pixel\_position()[**0**] - **28**, self.get\_pixel\_position()[**1**]) **in** maze.maze\_pixel\_list:

# IF THE NUMBER IS TWO AND THE PLAYER DOESNT HAVE A WALL TO THEIR LEFT

**return** -**1**, **0**

# GO LEFT

**if** direction == **3** **and** (self.get\_pixel\_position()[**0**], self.get\_pixel\_position()[**1**] + **28**) **in** maze.maze\_pixel\_list:

# IF THE NUMBER IS THREE AND THE PLAYER DOESNT HAVE A WALL BELOW

**return** **0**, **1**

# GO DOWN

**if** direction == **4** **and** (self.get\_pixel\_position()[**0**], self.get\_pixel\_position()[**1**] - **28**) **in** maze.maze\_pixel\_list:

# IF THE NUMBER IS FOUR AND THE PLAYER DOESNT HAVE A WALL ABOVE

**return** **0**, -**1**

# GO UP

**else**:

**return** **0**, **0**

# OTHERWISE DO NOT MOVE

**else**:

**return** direction\_input

# OTHERWISE KEEP MOVING IN THE SAME DIRECTION

**def** **closest\_distance\_calculator**(self, position\_variable, list):

##### closest\_distance\_calculator #######

# Parameters : position\_variable:Tuple, list:List

# Return Type : position:Tuple

# Purpose :- Finds the closest position from a list of positions

##########################

distance = **10000**

position = []

**for** each\_position **in** list:

# LOOP THROUGH ALL POSITIONS IN LIST

my\_distance = **0**

**if** position\_variable[**0**] >= each\_position[**0**]:

# IF THE X POSITION IS TO THE RIGHT

my\_distance += ((position\_variable[**0**]//**28**) - (each\_position[**0**]//**28**))

**else**:

# X POSITION TO THE LEFT

my\_distance += ((each\_position[**0**]//**28**) - (position\_variable[**0**]//**28**))

**if** position\_variable[**1**] >= each\_position[**1**]:

# Y POSITION BELOW

my\_distance += ((position\_variable[**1**] // **28**) - (each\_position[**1**] // **28**))

**else**:

# Y POSITION ABOVE

my\_distance += ((each\_position[**1**] // **28**) - (position\_variable[**1**] // **28**))

**if** my\_distance <= distance:

# IF THE POSITION IS CLOSER

position = each\_position

distance = my\_distance

# UPDATE POSITION AND DISTANCE VARIABLES

**return** position

**def** **adjacent\_search**(self, cell\_to\_check, end\_location, walls):

##### adjacent\_search #######

# Parameters : cell\_to\_check:List, end\_location:Tuple, walls:List

# Return Type : priority\_queue:List

# Purpose :- Finds all the positions that are adjacent

##########################

priority\_queue = []

**if** (cell\_to\_check[**1**][**0**] + **28**, cell\_to\_check[**1**][**1**]) **not** **in** walls **and** cell\_to\_check[**1**][**0**] + **28** < screen\_width **and** (cell\_to\_check[**1**][**0**] + **28**, cell\_to\_check[**1**][**1**]) != cell\_to\_check[**2**] **and** (cell\_to\_check[**1**][**0**] + **28**, cell\_to\_check[**1**][**1**]) **not** **in** self.visited:

# IF THE CELL IS IN THE MAZE AND NOT A WALL AND HAS NOT BEEN SEARCHED BEFORE

absolute\_distance = abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector((cell\_to\_check[**1**][**0**] + **28**) // **28**, cell\_to\_check[**1**][**1**] // **28**))[**0**])) + abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector((cell\_to\_check[**1**][**0**] + **28**) // **28**, cell\_to\_check[**1**][**1**] // **28**))[**1**]))

# CALCULATE THE MANHATTAN DISTANCE

**if** end\_location[**0**] > (cell\_to\_check[**1**][**0**] + **28**):

# IF YOU ARE MOVING IN THE RIGHT DIRECTION

absolute\_distance -= **1**

# REMOVE ONE OFF DISTANCE

priority\_queue.append((absolute\_distance, (int(cell\_to\_check[**1**][**0**] + **28**), int(cell\_to\_check[**1**][**1**])), (int(cell\_to\_check[**1**][**0**]), int(cell\_to\_check[**1**][**1**]))))

# ADD POSITION TO PRIORITY QUEUE

**if** (cell\_to\_check[**1**][**0**] - **28**, cell\_to\_check[**1**][**1**]) **not** **in** walls **and** cell\_to\_check[**1**][**0**] - **28** > **0** **and** (cell\_to\_check[**1**][**0**] - **28**, cell\_to\_check[**1**][**1**]) != cell\_to\_check[**2**] **and** (cell\_to\_check[**1**][**0**] - **28**, cell\_to\_check[**1**][**1**]) **not** **in** self.visited:

# IF THE CELL IS IN THE MAZE AND NOT A WALL AND HAS NOT BEEN SEARCHED BEFORE

absolute\_distance = abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector((cell\_to\_check[**1**][**0**] - **28**) // **28**, cell\_to\_check[**1**][**1**] // **28**))[**0**])) + abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector((cell\_to\_check[**1**][**0**] - **28**) // **28**, cell\_to\_check[**1**][**1**] // **28**))[**1**]))

# CALCULATE THE MANHATTAN DISTANCE

**if** end\_location[**0**] < (cell\_to\_check[**1**][**0**] - **28**):

# IF YOU ARE MOVING IN THE RIGHT DIRECTION

absolute\_distance -= **1**

# REMOVE ONE OFF DISTANCE

priority\_queue.append((absolute\_distance, (int(cell\_to\_check[**1**][**0**] - **28**), int(cell\_to\_check[**1**][**1**])), (int(cell\_to\_check[**1**][**0**]), int(cell\_to\_check[**1**][**1**]))))

# ADD POSITION TO PRIORITY QUEUE

**if** (cell\_to\_check[**1**][**0**], cell\_to\_check[**1**][**1**] + **28**) **not** **in** walls **and** cell\_to\_check[**1**][**1**] + **28** < screen\_length **and** (cell\_to\_check[**1**][**0**], cell\_to\_check[**1**][**1**] + **28**) != cell\_to\_check[**2**] **and** (cell\_to\_check[**1**][**0**], cell\_to\_check[**1**][**1**] + **28**) **not** **in** self.visited:

# IF THE CELL IS IN THE MAZE AND NOT A WALL AND HAS NOT BEEN SEARCHED BEFORE

absolute\_distance = abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector(cell\_to\_check[**1**][**0**] // **28**, (cell\_to\_check[**1**][**1**] + **28**) // **28**))[**0**])) + abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector(cell\_to\_check[**1**][**0**] // **28**, (cell\_to\_check[**1**][**1**] + **28**) // **28**))[**1**]))

# CALCULATE THE MANHATTAN DISTANCE

**if** end\_location[**1**] > (cell\_to\_check[**1**][**1**] + **28**):

# IF YOU ARE MOVING IN THE RIGHT DIRECTION

absolute\_distance -= **1**

# REMOVE ONE OFF DISTANCE

priority\_queue.append((absolute\_distance, (int(cell\_to\_check[**1**][**0**]), int(cell\_to\_check[**1**][**1**] + **28**)), (int(cell\_to\_check[**1**][**0**]), int(cell\_to\_check[**1**][**1**]))))

# ADD POSITION TO PRIORITY QUEUE

**if** (cell\_to\_check[**1**][**0**], cell\_to\_check[**1**][**1**] - **28**) **not** **in** walls **and** cell\_to\_check[**1**][**1**] - **28** > **0** **and** (cell\_to\_check[**1**][**0**], cell\_to\_check[**1**][**1**] - **28**) != cell\_to\_check[**2**] **and** (cell\_to\_check[**1**][**0**], cell\_to\_check[**1**][**1**] - **28**) **not** **in** self.visited:

# IF THE CELL IS IN THE MAZE AND NOT A WALL AND HAS NOT BEEN SEARCHED BEFORE

absolute\_distance = abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector(cell\_to\_check[**1**][**0**] // **28**, (cell\_to\_check[**1**][**1**] - **28**) // **28**))[**0**])) + abs(int((vector(end\_location[**0**] // **28**, end\_location[**1**] // **28**) - vector(cell\_to\_check[**1**][**0**] // **28**, (cell\_to\_check[**1**][**1**] - **28**) // **28**))[**1**]))

# CALCULATE THE MANHATTAN DISTANCE

**if** end\_location[**1**] < (cell\_to\_check[**1**][**1**] - **28**):

# IF YOU ARE MOVING IN THE RIGHT DIRECTION

absolute\_distance -= **1**

# REMOVE ONE OFF DISTANCE

priority\_queue.append((absolute\_distance, (int(cell\_to\_check[**1**][**0**]), int(cell\_to\_check[**1**][**1**] - **28**)), (int(cell\_to\_check[**1**][**0**]), int(cell\_to\_check[**1**][**1**]))))

# ADD POSITION TO PRIORITY QUEUE

**return** priority\_queue

# INHERITANCE

**def** **reset**(self, starting\_position):

##### reset #######

# Parameters : starting\_position: Tuple

# Return Type : None

# Purpose :- Reset starting position and starting variables

##########################

self.rect.center = (starting\_position[**0**] + cell\_width // **2**, starting\_position[**1**] + cell\_height // **2**)

self.update\_direction((**0**, **0**))

self.update\_old\_direction(False)

self.update\_end\_pos(None)

self.update\_start\_pos(None)

self.update\_change\_mode(None)

self.update\_ghost\_clock(**0**)

self.update\_is\_alive(True)

self.update\_is\_frightened(False)

# RESET ALL VARIABLES TO STARTING VALUES

**def** **is\_running**(self, position, walls):

##### is\_running #######

# Parameters : position:Tuple, walls:List

# Return Type : None

# Purpose :- Makes the ghost run to the ghost cage

##########################

self.search((**0**, self.get\_pixel\_position(), self.get\_pixel\_position()), (**252**, **280**), walls)

# CALCULATE HOW TO GET TO GHOST CAGE

tempA = self.direction\_calculator(self.visited, self.get\_direction(), position, walls)

self.update\_direction(tempA)

# MOVE THERE

**def** **reset\_game**(self, starting\_position):

##### reset\_game #######

# Parameters : starting\_position:Tuple

# Return Type : None

# Purpose :- Resets all variables

##########################

self.reset(starting\_position)

self.movement = []

self.priority\_queue = []

self.visited = []

# RESET ALL VARIABLES

**def** **singleplayer\_update**(self):

##### singleplayer\_update #######

# Parameters : None

# Return Type : None

# Purpose :- Logic for singleplayer game

##########################

self.image\_animator()

# MAKE GHOST LOOK ANIMATED

**if** **not** self.pacman.get\_is\_paused() **and** self.pacman.get\_has\_started():

# IF THE GAME IS NOT PAUSED AND THE PLAYER HAS STARTED

self.ghost\_essentials()

# THE GHOST WILL BEGIN THE GAME

**if** self.get\_pixel\_position() == self.pacman.get\_pixel\_position():

# IF THE GHOST AND PLAYER COLLIDE

**if** self.get\_is\_frightened():

# IF THE GHOST IS FRIGHTENED

self.update\_is\_alive(False)

# KILL GHOST

**else**:

self.pacman.update\_is\_alive(False)

self.pacman.update\_timer(**0**)

self.pacman.update\_has\_started(False)

# KILL PLAYER

**if** self.get\_pixel\_position() == (**252**, **280**) **and** **not** self.get\_is\_alive():

# IF THE GHOST MAKES IT TO THE GHOST CAGE AND IS DEAD

self.update\_is\_alive(True)

self.update\_is\_frightened(False)

self.pacman.update\_score(self.pacman.get\_score() + **15**)

# BRING IT BACK TO LIFE AND RESET FRIGHTENED AND IS ALIVE VARIABLES

**def** **multiplayer\_update**(self):

##### multiplayer\_update #######

# Parameters : None

# Return Type : None

# Purpose :- Logic for multiplayer game

##########################

self.image\_animator()

# MAKE THE PLAYER LOOK ANIMATED

self.ghost\_essentials()

# GHOST WILL UPDATE POSITION AND TIME

**if** self.get\_pixel\_position() == (**252**, **280**) **and** **not** self.get\_is\_alive():

# IF THE GHOST REACHES THE CAGE

self.update\_is\_alive(True)

self.update\_is\_frightened(False)

# BRING IT BACK TO LIFE

**def** **ghost\_essentials**(self):

##### ghost\_essentials #######

# Parameters : None

# Return Type : None

# Purpose :- Moves ghost, updates ghost clock and pixel position

##########################

self.update\_ghost\_clock(self.get\_ghost\_clock() + **1**)

# INCREASE TIME BY ONE

self.update\_pixel\_position((int((self.rect.center[**0**] // cell\_width) \* cell\_width), int((self.rect.center[**1**] // cell\_width) \* cell\_width)))

# UPDATE THE PIXEL POSITION

self.rect.center += vector(self.get\_direction())

# MOVE THE GHOST

**def** **force\_pause**(self):

##### force\_pause #######

# Parameters : None

# Return Type : None

# Purpose :- Freezes the ghost in the middle of the cell

##########################

self.rect.center = (self.get\_pixel\_position()[**0**] + (cell\_width // **2**), self.get\_pixel\_position()[**1**] + (cell\_height // **2**))

# FREEZE THE GHOST IN PLACE

**def** **image\_animator**(self):

##### image\_animator #######

# Parameters : None

# Return Type : None

# Purpose :- Makes the ghost image looks animated depending on its orientation and certain variables

##########################

**if** **not** self.get\_is\_alive():

# IF THE GHOST IS DEAD

self.orientation\_images(self.get\_direction(), self.right\_eye, self.left\_eye, self.up\_eye, self.down\_eye, **0.15**, **0**, **0**)

# UPDATE TO EYES IMAGES

**else**:

**if** self.get\_change\_mode() **and** **not** self.get\_is\_frightened():

# IF THE BLACK GHOST HAS CHANGED MODE AND IS NOT FRIGHTENED

self.orientation\_images(self.get\_direction(), self.white\_right, self.white\_left, self.white\_up, self.white\_down, **0.15**, **4**, **4**)

# DISPLAY WHITE GHOST

**else**:

**if** **not** self.get\_is\_frightened():

# IF THE GHOST IS NOT FRIGHTENED

self.orientation\_images(self.get\_direction(), self.right\_images, self.left\_images, self.up\_images, self.down\_images, **0.15**, **4**, **4**)

# DISPLAY GHOST IMAGES DEPENDING ON ORIENTATION

**def** **get\_change\_mode**(self):

##### get\_change\_mode #######

# Parameters : None

# Return Type : \_\_change\_mode:Boolean

# Purpose :- Get change\_mode variable

##########################

**return** self.\_\_change\_mode

**def** **update\_change\_mode**(self, change\_mode):

##### update\_change\_mode #######

# Parameters : change\_mode:Boolean

# Return Type : None

# Purpose :- Updates change\_mode variable

##########################

self.\_\_change\_mode = change\_mode

**def** **get\_is\_frightened**(self):

##### get\_is\_frightened #######

# Parameters : None

# Return Type : \_\_is\_frightened:Boolean

# Purpose :- Gets is\_frightened variable

##########################

**return** self.\_\_is\_frightened

**def** **update\_is\_frightened**(self, is\_frightened):

##### update\_is\_frightened #######

# Parameters : is\_frightened:Boolean

# Return Type : None

# Purpose :- Updates is\_frightened variable

##########################

self.\_\_is\_frightened = is\_frightened

**def** **get\_is\_alive**(self):

##### get\_is\_alive #######

# Parameters : None

# Return Type : \_\_is\_alive:Boolean

# Purpose :- Gets is\_alive variable

##########################

**return** self.\_\_is\_alive

**def** **update\_is\_alive**(self, is\_alive):

##### update\_is\_alive #######

# Parameters : is\_alive:Boolean

# Return Type : None

# Purpose :- Update is alive variable

##########################

self.\_\_is\_alive = is\_alive

**def** **get\_ghost\_clock**(self):

##### get\_ghost\_clock #######

# Parameters : None

# Return Type : \_\_ghost\_clock:int

# Purpose :- Get ghost\_clock variable

##########################

**return** self.\_\_ghost\_clock

**def** **update\_ghost\_clock**(self, ghost\_clock):

##### update\_ghost\_clock #######

# Parameters : ghost\_clock:int

# Return Type : None

# Purpose :- Update ghost clock variable

##########################

self.\_\_ghost\_clock = ghost\_clock

**def** **get\_start\_pos**(self):

##### get\_start\_pos #######

# Parameters : None

# Return Type : \_\_start\_pos:Tuple

# Purpose :- Gets \_\_start\_pos variable

##########################

**return** self.\_\_start\_pos

**def** **update\_start\_pos**(self, start\_pos):

##### update\_start\_pos #######

# Parameters : start\_pos:Tuple

# Return Type : None

# Purpose :- Updates start\_pos variable

##########################

self.\_\_start\_pos = start\_pos

**def** **get\_end\_pos**(self):

##### get\_end\_pos #######

# Parameters : None

# Return Type : \_\_end\_pos:Tuple

# Purpose :- Gets end\_pos variable

##########################

**return** self.\_\_end\_pos

**def** **update\_end\_pos**(self, end\_pos):

##### update\_end\_pos #######

# Parameters : end\_pos:Tuple

# Return Type : None

# Purpose :- Updates end pos variable

##########################

self.\_\_end\_pos = end\_pos

**def** **get\_old\_direction**(self):

##### get\_old\_direction #######

# Parameters : None

# Return Type : \_\_old\_dirction:Tuple

# Purpose :- Get old\_direction variable

##########################

**return** self.\_\_old\_direction

**def** **update\_old\_direction**(self, old\_direction):

##### update\_old\_direction #######

# Parameters : old\_direction:Tuple

# Return Type : None

# Purpose :- Update old\_direction variable

##########################

self.\_\_old\_direction = old\_direction

**def** **get\_direction**(self):

##### get\_direction #######

# Parameters : None

# Return Type : \_\_direction:Tuple

# Purpose :- Gets direction variable

##########################

**return** self.\_\_direction

**def** **update\_direction**(self, direction):

##### update\_direction #######

# Parameters : direction:Tuple

# Return Type : None

# Purpose :- Update direction variable

##########################

self.\_\_direction = direction

**def** **get\_pixel\_position**(self):

##### get\_pixel\_position #######

# Parameters : None

# Return Type : \_\_pixel\_position:Tuple

# Purpose :- Gets pixel position variable

##########################

**return** self.\_\_pixel\_position

**def** **update\_pixel\_position**(self, pixel\_position):

##### update\_pixel\_position #######

# Parameters : pixel\_position:Tuple

# Return Type : None

# Purpose :- Updates pixel position variable

##########################

self.\_\_pixel\_position = pixel\_position

**singleplayer.py**

**from** **pacman** **import** \*

**from** **ghosts** **import** \*

**from** **maze** **import** \*

**import** **sqlite3**

**import** **pygame**

**from** **pygame** **import** mixer

pygame.init()

**class** **singleplayer**:

**def** **\_\_init\_\_**(self, player\_class):

self.all\_sprites = pygame.sprite.Group()

self.maze\_class = Maze()

self.maze\_class.random\_fruit(self.maze\_class.dots\_list, self.maze\_class.fruit\_positions, self.maze\_class.fruit\_images)

self.player\_class = player\_class

self.pacman = Pacman((starting\_position\_x, starting\_position\_y), self.maze\_class, "singleplayer", self)

self.red\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "redghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "redghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "redghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "redghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_down\_2.png"))], red\_ghost\_starting\_position, self.pacman, "red", "singleplayer")

self.blue\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "blueghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blueghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blueghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blueghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_down\_2.png"))], blue\_ghost\_starting\_position, self.pacman, "blue", "singleplayer")

self.orange\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "orangeghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "orangeghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "orangeghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "orangeghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_down\_2.png"))], orange\_ghost\_starting\_position, self.pacman, "orange", "singleplayer")

self.black\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "blackghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blackghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blackghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blackghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_down\_2.png"))], black\_ghost\_starting\_position, self.pacman, "black", "singleplayer")

# CREATING NEW INSTANCES OF GHOSTS PACMAN AND MAZE

self.all\_sprites.add(self.red\_ghost)

self.all\_sprites.add(self.blue\_ghost)

self.all\_sprites.add(self.orange\_ghost)

self.all\_sprites.add(self.black\_ghost)

self.all\_sprites.add(self.pacman)

# ADDS ALL SPRITES TO SPRITES LIST

self.play\_music\_once = True

# EVENTS

**def** **event\_manager**(self):

##### event\_manager #######

# Parameters : None

# Return Type : None

# Purpose :- Controls entire game events

##########################

**if** **not** self.pacman.get\_is\_paused():

# IF THE GAME IS NOT PAUSED

**if** **not** self.pacman.get\_level\_over():

# IF THE PLAYER HAS NOT EATEN ALL THE DOTS

**if** self.pacman.get\_is\_alive():

# IF THE PLAYER IS STILL ALIVE

**if** self.pacman.get\_has\_started():

# IF THE PLAYER HAS STARTED

**if** self.play\_music\_once:

mixer.music.load(os.path.join(music\_folder, "background\_music.ogg"))

mixer.music.play(-**1**)

self.play\_music\_once = False

self.player\_class.movement(self.pacman)

# MOVE THE PLAYER

# RED GHOST MOVEMENT

**if** self.red\_ghost.get\_is\_frightened():

# IF THE RED GHOST IS FRIGHTENED

**if** self.red\_ghost.get\_is\_alive():

# IF THE RED GHOST IS ALIVE

**if** (**12**\*fps) >= self.pacman.get\_timer():

# IF IT HASN'T BEEN 12 SECONDS

self.red\_ghost.search((**0**, self.red\_ghost.get\_pixel\_position(), self.red\_ghost.get\_pixel\_position()), (**28**, **28**), self.maze\_class.walls)

self.red\_ghost.update\_direction(self.red\_ghost.direction\_calculator(self.red\_ghost.visited, self.red\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# MAKE THE GHOST RUN TO ITS CORNER

**if** self.pacman.get\_timer() >= (**10** \* fps):

# IF THE GHOST IS IN ITS REMAINING 2 SECONDS OF BEING FRIGHTENED

self.red\_ghost.update\_sprite(self.red\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# CHANGE THE IMAGE TO MAKE IT FLICKER

**else**:

self.red\_ghost.update\_sprite(self.red\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

# OR JUST KEEP THE IMAGE AS A FRIGHTENED GHOST

**else**:

self.red\_ghost.update\_is\_frightened(False)

self.pacman.update\_score(self.pacman.get\_score() + **10**)

# ONCE 12 SECONDS HAS PASSED RETURN GHOST TO NORMAL STATE AND GIVE THE PLAYER 10 POINTS

**else**:

self.red\_ghost.is\_running(self.pacman.get\_pixel\_position(), self.maze\_class.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

**if** **not** self.red\_ghost.get\_is\_frightened():

# IF THE RED GHOST IS NOT FRIGHTENED

self.red\_ghost.search((**0**, self.red\_ghost.get\_pixel\_position(), self.red\_ghost.get\_pixel\_position()), self.pacman.get\_pixel\_position(), self.maze\_class.walls)

self.red\_ghost.update\_direction(self.red\_ghost.direction\_calculator(self.red\_ghost.visited, self.red\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# CHASE PLAYER

# BLUE GHOST MOVEMENT

**if** self.blue\_ghost.get\_is\_frightened():

# IF THE BLUE GHOST IS FRIGHTENED

**if** self.blue\_ghost.get\_is\_alive():

# IF THE BLUE GHOST IS ALIVE

**if** self.pacman.get\_score() >= **200**:

# IF THE PLAYER HAS GAINED MORE THAT 200 POINTS

**if** (**12** \* fps) >= self.pacman.get\_timer():

# IF 12 SECONDS HAS NOT PASSED

self.blue\_ghost.search((**0**, self.blue\_ghost.get\_pixel\_position(), self.blue\_ghost.get\_pixel\_position()), (**476**, **560**), self.maze\_class.walls)

self.blue\_ghost.update\_direction(self.blue\_ghost.direction\_calculator(self.blue\_ghost.visited, self.blue\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# BLUE GHOST WILL RUN TO ITS CORNER

**if** self.pacman.get\_timer() >= (**10** \* fps):

# IF THE GHOST IS IN ITS REMAINING 2 SECONDS OF BEING FRIGHTENED

self.blue\_ghost.update\_sprite(self.blue\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# CHANGE THE IMAGE TO MAKE IT FLICKER

**else**:

self.blue\_ghost.update\_sprite(self.blue\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

# OR JUST KEEP THE IMAGE AS A FRIGHTENED GHOST

**else**:

self.blue\_ghost.update\_is\_frightened(False)

self.pacman.update\_score(self.pacman.get\_score() + **10**)

# ONCE 12 SECONDS HAS PASSED RETURN GHOST TO NORMAL STATE AND GIVE THE PLAYER 10 POINTS

**else**:

self.blue\_ghost.update\_sprite(self.blue\_ghost.right\_images, **0.15**, **4**, **4**)

# IF THE GHOST HAS NOT BEEN ACTIVATED JUST GIVE A LOOKING RIGHT ANIMATION

**else**:

self.blue\_ghost.is\_running(self.pacman.get\_pixel\_position(), self.maze\_class.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

**if** **not** self.blue\_ghost.get\_is\_frightened():

# IF THE BLUE GHOST IS NOT FRIGHTENED

**if** (self.pacman.get\_score() >= **200**) **or** self.blue\_ghost.get\_ghost\_clock() >= (**25** \* fps):

# IF THE PLAYER HAS 200 OR MORE POINTS OR 25 SECONDS HAS PASSED

**if** len(self.pacman.maze\_class.special\_dots\_list) >= **1**:

# IF THERE ARE STILL SPECIAL DOTS LEFT IN THE GAME

**if** self.blue\_ghost.closest\_distance\_calculator(self.pacman.get\_pixel\_position(), self.maze\_class.special\_dots\_list) == self.blue\_ghost.get\_start\_pos():

# IF THE DOT THE GHOST IS AIMING FOR IS THE SAME DOT AS BEFORE

**if** self.blue\_ghost.closest\_distance\_calculator(self.pacman.get\_pixel\_position(), self.maze\_class.special\_dots\_list) == self.blue\_ghost.get\_end\_pos():

# IF THE GHOST HAS ALREADY REACHED THIS DOT

self.blue\_ghost.update\_direction(self.blue\_ghost.random\_movement(self.blue\_ghost.get\_direction(), self.maze\_class))

# MOVE RANDOMLY

**else**:

self.blue\_ghost.search((**0**, self.blue\_ghost.get\_pixel\_position(), self.blue\_ghost.get\_pixel\_position()), self.blue\_ghost.closest\_distance\_calculator(self.pacman.get\_pixel\_position(), self.maze\_class.special\_dots\_list), self.maze\_class.walls)

self.blue\_ghost.update\_direction(self.blue\_ghost.direction\_calculator(self.blue\_ghost.visited, self.blue\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# OTHERWISE MOVE TO THE SPECIAL DOT THAT IS CLOSEST TO THE PLAYER

**else**:

self.blue\_ghost.update\_end\_pos(None)

# OTHERWISE RESET THE END POSITION

self.blue\_ghost.search((**0**, self.blue\_ghost.get\_pixel\_position(), self.blue\_ghost.get\_pixel\_position()), self.blue\_ghost.closest\_distance\_calculator(self.pacman.get\_pixel\_position(), self.maze\_class.special\_dots\_list), self.maze\_class.walls)

self.blue\_ghost.update\_direction(self.blue\_ghost.direction\_calculator(self.blue\_ghost.visited, self.blue\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# MOVE TO THE SPECIAL DOT THAT IS CLOSEST TO THE PLAYER

self.blue\_ghost.update\_start\_pos(self.blue\_ghost.closest\_distance\_calculator(self.pacman.get\_pixel\_position(), self.maze\_class.special\_dots\_list))

# CALCULATE THE SPECIAL DOT THAT IS CLOSEST TO THE PLAYER AND ASSIGN START POS THAT VALUE

**else**:

self.blue\_ghost.update\_direction(self.blue\_ghost.random\_movement(self.blue\_ghost.get\_direction(), self.maze\_class))

# IF THERE ARE NO MORE SPECIAL DOTS LEFT JUST MOVE RANDOMLY

# BLACK GHOST MOVEMENT

**if** self.black\_ghost.get\_is\_frightened():

# IF THE GHOST IS FRIGHTENED

**if** self.black\_ghost.get\_is\_alive():

# IF THE BLACK GHOST IS ALIVE

**if** self.pacman.get\_level() >= **3**:

# IF WE'RE PAST LEVEL 3

**if** (**12** \* fps) >= self.pacman.get\_timer():

# IF WE'RE WITHIN 12 SECONDS

self.black\_ghost.search((**0**, self.black\_ghost.get\_pixel\_position(), self.black\_ghost.get\_pixel\_position()), (**28**, **560**), self.maze\_class.walls)

self.black\_ghost.update\_direction(self.black\_ghost.direction\_calculator(self.black\_ghost.visited, self.black\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# CALCULATE THE SHORTEST DISTANCE TO THE CORNER AND MOVE THERE

**if** self.pacman.get\_timer() >= (**10** \* fps):

# IF THE GHOST IS IN ITS REMAINING 2 SECONDS OF BEING FRIGHTENED

self.black\_ghost.update\_sprite(self.black\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# CHANGE THE IMAGE AND MAKE IT FLICKER

**else**:

# OTHERWISE KEEP IT THE SAME

self.black\_ghost.update\_sprite(self.black\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

**else**:

# IF ITS PAST 120 SECONDS

self.black\_ghost.update\_is\_frightened(False)

self.pacman.update\_score(self.pacman.get\_score() + **10**)

# STOP THE GHOST FROM BEING FRIGHTENED AND GIVE THE PLAYER 10 POINTS

**else**:

self.black\_ghost.update\_sprite(self.black\_ghost.right\_images, **0.15**, **4**, **4**)

# IF THE GHOST IS NOT ACTIVE USE LOOKING RIGHT ANIMATION

**else**:

self.black\_ghost.is\_running(self.pacman.get\_pixel\_position(), self.maze\_class.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

**if** **not** self.black\_ghost.get\_is\_frightened():

# IF THE BLACK GHOST IS NOT FRIGHTENED

**if** self.pacman.get\_level() >= **3**:

# IF WE ARE ON OR PAST LEVEL THREE

**if** self.black\_ghost.get\_change\_mode():

# IF THE BLACK GHOST HAS CHANGED MODE

**if** self.black\_ghost.get\_ghost\_clock() >= (**25** \* fps):

# IF 25 SECONDS HAS PASSED (15 SECONDS OF CHASING THE PLAYER)

self.black\_ghost.update\_change\_mode(False)

self.black\_ghost.update\_ghost\_clock(**0**)

# CHANGE THE GHOST BACK AND RESET THE CLOCK

self.black\_ghost.update\_direction((**0**, **0**))

self.black\_ghost.rect.center = (self.black\_ghost.get\_pixel\_position()[**0**] + (cell\_width // **2**), self.black\_ghost.get\_pixel\_position()[**1**] + (cell\_height // **2**))

**else**:

self.black\_ghost.search((**0**, self.black\_ghost.get\_pixel\_position(), self.black\_ghost.get\_pixel\_position()), self.pacman.get\_pixel\_position(), self.maze\_class.walls)

self.black\_ghost.update\_direction(self.black\_ghost.direction\_calculator(self.black\_ghost.visited, self.black\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# OTHERWISE MAKE THE GHOST CHASE THE PLAYER AND CHANGE THE IMAGE

**else**:

**if** self.black\_ghost.get\_ghost\_clock() >= (**10** \* fps):

# IF 10 SECONDS HAS PASSED

self.black\_ghost.update\_change\_mode(True)

# CHANGE MODE

self.black\_ghost.update\_direction((**0**, **0**))

self.black\_ghost.rect.center = (self.black\_ghost.get\_pixel\_position()[**0**] + (cell\_width // **2**), self.black\_ghost.get\_pixel\_position()[**1**] + (cell\_height // **2**))

**else**:

self.black\_ghost.update\_direction(self.black\_ghost.random\_movement(self.black\_ghost.get\_direction(), self.maze\_class))

# OTHERWISE THE GHOST MOVES RANDOMLY

# ORANGE GHOST MOVEMENT

**if** self.orange\_ghost.get\_is\_frightened():

# IF THE ORANGE GHOST IS FRIGHTENED

**if** self.orange\_ghost.get\_is\_alive():

# IF THE GHOST IS ALIVE

**if** self.pacman.get\_level() >= **4**:

# IF WE HAVE PASSED LEVEL 4

**if** (**12** \* fps) >= self.pacman.get\_timer():

# IF 12 SECONDS HAS NOT PASSED

self.orange\_ghost.search((**0**, self.orange\_ghost.get\_pixel\_position(), self.orange\_ghost.get\_pixel\_position()), (**476**, **28**), self.maze\_class.walls)

self.orange\_ghost.update\_direction(self.orange\_ghost.direction\_calculator(self.orange\_ghost.visited, self.orange\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# RUN TO GHOST CORNER

**if** self.pacman.get\_timer() >= (**10** \* fps):

# IF THE GHOST HAS 2 SECONDS REMAINING OF BEING FRIGHTENED

self.orange\_ghost.update\_sprite(self.orange\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# UPDATE IMAGE TO FLICKER

**else**:

self.orange\_ghost.update\_sprite(self.orange\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

# OTHERWISE KEEP AS A FRIGHTENED GHOST

**else**:

# 12 SECONDS HAS PASSED

self.orange\_ghost.update\_is\_frightened(False)

self.pacman.update\_score(self.pacman.get\_score() + **10**)

# GHOST IS NO LONGER FRIGHTENED AND GIVE PLAYER 10 POINTS

**else**:

self.orange\_ghost.update\_sprite(self.orange\_ghost.right\_images, **0.15**, **4**, **4**)

# IF THE GHOST HAS NOT BEEN ACTIVATED GIVE IT THE MOVING RIGHT ANIMATION

**else**:

self.orange\_ghost.is\_running(self.pacman.get\_pixel\_position(), self.maze\_class.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

**if** **not** self.orange\_ghost.get\_is\_frightened():

# IF THE ORANGE GHOST IS NOT FRIGHTENED

**if** self.pacman.get\_level() >= **4**:

# IF WE ARE PAST LEVEL 3

**if** self.orange\_ghost.get\_ghost\_clock() > (**0** \* fps):

# IF 10 SECONDS HAS PASSED

self.orange\_ghost.search((**0**, self.orange\_ghost.get\_pixel\_position(), self.orange\_ghost.get\_pixel\_position()), self.orange\_ghost.orange\_ghost\_movement(self.pacman.get\_pixel\_position(), self.maze\_class.maze\_pixel\_list, self.pacman), self.maze\_class.walls)

self.orange\_ghost.update\_direction(self.orange\_ghost.direction\_calculator(self.orange\_ghost.visited, self.orange\_ghost.get\_direction(), self.pacman.get\_pixel\_position(), self.maze\_class))

# MAKE THE GHOST CHASE TWO POSITIONS AHEAD OF PACMAN

self.pacman.update\_timer(self.pacman.get\_timer() + **1**)

# ADD ONE TO PLAYER TIMER

**else**:

# IF THE PLAYER DIES

**if** self.pacman.get\_lives() >= **1**:

# IF THE PLAYER STILL HAS LIVES REMAINING

self.reset()

# MOVE ALL ENTITIES TO THEIR STARTING POS AND CHANGE DIRECTION TO NEUTRAL

**if** self.pacman.get\_timer() <= **420**:

# IF THE ANIMATION HAS NOT RUN LONG ENOUGH RUN AGAIN

self.pacman.update\_sprite(self.pacman.death\_images, **0.06**, **5**, **5**)

# MAKE IMAGE LOOK ANIMATED

**else**:

# PLAYER HAS RUN OUT OF LIVES AND THE GAME IS OVER

self.red\_ghost.update\_direction((**0**, **0**))

self.blue\_ghost.update\_direction((**0**, **0**))

self.black\_ghost.update\_direction((**0**, **0**))

self.orange\_ghost.update\_direction((**0**, **0**))

self.pacman.update\_update\_direction((**0**, **0**))

self.pacman.update\_direction((**0**, **0**))

# FREEZE ALL ENTITIES

database = sqlite3.connect("database/players.db")

# CONNECT TO DATABASE

cursor = database.cursor()

# CREATE CURSOR

**try**:

get\_score = 'SELECT highscore FROM user WHERE username = ?'

# SQL QUERY TO GET PLAYERS SCORE

cursor.execute(get\_score, [self.player\_class.login.username])

# GETS PLAYERS SCORE

**if** cursor.fetchall()[**0**][**0**] < self.pacman.get\_score():

# IF THE PLAYERS GAME SCORE IS GREATER THAN THE ONE ON THE DATABASE

update\_data = 'UPDATE user SET highscore = ? WHERE username = ?;'

# SQL QUERY TO UPDATE THEIR SCORE

cursor.execute(update\_data, [self.pacman.get\_score(), self.player\_class.login.username])

# EXECUTE SQL QUERY AND UPDATE SCORE

**except**:

**print**("UNABLE TO SAVE DATA AN UNNEXPECTED ERROR OCCURED")

database.commit()

# CLOSE DATABASE CONNECTION

cursor.close()

# CLOSE CURSOR

self.player\_class.state = "gameover\_singleplayer"

# DISPLAY IMAGE ON SCREEN

**else**:

# IF THE PLAYER COMPLETES THE LEVEL

self.reset()

self.maze\_class.reset()

# RESET ALL ENTITIES AND THE MAZE

**else**:

mixer.music.stop()

self.play\_music\_once = True

self.red\_ghost.force\_pause()

self.blue\_ghost.force\_pause()

self.blue\_ghost.force\_pause()

self.orange\_ghost.force\_pause()

self.pacman.force\_pause()

self.all\_sprites.update()

# UPDATE ALL SPRITES

# DRAWING SCREEN

**def** **draw\_screen**(self, screen, game\_background, game\_over\_image):

##### draw\_screen #######

# Parameters : screen:Canvas, game\_background:Image, game\_over\_image, Image

# Return Type : None

# Purpose :- Controls entire screen for singleplayer game

##########################

screen.fill(BLACK)

screen.blit(game\_background, (**11**, **0**))

# DRAW GAME MAZE ON SCREEN

**for** each\_coin **in** self.maze\_class.dots\_list:

screen.blit(self.player\_class.dot\_image, (each\_coin[**0**] + **10**, each\_coin[**1**] + **10**))

# DRAW ALL THE DOTS ON THE SCREEN

**for** each\_special\_coin **in** self.maze\_class.special\_dots\_list:

screen.blit(self.player\_class.special\_dot\_image, (each\_special\_coin[**0**] + **2**, each\_special\_coin[**1**] + **2**))

# DRAW THE SPECIAL DOTS ON SCREEN

**for** each\_fruit **in** self.maze\_class.fruit\_positions:

fruit\_im = pygame.transform.scale(self.player\_class.fruit\_images[self.maze\_class.fruit\_images[self.maze\_class.fruit\_positions.index(each\_fruit)]], (**15**, **15**))

screen.blit(fruit\_im, (each\_fruit[**0**] + **4**, each\_fruit[**1**] + **5**))

# DRAW FRUIT ON SCREEN

x\_value = **460**

**for** each\_image **in** self.maze\_class.fruit\_images:

image = pygame.transform.scale(self.player\_class.fruit\_images[each\_image], (**10**, **10**))

screen.blit(image, (x\_value, **604**))

x\_value += **15**

# DRAW THE ACCORDING FRUIT

self.all\_sprites.draw(screen)

# DRAW ALL THE PLAYERS AND GHOSTS

**if** **not** self.pacman.get\_has\_started() **and** self.pacman.get\_lives() >= **1**:

# IF THE GAME HAST STARTED, THE PLAYER IS NOT WAITING AND THEY STILL HAVE LIVES REMAINING

self.pacman.update\_timer(self.pacman.get\_timer() + **1**)

# INCREASE THEIR TIME

**if** self.pacman.get\_timer() < **120**:

self.player\_class.text(menu\_font, **20**, WHITE, "3", BLACK, screen, **265**, **270**)

**if** **120** <= self.pacman.get\_timer() < **240**:

self.player\_class.text(menu\_font, **20**, WHITE, "2", BLACK, screen, **265**, **270**)

**if** **240** <= self.pacman.get\_timer() < **360**:

self.player\_class.text(menu\_font, **20**, WHITE, "1", BLACK, screen, **265**, **270**)

**if** **360** <= self.pacman.get\_timer() < **420**:

self.player\_class.text(menu\_font, **20**, WHITE, "GO!", BLACK, screen, **265**, **270**)

**if** self.pacman.get\_timer() == **420**:

# TIME TO START

**if** **not** self.pacman.get\_is\_alive():

# IF THE PLAYER HAD DIED

self.pacman.reset((starting\_position\_x, starting\_position\_y))

self.pacman.update\_lives(self.pacman.get\_lives() - **1**)

self.play\_music\_once = True

# PLAYER LOSES A LIFE RESET PLAYER

**if** self.pacman.get\_level\_over():

# IF THE PLAYER COMPLETED THE LEVEL

self.pacman.reset((starting\_position\_x, starting\_position\_y))

self.maze\_class.random\_fruit(self.maze\_class.dots\_list, self.maze\_class.fruit\_positions, self.maze\_class.fruit\_images)

self.play\_music\_once = True

# RESET POSITIONS AND MAZE

self.pacman.update\_timer(**0**)

self.pacman.update\_has\_started(True)

# UPDATE VARIABLES

self.player\_class.text(arialbold\_font, **14**, WHITE, "LIVES: ", BLACK, screen, **40**, **10**)

# LIVES TEXT HEADING ON SCREEN

self.player\_class.text(arialbold\_font, **14**, WHITE, "SCORE: {}".format(self.pacman.get\_score()), BLACK, screen, (screen\_width-width\_buffer)-len("SCORE: {}".format(self.pacman.get\_score()))-**20**, **10**)

# SCORE TEXT ON SCREEN AS WELL AS SCORE

self.player\_class.draw\_lives(**65**, **20**, screen, self.pacman.get\_lives(), self.pacman.heart\_image)

# DRAW LIVES AS PACMAN IMAGES

self.player\_class.text(arialbold\_font, **14**, WHITE, "LEVEL: {} ".format(self.pacman.get\_level()), BLACK, screen, screen\_width//**2**, **10**)

# LEVEL TEXT ON SCREEN AS WELL AS LEVEL

**if** self.player\_class.state == "gameover\_singleplayer":

# IF THE GAME IS OVER

mixer.music.stop()

# STOP MUSIC

self.play\_music\_once = True

screen.blit(self.player\_class.grey\_screen, (**0**, **0**))

# MAKE THE SCREEN GRAY

screen.blit(game\_over\_image, (**92**, **100**))

# DRAW GAME OVER IMAGE OVER THE GAME

self.player\_class.button(**60**, **131**, **400**, **35**, WHITE, GRAY, screen, menu\_font, **20**, BLACK, "MENU SCREEN", "menu")

# MENU BUTTON

self.player\_class.button(**343**, **131**, **400**, **35**, WHITE, GRAY, screen, menu\_font, **20**, BLACK, "PLAY AGAIN", "singleplayer")

# PLAY AGAIN BUTTON

**if** self.player\_class.state != "gameover\_singleplayer":

self.reset\_game()

# RESET ALL ENTITIES AND MAZE

**if** self.pacman.get\_is\_paused():

# IF THE PLAYER PAUSED THE GAME

screen.blit(self.player\_class.grey\_screen, (**0**, **0**))

self.player\_class.button(**202**, **131**, **300**, **35**, LIGHT\_MAZE\_BLUE, DARK\_MAZE\_BLUE, screen, menu\_font, **20**, BLACK, "MENU SCREEN", "menu")

self.player\_class.button(**202**, **131**, **368**, **35**, LIGHT\_MAZE\_BLUE, DARK\_MAZE\_BLUE, screen, menu\_font, **20**, BLACK, "RESUME GAME", "resume")

self.player\_class.button(**202**, **131**, **436**, **35**, LIGHT\_MAZE\_BLUE, DARK\_MAZE\_BLUE, screen, menu\_font, **20**, BLACK, "QUIT", "quit")

# DRAW ON SCREEN

key\_pressed = pygame.key.get\_pressed()

# GET KEY PRESSED

**if** key\_pressed[pygame.K\_r] **or** self.player\_class.state == "resume":

# IF THE PLAYER PRESSES RESUME OR R

self.player\_class.state = "singleplayer"

self.pacman.update\_is\_paused(False)

self.pacman.update\_has\_started(False)

# RESUME GAME

**if** self.player\_class.state == "menu":

# IF THE PLAYER WANTS TO GO TO THE MENU

pygame.mixer.stop()

self.reset\_game()

# STOP MUSIC AND RESET GAME

pygame.display.flip()

# UPDATE THE SCREEN

**def** **reset**(self):

##### reset #######

# Parameters : None

# Return Type : None

# Purpose :- Resets all entities to their starting positions

##########################

# RESET FUNCTION

mixer.music.stop()

self.red\_ghost.reset(red\_ghost\_starting\_position)

self.blue\_ghost.reset(blue\_ghost\_starting\_position)

self.black\_ghost.reset(black\_ghost\_starting\_position)

self.orange\_ghost.reset(orange\_ghost\_starting\_position)

self.pacman.update\_direction((**0**, **0**))

self.pacman.update\_update\_direction((**0**, **0**))

self.pacman.rect.center = ((starting\_position\_x, starting\_position\_y))

self.pacman.update\_pixel\_position(vector((self.pacman.rect.x // cell\_width) \* cell\_width, (self.pacman.rect.y // cell\_width) \* cell\_width))

# RESETS ALL ENTITIES TO STARTING POSITIONS AND MAKES THEM STATIONARY

**def** **reset\_game**(self):

##### reset\_game #######

# Parameters : None

# Return Type : None

# Purpose :- Resets entire game back to the start

##########################

# RESET GAME FUNCTION

self.pacman.reset\_game((starting\_position\_x, starting\_position\_y))

self.red\_ghost.reset\_game(red\_ghost\_starting\_position)

self.blue\_ghost.reset\_game(blue\_ghost\_starting\_position)

self.black\_ghost.reset\_game(black\_ghost\_starting\_position)

self.orange\_ghost.reset\_game(orange\_ghost\_starting\_position)

self.maze\_class.reset()

# RESETS ALL ENTITIES AND MAZES TO THEIR ORIGINAL VALUE

**multiplayer.py**

**from** **pacman** **import** \*

**from** **ghosts** **import** \*

**from** **maze** **import** \*

**from** **client\_network** **import** \*

**import** **json**

vector = pygame.math.Vector2

**class** **multiplayer**:

**def** **\_\_init\_\_**(self, player\_class):

self.maze = Maze()

self.all\_sprites = pygame.sprite.Group()

self.player\_class = player\_class

self.client\_variable = client()

player\_one = json.loads(self.client\_variable.return\_data())

self.player\_one = Pacman((player\_one[**0**][**0**], player\_one[**0**][**1**]), self.maze, "multiplayer", self)

self.player\_one.update\_update\_direction((player\_one[**0**][**2**], player\_one[**0**][**3**]))

self.player\_one.score = player\_one[**0**][**4**]

self.player = player\_one[**0**][**5**]

self.time = player\_one[**1**]

self.is\_paused\_time = **30** \* **120**

self.is\_waiting = player\_one[**2**]

self.player\_two = Pacman((**294**, **350**), self.maze, "multiplayer", self)

self.red\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "redghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "redghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "redghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "redghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "redghost\_down\_2.png"))], red\_ghost\_starting\_position, self.player\_one, "red", "multiplayer")

self.blue\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "blueghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blueghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blueghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blueghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blueghost\_down\_2.png"))], blue\_ghost\_starting\_position, self.player\_one, "blue", "multiplayer")

self.orange\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "orangeghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "orangeghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "orangeghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "orangeghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "orangeghost\_down\_2.png"))], orange\_ghost\_starting\_position, self.player\_one, "orange", "multiplayer")

self.black\_ghost = Ghost([pygame.image.load(os.path.join(ghost\_images, "blackghost\_right\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_right\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blackghost\_left\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_left\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blackghost\_up\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_up\_2.png"))], [pygame.image.load(os.path.join(ghost\_images, "blackghost\_down\_1.png")), pygame.image.load(os.path.join(ghost\_images, "blackghost\_down\_2.png"))], black\_ghost\_starting\_position, self.player\_one, "black", "multiplayer")

# CREATING NEW INSTANCES OF GHOSTS PACMAN AND MAZE

self.all\_sprites.add(self.red\_ghost)

self.all\_sprites.add(self.blue\_ghost)

self.all\_sprites.add(self.orange\_ghost)

self.all\_sprites.add(self.black\_ghost)

self.all\_sprites.add(self.player\_one)

self.all\_sprites.add(self.player\_two)

# ADDS ALL SPRITES TO SPRITES LIST

**def** **event\_manager**(self):

##### event\_manager #######

# Parameters : None

# Return Type : None

# Purpose :- Controls entire game events

##########################

**if** self.is\_waiting:

waiting = json.loads(self.client\_variable.send\_data(json.dumps("wait")))

self.is\_waiting = waiting

**if** **not** self.is\_waiting:

player\_two = json.loads(self.client\_variable.send\_data(json.dumps((self.player\_one.rect.center[**0**], self.player\_one.rect.center[**1**], int(self.player\_one.get\_direction()[**0**]), int(self.player\_one.get\_direction()[**1**]), int(self.player\_one.get\_score()), self.player\_one.get\_is\_paused()))))

# SENDS OWN DATA TO SERVER AND GETS OTHER PLAYERS DATA AS WELL AS SOME GLOBAL VARIABLES

self.player\_two.rect.center = (player\_two[**0**][**0**], player\_two[**0**][**1**])

self.player\_two.update\_direction((player\_two[**0**][**2**], player\_two[**0**][**3**]))

self.player\_two.update\_score(player\_two[**0**][**4**])

self.player\_two.update\_is\_paused(player\_two[**0**][**6**])

self.time = player\_two[**2**]

self.player\_one.update\_level(player\_two[**6**])

self.player\_one.update\_is\_alive(player\_two[**4**])

self.player\_two.update\_is\_alive(player\_two[**4**])

self.player\_one.update\_lives(player\_two[**3**])

self.player\_one.update\_has\_started(player\_two[**7**])

# ASSIGN ALL THE VARIABLES

# GAME

**if** **not** self.player\_one.get\_is\_paused() **and** **not** self.player\_two.get\_is\_paused():

**if** **not** player\_two[**5**]:

# IF THE LEVEL IS NOT OVER

**if** player\_two[**4**]:

# IF BOTH PLAYERS ARE STILL ALIVE

**if** player\_two[**7**]:

self.player\_class.movement(self.player\_one)

# MAKE THE PLAYER MOVE

# RED GHOST

**if** player\_two[**1**][**0**][**6**]:

# IF THE RED GHOST IS FRIGHTENED

**if** self.red\_ghost.get\_pixel\_position() == self.player\_one.get\_pixel\_position() **or** self.red\_ghost.get\_pixel\_position() == self.player\_two.get\_pixel\_position():

# IF ANY OF THE PLAYERS COLLIDE DURING THE GHOST BEING FRIGHTENED

self.red\_ghost.update\_is\_alive(False)

# RED GHOST IS NO LONGER ALIVE

**else**:

# IF THE PLAYER HAS NOT COLLIDED WITH THE GHOST

**if** player\_two[**1**][**0**][**7**]:

# IF THE RED GHOST IS ALIVE

**if** (**12** \* fps) >= self.time:

# IF IT HAS NOT BEEN 12 SECONDS

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.player\_zero\_function(player\_two, (**28**, **28**), self.red\_ghost, **0**)

# CALCULATE THE SHORTEST PATH TO CORNER AND MOVE THERE

**if** self.time >= (**10** \* fps):

# IF THE GHOST HAS 2 SECONDS REMAINING OF BEING FRIGHTENED

self.red\_ghost.update\_sprite(self.red\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# UPDATE IMAGE TO FLICKER

**else**:

self.red\_ghost.update\_sprite(self.red\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

# OTHERWISE KEEP IMAGE AS A FRIGHTENED GHOST

**else**:

# IF ITS PAST 12 SECONDS

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.red\_ghost.update\_is\_frightened(False)

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GHOST IS NO LONGER FRIGHTENED AND GIVE PLAYER 10 POINTS

**if** self.player == **1**:

# IF IT IS PLAYER 1

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GIVE PLAYER 10 POINTS

**else**:

self.red\_ghost.is\_running(None, self.maze.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

# BLUE GHOST

**if** player\_two[**1**][**1**][**6**]:

# IF THE BLUE GHOST IS FRIGHTENED

**if** self.blue\_ghost.get\_pixel\_position() == self.player\_one.get\_pixel\_position() **or** self.blue\_ghost.get\_pixel\_position() == self.player\_two.get\_pixel\_position():

# IF ANY OF THE PLAYERS COLLIDE DURING THE GHOST BEING FRIGHTENED

self.blue\_ghost.update\_is\_alive(False)

# GHOST IS NO LONGER ALIVE

**else**:

# IF THE PLAYER HAS NOT COLLIDED WITH THE GHOST

**if** player\_two[**1**][**1**][**7**]:

# IF THE BLUE GHOST IS ALIVE

**if** (self.player\_one.get\_score() + self.player\_two.get\_score()) >= **200**:

# IF THE PLAYERS SCORES ARE GREATER THAN OR EQUAL TO 200 COLLECTIVELY

**if** (**12** \* fps) >= self.time:

# IF IT HAS NOT BEEN 12 SECONDS

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.player\_zero\_function(player\_two, (**476**, **560**), self.blue\_ghost, **1**)

# CALCULATE THE SHORTEST PATH TO CORNER AND MOVE THERE

**if** self.time >= (**10** \* fps):

# IF THE GHOST HAS 2 SECONDS REMAINING OF BEING FRIGHTENED

self.blue\_ghost.update\_sprite(self.blue\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# UPDATE IMAGE TO FLICKER

**else**:

self.blue\_ghost.update\_sprite(self.blue\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

# OTHERWISE KEEP IMAGE AS A FRIGHTENED GHOST

**else**:

# IF ITS PAST 12 SECONDS

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.blue\_ghost.update\_is\_frightened(False)

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GHOST IS NO LONGER FRIGHTENED AND GIVE PLAYER 10 POINTS

**if** self.player == **1**:

# IF IT IS PLAYER 1

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GIVE PLAYER 10 POINTS

**else**:

self.blue\_ghost.update\_sprite(self.blue\_ghost.right\_images, **0.15**, **4**, **4**)

# IF THE GHOST IS NOT ACTIVE GIVE IT THE MOVING RIGHT ANIMATION

**else**:

**if** self.player == **0**:

# IF IT IS PLAYER 0

closest\_player\_blue\_ghost = self.blue\_ghost.closest\_distance\_calculator((player\_two[**1**][**1**][**0**], player\_two[**1**][**1**][**1**]),[self.player\_one.get\_pixel\_position(), self.player\_two.get\_pixel\_position()])

self.blue\_ghost.is\_running(closest\_player\_blue\_ghost, self.maze.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

# BLACK GHOST

**if** player\_two[**1**][**2**][**6**]:

# IF THE BLACK GHOST IS FRIGHTENED

**if** self.black\_ghost.get\_pixel\_position() == self.player\_one.get\_pixel\_position() **or** self.black\_ghost.get\_pixel\_position() == self.player\_two.get\_pixel\_position():

# IF ANY OF THE PLAYERS COLLIDE DURING THE GHOST BEING FRIGHTENED

self.black\_ghost.update\_is\_alive(False)

# GHOST IS NO LONGER ALIVE

**else**:

# IF THE PLAYER HAS NOT COLLIDED WITH THE GHOST

**if** player\_two[**1**][**2**][**7**]:

# IF THE BLACK GHOST IS ALIVE

**if** self.player\_one.get\_level() >= **3**:

# IF WE'RE PAST LEVEL 3

**if** (**12** \* fps) >= self.time:

# IF WE'RE WITHIN 12 SECONDS

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.player\_zero\_function(player\_two, (**28**, **560**), self.black\_ghost, **2**)

# CALCULATE THE SHORTEST PATH TO CORNER AND MOVE THERE

**if** self.time >= (**10** \* fps):

# IF THE GHOST HAS 2 SECONDS REMAINING OF BEING FRIGHTENED

self.black\_ghost.update\_sprite(self.black\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# UPDATE IMAGE TO FLICKER

**else**:

self.black\_ghost.update\_sprite(self.black\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

# OTHERWISE KEEP IMAGE AS A FRIGHTENED GHOST

**else**:

# IF ITS PAST 12 SECONDS

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.black\_ghost.update\_is\_frightened(False)

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GHOST IS NO LONGER FRIGHTENED AND GIVE PLAYER 10 POINTS

**if** self.player == **1**:

# IF IT IS PLAYER 1

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GIVE PLAYER 10 POINTS

**else**:

self.black\_ghost.update\_sprite(self.black\_ghost.right\_images, **0.15**, **4**, **4**)

# IF THE GHOST IS NOT ACTIVE GIVE IT THE MOVING RIGHT ANIMATION

**else**:

self.black\_ghost.is\_running(None, self.maze.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

# ORANGE GHOST

**if** player\_two[**1**][**3**][**6**]:

# IF THE ORANGE GHOST IS FRIGHTENED

**if** self.orange\_ghost.get\_pixel\_position() == self.player\_one.get\_pixel\_position() **or** self.orange\_ghost.get\_pixel\_position() == self.player\_two.get\_pixel\_position():

# IF ANY OF THE PLAYERS COLLIDE DURING THE GHOST BEING FRIGHTENED

self.orange\_ghost.update\_is\_alive(False)

# GHOST IS NO LONGER ALIVE

**else**:

# IF THE PLAYER HAS NOT COLLIDED WITH THE GHOST

**if** player\_two[**1**][**3**][**7**]:

# IF THE ORANGE GHOST IS ALIVE

**if** self.player\_one.get\_level() >= **4**:

# IF WE HAVE PASSED LEVEL FIVE

**if** (**12** \* fps) >= self.time:

# IF IT HAS NOT BEEN 12 SECONDS

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.player\_zero\_function(player\_two, (**476**, **28**), self.orange\_ghost, **3**)

# CALCULATE THE SHORTEST PATH TO CORNER AND MOVE THERE

**if** self.time >= (**10** \* fps):

# IF THE GHOST HAS 2 SECONDS REMAINING OF BEING FRIGHTENED

self.orange\_ghost.update\_sprite(self.orange\_ghost.flicker\_mode\_images, **0.15**, **4**, **4**)

# UPDATE IMAGE TO FLICKER

**else**:

self.orange\_ghost.update\_sprite(self.orange\_ghost.frightened\_mode\_images, **0.15**, **4**, **4**)

# OTHERWISE KEEP IMAGE AS A FRIGHTENED GHOST

**else**:

# IF 12 SECONDS HAS PASSED

**if** self.player == **0**:

# IF IT IS PLAYER 0

self.orange\_ghost.update\_is\_frightened(False)

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GHOST IS NO LONGER FRIGHTENED AND GIVE PLAYER 10 POINTS

**if** self.player == **1**:

# IF IT IS PLAYER 1

self.player\_one.update\_score(self.player\_one.get\_score() + **10**)

# GIVE PLAYER 10 POINTS

**else**:

self.orange\_ghost.update\_sprite(self.orange\_ghost.right\_images, **0.15**, **4**, **4**)

# IF THE GHOST IS NOT ACTIVE GIVE IT THE MOVING RIGHT ANIMATION

**else**:

self.orange\_ghost.is\_running(None, self.maze.walls)

# WHEN THE PLAYER HAS EATEN THE GHOST MAKE IT RUN TO THE GHOST CHAMBER

# CHECK TO UPDATE MAZE AND SEE IF ANY PLAYER HAS EATEN THE SPECIAL DOT

self.dot\_check(self.player\_one)

# CHECK IF PLAYER ONE IS ON AND DOTS OR SPECIAL DOTS

self.dot\_check(self.player\_two)

# CHECK IF PLAYER TWO IS ON AND DOTS OR SPECIAL DOTS

**if** self.player == **0**:

# IF IT IS PLAYER 0

closest\_player\_red\_ghost = self.red\_ghost.closest\_distance\_calculator((player\_two[**1**][**0**][**0**], player\_two[**1**][**0**][**1**]), [self.player\_one.get\_pixel\_position(), self.player\_two.get\_pixel\_position()])

closest\_player\_blue\_ghost = self.blue\_ghost.closest\_distance\_calculator((player\_two[**1**][**1**][**0**], player\_two[**1**][**1**][**1**]), [self.player\_one.get\_pixel\_position(), self.player\_two.get\_pixel\_position()])

closest\_player\_black\_ghost = self.black\_ghost.closest\_distance\_calculator((player\_two[**1**][**2**][**0**], player\_two[**1**][**2**][**1**]), [self.player\_one.get\_pixel\_position(), self.player\_two.get\_pixel\_position()])

closest\_player\_orange\_ghost = self.orange\_ghost.closest\_distance\_calculator((player\_two[**1**][**3**][**0**], player\_two[**1**][**3**][**1**]), [self.player\_one.get\_pixel\_position(), self.player\_two.get\_pixel\_position()])

# CALCULATE THE CLOSEST PLAYER TO EACH INDIVIDUAL GHOST

**if** closest\_player\_orange\_ghost == self.player\_one.get\_pixel\_position():

# IF THE CLOSEST PLAYER TO THE ORANGE GHOST IS PLAYER ONE

orange\_closest\_player = self.player\_one

# ASSIGN ORANGE CLOSEST PLAYER AS PLAYER ONE

**if** closest\_player\_orange\_ghost == self.player\_two.get\_pixel\_position():

# IF THE CLOSEST PLAYER TO THE ORANGE GHOST IS PLAYER TWO

orange\_closest\_player = self.player\_two

# ASSIGN ORANGE CLOSEST PLAYER AS PLAYER TWO

# RED GHOST MOVEMENT

**if** **not** player\_two[**1**][**0**][**6**]:

# IF THE RED GHOST IS NOT FRIGHTENED

self.pacman\_dead\_check(self.red\_ghost)

# CHECK TO SEE IF A PLAYER HAS COLLIDED WITH GHOST

self.player\_zero\_function(player\_two, closest\_player\_red\_ghost, self.red\_ghost, **0**)

# FIND SHORTEST PATH TO CLOSEST PLAYER AND MOVE THERE

# BLUE GHOST MOVEMENT

**if** **not** player\_two[**1**][**1**][**6**]:

# IF THE BLUE GHOST IS NOT FRIGHTENED

**if** (self.player\_one.get\_score() + self.player\_two.get\_score()) >= **200**:

# IF THE SCORE IS GREATER THAN OR EQUAL TO 200 COLLECTIVELY

self.pacman\_dead\_check(self.blue\_ghost)

# CHECK TO SEE IF A PLAYER HAS COLLIDED WITH GHOST

**if** len(self.maze.special\_dots\_list) >= **1**:

# IF THERE ARE STILL SPECIAL DOTS LEFT IN THE GAME

**if** self.blue\_ghost.closest\_distance\_calculator(closest\_player\_blue\_ghost, self.maze.special\_dots\_list) == self.blue\_ghost.get\_start\_pos():

# IF THE DOT THE GHOST IS AIMING FOR IS THE SAME DOT AS BEFORE

**if** self.blue\_ghost.closest\_distance\_calculator(closest\_player\_blue\_ghost, self.maze.special\_dots\_list) == self.blue\_ghost.get\_end\_pos():

# IF THE GHOST HAS ALREADY REACHED THIS DOT

self.blue\_ghost.update\_direction(self.blue\_ghost.random\_movement(self.blue\_ghost.get\_direction(), self.maze))

# MOVE RANDOMLY

**else**:

self.player\_zero\_function(player\_two, self.blue\_ghost.closest\_distance\_calculator(closest\_player\_blue\_ghost, self.maze.special\_dots\_list), self.blue\_ghost, **1**)

# OTHERWISE MOVE TO THE SPECIAL DOT THAT IS CLOSEST TO THE PLAYER

**else**:

self.blue\_ghost.update\_end\_pos(None)

# OTHERWISE RESET THE END POSITION

self.player\_zero\_function(player\_two, self.blue\_ghost.closest\_distance\_calculator(closest\_player\_blue\_ghost, self.maze.special\_dots\_list), self.blue\_ghost, **1**)

# MOVE TO THE SPECIAL DOT THAT IS CLOSEST TO THE PLAYER

self.blue\_ghost.update\_start\_pos(self.blue\_ghost.closest\_distance\_calculator(closest\_player\_blue\_ghost, self.maze.special\_dots\_list))

# CALCULATE THE SPECIAL DOT THAT IS CLOSEST TO THE PLAYER AND ASSIGN START POS THAT VALUE

**else**:

self.blue\_ghost.update\_direction(self.blue\_ghost.random\_movement(self.blue\_ghost.get\_direction(), self.maze))

# IF THERE ARE NO MORE SPECIAL DOTS LEFT JUST MOVE RANDOMLY

# BLACK GHOST MOVEMENT

**if** **not** player\_two[**1**][**2**][**6**]:

# IF THE BLACK GHOST IS NOT FRIGHTENED

**if** self.player\_one.get\_level() >= **3**:

# IF WE ARE ON OR PAST LEVEL THREE

self.pacman\_dead\_check(self.black\_ghost)

# CHECK TO SEE IF A PLAYER HAS COLLIDED WITH GHOST

**if** self.black\_ghost.get\_change\_mode():

# IF THE BLACK GHOST HAS CHANGED MODE

**if** self.black\_ghost.get\_ghost\_clock() >= (**25** \* fps):

# IF 25 SECONDS HAS PASSED (15 SECONDS OF CHASING THE PLAYER)

self.black\_ghost.update\_change\_mode(False)

self.black\_ghost.update\_ghost\_clock(**0**)

# CHANGE THE GHOST BACK AND RESET THE CLOCK

**else**:

self.player\_zero\_function(player\_two, closest\_player\_black\_ghost, self.black\_ghost, **2**)

# OTHERWISE MAKE THE GHOST CHASE THE PLAYER AND CHANGE THE IMAGE

**else**:

**if** self.black\_ghost.get\_ghost\_clock() >= (**10** \* fps):

# IF 10 SECONDS HAS PASSED

self.black\_ghost.update\_change\_mode(True)

# CHANGE MODE

**else**:

self.black\_ghost.update\_direction(self.black\_ghost.random\_movement(self.black\_ghost.get\_direction(), self.maze))

# OTHERWISE THE GHOST MOVES RANDOMLY

# ORANGE GHOST MOVEMENT

**if** **not** player\_two[**1**][**2**][**6**]:

# IF THE ORANGE GHOST IS NOT FRIGHTENED

**if** self.player\_one.get\_lives() >= **4**:

# IF WE ARE PASSED LEVEL FIVE

**if** self.orange\_ghost.get\_ghost\_clock() > (**10** \* fps):

# IF 10 SECONDS HAS PASSED

self.pacman\_dead\_check(self.orange\_ghost)

# CHECK TO SEE IF A PLAYER HAS COLLIDED WITH GHOST

self.player\_zero\_function(player\_two, self.orange\_ghost.orange\_ghost\_movement(closest\_player\_orange\_ghost, self.maze.maze\_pixel\_list, orange\_closest\_player), self.orange\_ghost, **3**)

# MAKE THE GHOST CHASE TWO POSITIONS AHEAD OF PACMAN

**if** len(self.maze.special\_dots\_list) == **0** **and** len(self.maze.dots\_list) == **0**:

# IF THE THERE ARE NO MORE DOTS LEFT IN THE MAZE

self.player\_one.update\_level\_over(True)

self.player\_one.update\_timer(**0**)

self.player\_one.update\_has\_started(False)

# THE GAME IS OVER

self.player\_one.update\_timer(self.player\_one.get\_timer() + **1**)

# INCREASE TIME BY ONE

**else**:

# IF A PLAYER DIES

**if** self.player\_one.get\_lives() >= **1**:

# IF THE PLAYERS STILL HAS LIVES REMAINING

self.reset(player\_two[**0**][**5**])

# MOVE ALL ENTITIES TO THEIR STARTING POS AND CHANGE DIRECTION TO NEUTRAL

self.player\_one.update\_sprite(self.player\_one.death\_images, **0.06**, **5**, **5**)

self.player\_two.update\_sprite(self.player\_two.death\_images, **0.06**, **5**, **5**)

# MAKE IMAGE LOOK ANIMATED

**else**:

# PLAYER HAS RUN OUT OF LIVES AND THE GAME IS OVER

self.red\_ghost.update\_direction((**0**, **0**))

self.blue\_ghost.update\_direction((**0**, **0**))

self.black\_ghost.update\_direction((**0**, **0**))

self.orange\_ghost.update\_direction((**0**, **0**))

self.player\_one.update\_update\_direction((**0**, **0**))

self.player\_one.update\_direction((**0**, **0**))

self.player\_class.state = "gameover\_multiplayer"

# FREEZE ALL ENTITIES AND DISPLAY IMAGE ON SCREEN

**else**:

# IF THE PLAYER COMPLETES THE LEVEL

self.reset(player\_two[**0**][**5**])

self.maze.reset()

# RESET ALL ENTITIES AND THE MAZE

**if** self.player == **0**:

# IF PLAYER IS PLAYER 0

**if** **not** self.player\_one.get\_has\_started() **and** **not** self.is\_waiting **and** self.player\_one.get\_lives() >= **1**:

# IF THE PLAYER HAS NOT STARTED BUT THEY ARE NOT WAITING AND THE PLAYER IS STILL ALIVE

self.player\_one.update\_timer(self.player\_one.get\_timer() + **1**)

# INCREASE THE TIME

**if** self.time == **420**:

# IF IT IS TIME TO START

self.player\_one.update\_timer(**0**)

self.player\_one.update\_has\_started(True)

# RESET VARIABLES

**if** **not** self.player\_one.get\_is\_alive():

# IF THE PLAYER DIED

self.player\_one.reset(multiplayer\_starting\_positions[player\_two[**0**][**5**]])

self.player\_one.update\_lives(self.player\_one.get\_lives() - **1**)

# RESET VARIABLES

**if** self.player\_one.get\_level\_over():

# IF THE LEVEL IS OVER

self.player\_one.reset(multiplayer\_starting\_positions[player\_two[**0**][**5**]])

# RESET POSITION

self.client\_variable.send\_data\_no\_return(json.dumps((((self.red\_ghost.get\_pixel\_position()[**0**], self.red\_ghost.get\_pixel\_position()[**1**], self.red\_ghost.get\_direction()[**0**], self.red\_ghost.get\_direction()[**1**], self.red\_ghost.rect.center[**0**], self.red\_ghost.rect.center[**1**], self.red\_ghost.get\_is\_frightened(), self.red\_ghost.get\_is\_alive(), **0**), (self.blue\_ghost.get\_pixel\_position()[**0**], self.blue\_ghost.get\_pixel\_position()[**1**], self.blue\_ghost.get\_direction()[**0**], self.blue\_ghost.get\_direction()[**1**], self.blue\_ghost.rect.center[**0**], self.blue\_ghost.rect.center[**1**], self.blue\_ghost.get\_is\_frightened(), self.blue\_ghost.get\_is\_alive(), **1**), (self.black\_ghost.get\_pixel\_position()[**0**], self.black\_ghost.get\_pixel\_position()[**1**], self.black\_ghost.get\_direction()[**0**], self.black\_ghost.get\_direction()[**1**], self.black\_ghost.rect.center[**0**], self.black\_ghost.rect.center[**1**], self.black\_ghost.get\_is\_frightened(), self.black\_ghost.get\_is\_alive(), **2**), (self.orange\_ghost.get\_pixel\_position()[**0**], self.orange\_ghost.get\_pixel\_position()[**1**], self.orange\_ghost.get\_direction()[**0**], self.orange\_ghost.get\_direction()[**1**], self.orange\_ghost.rect.center[**0**], self.orange\_ghost.rect.center[**1**], self.orange\_ghost.get\_is\_frightened(), self.orange\_ghost.get\_is\_alive(), **3**)), self.player\_one.get\_timer(), self.player\_one.get\_lives(), self.player\_one.get\_level(), self.player\_one.get\_is\_alive(), self.player\_one.get\_level\_over(), self.player\_one.get\_has\_started())))

# SEND GHOST DATA AS WELL AS GLOBAL VARIABLES TO THE SERVER

**if** self.player == **1**:

# IF PLAYER 1

**if** **not** self.player\_one.get\_has\_started() **and** **not** self.is\_waiting **and** self.player\_one.get\_lives() >= **1**:

# IF THE PLAYER HAS NOT STARTED BUT THEY ARE NOT WAITING AND THE PLAYER IS STILL ALIVE

**if** self.time == **420**:

# IF IT IS TIME TO START

self.player\_one.reset(multiplayer\_starting\_positions[player\_two[**0**][**5**]])

# RESET THE PLAYER

self.player\_one\_function(player\_two, self.red\_ghost, **0**)

self.player\_one\_function(player\_two, self.blue\_ghost, **1**)

self.player\_one\_function(player\_two, self.black\_ghost, **2**)

self.player\_one\_function(player\_two, self.orange\_ghost, **3**)

# UPDATE GHOST POSITIONS AND VARIABLES DEPENDING ON DATA FROM SERVER

**if** self.is\_paused\_time != **30** \* **120**:

# IF THE TIME IS INCORRECT

self.is\_paused\_time = **30** \* **120**

# RESET TIME

**else**:

self.red\_ghost.force\_pause()

self.red\_ghost.update\_direction((**0**, **0**))

self.blue\_ghost.force\_pause()

self.blue\_ghost.update\_direction((**0**, **0**))

self.black\_ghost.force\_pause()

self.black\_ghost.update\_direction((**0**, **0**))

self.orange\_ghost.force\_pause()

self.orange\_ghost.update\_direction((**0**, **0**))

self.player\_one.force\_pause()

self.is\_paused\_time -= **2**

# PAUSE ALL ENTITIES AND REDUCE TIME

**if** self.player\_one.get\_is\_paused():

# IF THE PLAYER PAUSED THE GAME

key\_pressed = pygame.key.get\_pressed()

# GET THE KEY PRESSED

**if** key\_pressed[pygame.K\_r]:

# IF THE PRESS R

self.player\_one.update\_is\_paused(False)

self.player\_one.update\_has\_started(False)

self.client\_variable.send\_data\_no\_return(json.dumps((self.player\_one.get\_has\_started(), **0**)))

# SEND DATA TO DATABASE

**if** self.is\_paused\_time == **0**:

# IF THEY RUN OUT OF TIME

self.player\_one.update\_is\_paused(False)

self.player\_one.update\_has\_started(False)

self.client\_variable.send\_data\_no\_return(json.dumps((self.player\_one.get\_has\_started(), **0**)))

# SEND DATA TO DATABASE

self.all\_sprites.update()

# UPDATES ALL SPRITES

**def** **draw\_screen**(self, screen, game\_background, game\_over\_image):

##### draw\_screen #######

# Parameters : screen:Canvas, game\_background:Image, game\_over\_image, Image

# Return Type : None

# Purpose :- Controls entire screen for multiplayer game

##########################

# DRAW SCREEN FUNCTION

screen.blit(game\_background, (**11**, **0**))

# DRAW GAME MAZE ON SCREEN

**for** each\_coin **in** self.maze.dots\_list:

screen.blit(self.player\_class.dot\_image, (each\_coin[**0**] + **10**, each\_coin[**1**] + **10**))

# DRAW ALL THE DOTS ON THE SCREEN

**for** each\_special\_coin **in** self.maze.special\_dots\_list:

screen.blit(self.player\_class.special\_dot\_image, (each\_special\_coin[**0**] + **2**, each\_special\_coin[**1**] + **2**))

# DRAW THE SPECIAL DOTS ON SCREEN

self.all\_sprites.draw(screen)

# DRAW ALL THE PLAYERS AND GHOSTS

**if** self.is\_waiting:

# IF THE GAME HAS NOT STARTED YET

screen.blit(self.player\_class.grey\_screen, (**0**, **0**))

**if** self.time < **120**:

self.player\_class.text(menu\_font, **25**, LIGHT\_BLUE, "Waiting For Player 2.", BLACK, screen, **284**, **125**)

**if** **120** <= self.time < **240**:

self.player\_class.text(menu\_font, **25**, LIGHT\_BLUE, "Waiting For Player 2..", BLACK, screen, **283**, **125**)

**if** **240** <= self.time < **360**:

self.player\_class.text(menu\_font, **25**, LIGHT\_BLUE, "Waiting For Player 2...", BLACK, screen, **282**, **125**)

**if** self.time == **360**:

self.time = **0**

self.time += **1**

# DRAW ON SCREEN WITH ANIMATED IMAGE

**if** **not** self.player\_one.get\_has\_started() **and** **not** self.is\_waiting **and** self.player\_one.get\_lives() >= **1**:

# IF THE GAME HAST STARTED, THE PLAYER IS NOT WAITING AND THEY STILL HAVE LIVES REMAINING

**if** self.time < **120**:

self.player\_class.text(menu\_font, **20**, BLUE, "3", BLACK, screen, **265**, **270**)

**if** **120** <= self.time < **240**:

self.player\_class.text(menu\_font, **20**, BLUE, "2", BLACK, screen, **265**, **270**)

**if** **240** <= self.time < **360**:

self.player\_class.text(menu\_font, **20**, BLUE, "1", BLACK, screen, **265**, **270**)

**if** **360** <= self.time < **420**:

self.player\_class.text(menu\_font, **20**, BLUE, "GO!", BLACK, screen, **265**, **270**)

# DRAW 3 2 1 ON THE SCREEN

self.player\_class.text(arialbold\_font, **14**, WHITE, "LIVES: ", BLACK, screen, **40**, **10**)

# LIVES TEXT HEADING ON SCREEN

self.player\_class.draw\_lives(**65**, **20**, screen, self.player\_one.get\_lives(), self.player\_one.heart\_image)

# DRAW LIVES AS PACMAN IMAGES

self.player\_class.text(arialbold\_font, **14**, WHITE, "LEVEL: {} ".format(self.player\_one.get\_level()), BLACK, screen, screen\_width//**2**, **10**)

# LEVEL TEXT ON SCREEN AS WELL AS LEVEL

self.player\_class.text(arialbold\_font, **14**, WHITE, "THEIR SCORE: {}".format(self.player\_two.get\_score()), BLACK, screen, (screen\_width-width\_buffer)-len("YOUR SCORE: {}".format(self.player\_two.get\_score()))-**30**, **610**)

# SCORE TEXT ON SCREEN AS WELL AS SCORE

self.player\_class.text(arialbold\_font, **14**, WHITE, "MY SCORE: {}".format(self.player\_one.get\_score()), BLACK, screen, (len("YOUR SCORE: {}".format(self.player\_two.get\_score())) + **50**), **610**)

# SCORE TEXT ON SCREEN AS WELL AS SCORE

**if** self.player\_one.get\_is\_paused():

# IF YOU PAUSED THE GAME

screen.blit(self.player\_class.grey\_screen, (**0**, **0**))

self.player\_class.text(menu\_font, **25**, LIGHT\_BLUE, "Press R To Resume Game", BLACK, screen, **284**, **125**)

self.player\_class.text(arialbold\_font, **14**, WHITE, "TIME: {} s".format(self.is\_paused\_time//**120**), BLACK, screen, (screen\_width - width\_buffer) - len("TIME: {}".format(self.is\_paused\_time)) - **20**, **10**)

self.player\_class.button(**202**, **131**, **368**, **35**, LIGHT\_MAZE\_BLUE, DARK\_MAZE\_BLUE, screen, menu\_font, **20**, BLACK, "MENU SCREEN", "menu")

self.player\_class.button(**202**, **131**, **436**, **35**, LIGHT\_MAZE\_BLUE, DARK\_MAZE\_BLUE, screen, menu\_font, **20**, BLACK, "QUIT", "quit")

# DISPLAY THIS ON SCREEN

**if** self.player\_two.get\_is\_paused():

# IF THE OTHER PLAYER PAUSED THE GAME

screen.blit(self.player\_class.grey\_screen, (**0**, **0**))

self.player\_class.text(arialbold\_font, **14**, WHITE, "TIME: {} s".format(self.is\_paused\_time//**120**), BLACK, screen, (screen\_width - width\_buffer) - len("TIME: {}".format(self.is\_paused\_time)) - **20**, **10**)

self.player\_class.text(menu\_font, **25**, LIGHT\_BLUE, "Player 2 Has Paused The Game", BLACK, screen, **284**, **125**)

self.player\_class.button(**202**, **131**, **368**, **35**, LIGHT\_MAZE\_BLUE, DARK\_MAZE\_BLUE, screen, menu\_font, **20**, BLACK, "MENU SCREEN", "menu")

self.player\_class.button(**202**, **131**, **436**, **35**, LIGHT\_MAZE\_BLUE, DARK\_MAZE\_BLUE, screen, menu\_font, **20**, BLACK, "QUIT", "quit")

# DISPLAY THIS ON SCREEN

**if** self.player\_class.state == "gameover\_multiplayer":

# IF THE GAME IS OVER

screen.blit(self.player\_class.grey\_screen, (**0**, **0**))

# MAKE THE SCREEN GRAY

screen.blit(game\_over\_image, (**92**, **100**))

# DRAW GAME OVER IMAGE OVER THE GAME

self.player\_class.button(**60**, **131**, **400**, **35**, WHITE, GRAY, screen, menu\_font, **20**, BLACK, "MENU SCREEN", "menu")

# MENU BUTTON

self.player\_class.button(**343**, **131**, **400**, **35**, WHITE, GRAY, screen, menu\_font, **20**, BLACK, "QUIT", "quit")

# PLAY AGAIN BUTTON

pygame.display.flip()

# UPDATE DISPLAY

**def** **player\_zero\_function**(self, input\_player, position, ghost, ghost\_identifier):

##### player\_zero\_function #######

# Parameters : input\_player:int, position:Tuple, ghost:object, ghost\_identifier:int

# Return Type : None

# Purpose :- Calculates shortest distance the ghost must travel to its end position

##########################

# FUNCTION FOR PLAYER 0

ghost.search((**0**, (input\_player[**1**][ghost\_identifier][**0**], input\_player[**1**][ghost\_identifier][**1**]), (input\_player[**1**][ghost\_identifier][**0**], input\_player[**1**][ghost\_identifier][**1**])), position, self.maze.walls)

ghost.update\_direction(ghost.direction\_calculator(ghost.visited, ghost.get\_direction(), position, self.maze))

# FINDS SHORTEST PATH TO A POSITION AND MOVES THERE

**def** **player\_one\_function**(self, input\_player, ghost, ghost\_identifier):

##### player\_one\_function #######

# Parameters : input\_player:int, ghost:object, ghost\_identifier:int

# Return Type : None

# Purpose :- Updates all of the ghost’s variables depending on the data received from the server

##########################

# FUNCTION FOR PLAYER 1

ghost.update\_direction((input\_player[**1**][ghost\_identifier][**2**], input\_player[**1**][ghost\_identifier][**3**]))

ghost.rect.center = (input\_player[**1**][ghost\_identifier][**4**], input\_player[**1**][ghost\_identifier][**5**])

ghost.update\_is\_frightened(input\_player[**1**][ghost\_identifier][**6**])

ghost.update\_is\_alive(input\_player[**1**][ghost\_identifier][**7**])

# UPDATE THE GHOSTS POSITION

**def** **reset**(self, player\_id):

##### reset #######

# Parameters : player\_id:int

# Return Type : None

# Purpose :- Resets all the positions and variables for the ghosts and pacman including their direction

##########################

# RESET FUNCTION

self.red\_ghost.reset(red\_ghost\_starting\_position)

self.blue\_ghost.reset(blue\_ghost\_starting\_position)

self.black\_ghost.reset(black\_ghost\_starting\_position)

self.orange\_ghost.reset(orange\_ghost\_starting\_position)

self.player\_one.update\_direction((**0**, **0**))

self.player\_one.update\_update\_direction((**0**, **0**))

self.player\_one.rect.center = multiplayer\_starting\_positions[player\_id]

self.player\_one.update\_pixel\_position(vector((self.player\_one.rect.x // cell\_width) \* cell\_width, (self.player\_one.rect.y // cell\_width) \* cell\_width))

# RESETS ALL ENTITIES TO STARTING POSITIONS AND MAKES THEM STATIONARY

**def** **reset\_game**(self, player\_id):

##### reset\_game #######

# Parameters : player\_id:int

# Return Type : None

# Purpose :- Entirely reset all the players , ghosts and maze back to default

##########################

# RESET GAME FUNCTION

self.player\_one.reset\_game(multiplayer\_starting\_positions[player\_id])

self.red\_ghost.reset\_game(red\_ghost\_starting\_position)

self.blue\_ghost.reset\_game(blue\_ghost\_starting\_position)

self.black\_ghost.reset\_game(black\_ghost\_starting\_position)

self.orange\_ghost.reset\_game(orange\_ghost\_starting\_position)

self.maze.reset()

# RESETS ALL ENTITIES AND MAZES TO THEIR ORIGINAL VALUE

**def** **pacman\_dead\_check**(self, ghost):

##### pacman\_dead\_check #######

# Parameters : ghost:object

# Return Type : None

# Purpose :- Checks to see if either of the players have collided with any ghosts.

##########################

# CHECKS IF EITHER PLAYER HAS COLLIDED WITH GHOST

**if** self.player\_one.get\_pixel\_position() == ghost.get\_pixel\_position() **or** self.player\_two.get\_pixel\_position() == ghost.get\_pixel\_position():

# COLLISION WITH GHOST

self.player\_one.update\_is\_alive(False)

self.player\_one.update\_timer(**0**)

self.player\_one.update\_has\_started(False)

# PLAYERS ARE NO LONGER ALIVE AND PLAYER TIMER RESTARTS

**def** **dot\_check**(self, player):

##### dot\_check #######

# Parameters : player:object

# Return Type : None

# Purpose :- Checks if either player has eaten a pac dot or a special dot

##########################

# DOT CHECK FUNCTION

**if** player.get\_pixel\_position() **in** self.maze.dots\_list:

# IF THE PLAYER IS ON A DOT

self.maze.dots\_list.remove(player.get\_pixel\_position())

# REMOVE THAT DOT FROM THE LIST

player.update\_score(player.get\_score() + **5**)

# GIVE THEM 5 POINTS

**if** player.get\_pixel\_position() **in** self.maze.special\_dots\_list:

# IF THE PLAYER IS ON A SPECIAL DOT

self.red\_ghost.update\_is\_frightened(True)

self.blue\_ghost.update\_is\_frightened(True)

self.black\_ghost.update\_is\_frightened(True)

self.orange\_ghost.update\_is\_frightened(True)

# MAKE ALL GHOSTS FRIGHTENED AND RESET HIS TIMER

self.maze.special\_dots\_list.remove(player.get\_pixel\_position())

# REMOVE SPECIAL DOT FROM LIST

player.update\_score(player.get\_score() + **20**)

# GIVES PLAYER 20 POINTS

self.player\_one.update\_timer(**0**)

self.player\_two.update\_timer(**0**)

# SETS TIME TO 0

**maze.py**

**from** **game\_settings** **import** \*

**import** **random**

# DIMENSIONS x = 19 y = 22

**class** **Maze**:

**def** **\_\_init\_\_**(self):

self.maze = [[**1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**],

[**1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**],

[**1**, **0**, **1**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **1**, **0**, **1**],

[**1**, **0**, **1**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **1**, **0**, **1**],

[**1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**],

[**1**, **0**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **0**, **1**],

[**1**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **1**],

[**1**, **1**, **1**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **1**, **1**, **1**],

[**0**, **0**, **0**, **1**, **0**, **1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**, **0**, **1**, **0**, **0**, **0**],

[**1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **0**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**],

[**1**, **0**, **0**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **0**, **0**, **1**],

[**1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**],

[**0**, **0**, **0**, **1**, **0**, **1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**, **0**, **1**, **0**, **0**, **0**],

[**1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**],

[**1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**],

[**1**, **0**, **1**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **0**, **1**, **1**, **0**, **1**],

[**1**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **1**],

[**1**, **1**, **0**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **0**, **1**, **1**],

[**1**, **0**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **1**, **0**, **0**, **0**, **0**, **1**],

[**1**, **0**, **1**, **1**, **1**, **1**, **1**, **1**, **0**, **1**, **0**, **1**, **1**, **1**, **1**, **1**, **1**, **0**, **1**],

[**1**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **0**, **1**],

[**1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**, **1**]

]

self.fruit\_images = []

self.fruit\_positions = []

self.reject\_list = [(**0**, **8**), (**1**, **8**), (**2**, **8**), (**0**, **12**), (**1**, **12**), (**2**, **12**), (**9**, **12**), (**8**, **10**), (**9**, **10**), (**10**, **10**), (**9**, **8**), (**9**, **9**), (**16**, **8**), (**17**, **8**), (**18**, **8**), (**16**, **12**), (**17**, **12**), (**18**, **12**), (**1**, **10**), (**2**, **10**), (**3**, **10**), (**15**, **10**), (**16**, **10**), (**17**, **10**)]

self.special\_dots\_list = [(**28**, **84**), (**476**, **84**), (**28**, **448**), (**476**, **448**)]

self.outside\_maze = [(**28**, **224**), (**56**, **224**), (**448**, **224**), (**476**, **224**), (**28**, **336**), (**56**, **336**), (**448**, **336**), (**476**, **336**)]

self.walls, self.maze\_pixel\_list, self.dots\_list = self.create\_walls(self.maze, self.outside\_maze)

# RUN METHOD TO MAKE WALLS DOTS LIST AND A LIST OF ALL AVAILABLE POSITIONS IN THE MAZE

**def** **create\_walls**(self, maze, outside\_maze):

##### create\_walls #######

# Parameters : maze:List, outside\_maze:List

# Return Type : wall\_list:List, maze\_pixel\_list:List, dots\_list:List

# Purpose :- Assigns a grid position to each wall in the maze and returns the value as a 2D array as well as the positions of all the dots in the game

##########################

wall\_list = []

maze\_pixel\_list = []

dots\_list = []

# MAZE CREATION FUNCTION

**for** y\_position **in** range(**0**, len(maze)):

**for** x\_position **in** range(**0**, len(maze[y\_position])):

# FOR EACH CELL IN THE MAZE

maze\_pixel\_list.append((x\_position \* cell\_width, y\_position \* cell\_height))

# APPEND IT TO MAZE\_PIXEL\_LIST

**if** maze[y\_position][x\_position] == **1**:

# IF THE POSITION IS A 1

wall\_list.append((x\_position \* cell\_width, y\_position \* cell\_height))

# IT IS A WALL ADD IT WALLS\_LIST

**if** maze[y\_position][x\_position] == **0** **and** (x\_position, y\_position) **not** **in** self.reject\_list **and** (x\_position\***28**, y\_position\***28**) **not** **in** self.special\_dots\_list:

# IF IT IS ZERO AND IT IS NOT A REJECTION SPACE OR A SPECIAL DOT SPACE

dots\_list.append((x\_position \* cell\_width, y\_position \* cell\_height))

# ADD IT TO DOTS LIST

**for** each\_pos **in** wall\_list:

# FOR EACH POSITION IN THE WALLS LIST

**if** each\_pos **in** maze\_pixel\_list:

# IF THE POSITION IS IN THE MAZE PIXEL LIST

maze\_pixel\_list.remove(each\_pos)

# REMOVE THAT POSITION

**for** each\_position **in** outside\_maze:

# FOR EACH POSITION THAT IS OUTSIDE THE MAZE

**if** each\_position **in** maze\_pixel\_list:

# IF THE POSITION IS IN THE MAZE PIXEL LIST

maze\_pixel\_list.remove(each\_position)

# REMOVE THIS POSITION FROM MAZE PIXEL LIST

**return** wall\_list, maze\_pixel\_list, dots\_list

# RETURN WALLS THE MAZE AND DOTS LIST

**def** **reset**(self):

##### reset #######

# Parameters : None

# Return Type : None

# Purpose :- Reset all maze variables

##########################

# RESET FUNCTION

self.fruit\_images = []

self.fruit\_positions = []

self.special\_dots\_list = [(**28**, **84**), (**476**, **84**), (**28**, **448**), (**476**, **448**)]

# RESET ALL VARIABLE VALUES

self.walls, self.maze\_pixel\_list, self.dots\_list = self.create\_walls(self.maze, self.outside\_maze)

# RESET DOTS LIST

**def** **random\_fruit**(self, dots\_list, fruit\_positions, fruit\_pictures):

##### random\_fruit #######

# Parameters : dots\_list:List, fruit\_positions:List, fruit\_pictures:List

# Return Type : None

# Purpose :- For singleplayer mode. Has a chance of randomly putting fruits into the maze

##########################

amount\_of\_fruit = random.randint(**0**, **3**)

# GENERATE A RANDOM AMOUNT OF FRUIT IN THE MAZE

**for** each\_fruit **in** range(**0**, amount\_of\_fruit):

# FOR EACH FRUIT THAT WILL BE IN THE MZE

fruit\_pos = random.randint(**0**, len(dots\_list) - **1**)

# RANDOMLY GENERATE THEIR POSITION IN THE MAZE

fruit\_picture = random.randint(**0**, **7**)

# RANDOMLY CHOOSE A FRUIT PICTURE

fruit\_positions.append(dots\_list[fruit\_pos])

# ADD THE POSITION TO FRUITS POSITION VARIABLE

fruit\_pictures.append(fruit\_picture)

# ADD THE FRUIT PICTURE ID TO PICTURES LIST

dots\_list.remove(dots\_list[fruit\_pos])

# REMOVE THE POSITION FROM DOTS LIST

**log\_in.py**

**from** **tkinter** **import** \*

**from** **PIL** **import** ImageTk, Image

**from** **tkinter** **import** messagebox

**import** **sqlite3**

connection = sqlite3.connect("database/players.db")

# CONNECT TO PLAYERS DATABASE

cursor = connection.cursor()

# CREATE CURSOR TO RUN SQL QUERIES

cursor.execute('CREATE TABLE IF NOT EXISTS user(user\_id integer PRIMARY KEY AUTOINCREMENT, username MESSAGE\_TEXT NOT NULL, password MESSAGE\_TEXT NOT NULL, highscore integer NOT NULL)')

# CREATE THE USER TABLE WITH THE VARIABLES IF IT DOES NOT ALREADY EXIST

connection.commit()

# CLOSE THE CONNECTION

cursor.close()

# CLOSE THE CURSOR

**class** **Login**:

**def** **\_\_init\_\_**(self, screen):

self.screen = screen

self.register\_screen = None

self.register\_username\_text = None

self.register\_password\_text = None

self.screen.title("Pacman Login System")

self.icon = PhotoImage(file = "images/pacman\_image.png")

self.logged\_in = False

self.username = None

self.password = None

register\_image = Image.open("images/register\_background.jpg")

register\_resized\_image = register\_image.resize((**400**, **500**), Image.ANTIALIAS)

self.register\_background\_image = ImageTk.PhotoImage(register\_resized\_image)

# REGISTER BACKGROUND

self.image = Image.open("images/log\_in\_screen.jpg")

self.resize\_image = self.image.resize((**600**, **400**), Image.ANTIALIAS)

self.background\_image = ImageTk.PhotoImage(self.resize\_image)

# MAIN SCREEN BACKGROUND

self.screen.iconphoto(False, self.icon)

self.screen.geometry("600x300")

self.screen.resizable(False, False)

# Screen Variables

Label(self.screen, image = self.background\_image).place(x = **0**, y= **0**, relwidth = **1**, relheight = **1**)

# BACKGROUND

login\_frame = Frame(self.screen, bg = "white")

login\_frame.place(x = **10**, y = **60**, height = "200", width = "240")

# CREATE A WHITE SQUARE ON THE BACKGROUND

Label(login\_frame, text="Login Here", font=("Fixedsys", **20**, "bold"), bg="white", fg = "#008AC9").place(x=**36**, y=**10**)

Label(login\_frame, text= "Player Login", font=("Terminal", **7**), bg="white",fg="#008AC9").place(x=**75**, y=**40**)

# TITLES AND HEADINGS

Label(login\_frame, text= "Username", font=("Fixedsys", **11**, "bold"), bg="white",fg="gray").place(x=**78**, y=**70**)

self.username\_text = Entry(login\_frame, font = ("Fixedsys", **11**), bg = "lightgray")

self.username\_text.place(x = **45**, y = **90**, height = **20**, width = **150**)

# USERNAME LABEL AND ENTERING

Label(login\_frame, text= "Password", font=("Fixedsys", **11**, "bold"), bg="white",fg="gray").place(x=**78**, y=**120**)

self.password\_text = Entry(login\_frame, font = ("Fixedsys", **11**), bg = "lightgray")

self.password\_text.place(x = **45**, y = **140**, height = **20**, width = **150**)

# PASSWORD LABEL AND ENTERING

Button(login\_frame, text = "LOGIN", bg = "white", fg = "#008AC9",font = ("Fixedsys", **15**, "bold"), command = self.login).place(x = **85**, y = **170**)

# LOGIN BUTTON

Button(login\_frame, text="New User?", bg="white", fg="gray", bd = **0**, font=("Fixedsys", **10**), command = self.register).place(x=**160**, y=**180**)

# REGISTER BUTTON

**def** **login**(self):

##### login #######

# Parameters : None

# Return Type : None

# Purpose :- Checks if the user meets the requirements to be logged into the game against the game database

##########################

# FUNCTION IF THE PLAYER CLICKS THE LOGIN BUTTON

**if** self.username\_text.get() == "" **or** self.password\_text.get() == "":

# IF THE PLAYER HAS NOT TYPED ANYTHING INTO EITHER THE USERNAME OR PASSWORD SECTION

messagebox.showerror("Error", "All fields must be filled", parent = self.screen)

# SEND AN ERROR TO THE USERS SCREEN

**else**:

# IF THE USER HAS ENTERED DATA INTO BOTH SECTIONS

database = sqlite3.connect("database/players.db")

# CONNECT TO PLAYERS DATABASE

cursor\_log = database.cursor()

# CREATE CURSOR FOR LOGIN

username\_search = 'SELECT password FROM user WHERE username = ?'

# SQL QUERY TO GET THE PLAYER PASSWORD

cursor\_log.execute(username\_search, [(self.username\_text.get())])

# SEARCH FOR THE PLAYERS PASSWORD

password = cursor\_log.fetchall()

# ASSIGN ALL THE VALUES THAT ARE RECEIVED TO PASSWORD VARIABLE

**if** password:

# IF THE PLAYERS USERNAME IS IN THE DATABASE

**if** (password[**0**][**0**]) == self.password\_text.get():

# IF THE PASSWORD IS THE SAME AS IN THE SYSTEM

self.username = self.username\_text.get()

self.password = self.password\_text.get()

self.logged\_in = True

self.screen.destroy()

# LOG PLAYER IN AND CLOSE THE SCREEN

**else**:

# INCORRECT PASSWORD

messagebox.showerror("Error", "Incorrect Password", parent = self.screen)

# SEND INCORRECT PASSWORD ERROR TO HE PLAYERS SCREEN

**else**:

# THE PLAYERS USERNAME CAN NOT BE FOUND

messagebox.showerror("Error", "No User With That Username Could Be Found", parent = self.screen)

# SEND AN ERROR TO THE PLAYERS SCREEN

database.commit()

# CLOSE THE DATABASE

cursor\_log.close()

# CLOSE THE CURSOR

**def** **register\_account**(self):

##### register\_account #######

# Parameters : None

# Return Type : None

# Purpose :- Registers a new user if their account meets requirements and saves their data to a database

##########################

# FUNCTION CALLED ONCE THE PLAYER ATTEMPTS TO CREATE A NEW ACCOUNT

**if** self.register\_username\_text.get() == "" **or** self.register\_password\_text.get() == "":

# IF THE PLAYER HAS NOT TYPED ANYTHING IN EITHER THE USERNAME OR PASSWORD SECTION

messagebox.showerror("Error", "All fields must be filled", parent = self.screen)

# SEND AN ERROR TO THEIR SCREEN

**elif** len(self.register\_username\_text.get()) < **3** **or** len(self.register\_username\_text.get()) > **15**:

# IF THEIR USERNAME IS LESS THAN 3 OR MORE THAN 15 CHARACTERS

messagebox.showerror("Error", "Username must be between 3 and 15 characters long", parent = self.screen)

# SEND AN ERROR TO THE USER

**elif** len(self.register\_password\_text.get()) < **4**:

# IF THEIR PASSWORD IS LESS THAN 4 CHARACTERS

messagebox.showerror("Error", "Password must be greater than 3 characters", parent = self.screen)

# SEND AN ERROR TO THE USERS SCREEN

**else**:

# IF THE USERNAME AND PASSWORD MEET THE REQUIREMENTS

database = sqlite3.connect("database/players.db")

# OPEN THE DATABASE

cursor\_reg = database.cursor()

# OPEN THE CURSOR

username\_search = 'SELECT username FROM user WHERE username = ?'

# SQL QUERY TO SEARCH IF THE USERNAME IS ALREADY TAKEN

cursor\_reg.execute(username\_search, [(self.register\_username\_text.get())])

# EXECUTE SQL QUERY

**if** cursor\_reg.fetchall():

# IF THE QUERY RETURNS ANY VALUE

messagebox.showerror("Error", "That Username Has Already Been Taken")

# SEND AN ERROR TO THE USER

**else**:

# OTHERWISE THE USERNAME IS NEW

insert\_data = 'INSERT INTO user(username, password, highscore) VALUES(?, ?, ?)'

# SQL QUERY TO ENTER THE PLAYER DETAILS INTO THE DATABASE

cursor\_reg.execute(insert\_data, [(self.register\_username\_text.get()), (self.register\_password\_text.get()), **0**])

# EXECUTE THE SQL QUERY

messagebox.showinfo("Successful Event", "Account Created Successfully")

# SEND AN INFO BOX TO THE USER TO SHOW THEIR ACCOUNT WAS CREATED SUCCESSFULLY

self.register\_screen.destroy()

# CLOSE THE SCREEN

database.commit()

# CLOSE THE DATABASE

cursor\_reg.close()

# CLOSE THE CURSOR

**def** **register**(self):

##### register #######

# Parameters : None

# Return Type : None

# Purpose :- A new register window and allows the user to enter the username and password they want for the account

##########################

# FUNCTION CALLED WHEN THE PLAYER CLICKS THE NEW USER BUTTON

self.register\_screen = Toplevel(self.screen)

self.register\_screen.title("Register")

self.register\_screen.geometry("300x350")

self.register\_screen.iconphoto(False, self.icon)

self.register\_screen.resizable(False, False)

# CREATING A NEW REGISTER WINDOW WITH THE SAME ICON AND NEW TITLE

Label(self.register\_screen, image = self.register\_background\_image).place(x = -**5**, y= **0**)

# PUTS BACKGROUND ONTO SCREEN

register\_frame = Frame(self.register\_screen, bg="white")

register\_frame.place(x=**10**, y=**70**, height="200", width="200")

# CREATES A WHITE SQUARE ON THE SCREEN

Label(register\_frame, text="Register Below", font=("Fixedsys", **16**, "bold"), bg="white", fg="#026FB3").place(x=**28**, y=**10**)

Label(register\_frame, text="User Registration", font=("Terminal", **10**), bg="white", fg="#026FB3").place(x=**53**, y=**30**)

# TITLES AND SUBTITLES

Label(register\_frame, text= "New Username", font=("Fixedsys", **11**, "bold"), bg="white", fg="gray").place(x=**45**, y=**50**)

self.register\_username\_text = Entry(register\_frame, font = ("Fixedsys", **11**), bg = "lightgray")

self.register\_username\_text.place(x = **26**, y = **70**, height = **20**, width = **150**)

# USERNAME LABEL AND ENTERING

Label(register\_frame, text= "New Password", font=("Fixedsys", **11**, "bold"), bg="white", fg="gray").place(x=**45**, y=**100**)

self.register\_password\_text = Entry(register\_frame, font = ("Fixedsys", **11**), bg = "lightgray")

self.register\_password\_text.place(x = **26**, y = **120**, height = **20**, width = **150**)

# PASSWORD LABEL AND ENTERING

Button(register\_frame, text="REGISTER", bg="white", fg="#008AC9", font=("Fixedsys", **15**, "bold"), command=self.register\_account).place(x=**55**, y=**160**)

# REGISTER BUTTON

**network.py**

**import** **socket**

**from** **\_thread** **import** \*

**import** **json**

**import** **copy**

PORT = **50001**

HOST = "192.168.0.4"

# [PLAYER[0], PLAYER[1], GHOSTS, TIME, LIVES, IS\_ALIVE, LEVEL\_OVER, LEVEL, IS\_WAITING, HAS\_STARTED]

# ORDER IS [CENTER\_POSITION[0], CENTER\_POSITION[1], DIRECTION[0], DIRECTION[1], SCORE, PLAYER\_NUMBER, IS\_PAUSED]

# GHOST ORDER: RED, BLUE, BLACK, ORANGE

# DATA ORDER IN GHOSTS: [PIXEL\_POS[0], PIXEL\_POS[1], DIRECTION[0], DIRECTION[1], CENTER\_POS[0], CENTER\_POS[1], IS\_FRIGHTENED, IS\_ALIVE, ID\_NUMBER]

positions = [[**238**, **350**, **0**, **0**, **5**, **0**, False], [**294**, **350**, **0**, **0**, **5**, **1**, False], [[**252**, **224**, **0**, **0**, **266**, **238**, False, True, **0**], [**224**, **280**, **0**, **0**, **238**, **294**, False, True, **1**], [**280**, **280**, **0**, **0**, **294**, **294**, False, True, **2**], [**252**, **280**, **0**, **0**, **266**, **294**, False, True, **3**]], **0**, **3**, True, False, **1**, True, False]

**def** **client**(connection, player, game\_id):

##### client #######

# Parameters : dots\_list:List, fruit\_positions:List, fruit\_pictures:List

# Return Type : None

# Purpose :- Threaded client that occurs when a player joins the network. Data is both sent, updated and received here.

##########################

connection.send(str.encode(json.dumps((game\_data[game\_id][player], game\_data[game\_id][**3**], game\_data[game\_id][**8**]))))

# SENDS INITIAL DATA UPON FIRST CONNECTING

**while** True:

**try**:

receive\_data = json.loads(connection.recv(**2048**).decode())

**if** game\_id **in** game\_data:

player\_data = game\_data[game\_id]

# TRY TO RECEIVE DATA

**if** **not** receive\_data:

**print**("you have disconnected")

# IF YOU COULD NOT RECEIVE DATA BREAK AND DISCONNECT

**else**:

**if** player\_data[**8**]:

# IF WAITING

**if** player == **1**:

# IF IT IS PLAYER TWO

player\_data[**8**] = False

# UPDATE WAITING TO FALSE

**if** receive\_data == "wait":

# IF PLAYER ONE IS WAITING

sending\_data = player\_data[**8**]

connection.sendall(str.encode(json.dumps(sending\_data)))

# SEND WAITING VARIABLE FROM GAME DATA

**if** len(receive\_data) == **2**:

# UPDATE PAUSED VARIABLES

player\_data[**9**] = receive\_data[**0**]

player\_data[**3**] = receive\_data[**1**]

**if** len(receive\_data) == **7**:

# DATA IS FOR THE GHOSTS AND SOME GLOBAL VARIABLES

player\_data[**2**] = receive\_data[**0**]

player\_data[**3**] = receive\_data[**1**]

**if** player\_data[**4**] != receive\_data[**2**]:

player\_data[**4**] = receive\_data[**2**]

**if** player\_data[**5**] != receive\_data[**4**]:

player\_data[**5**] = receive\_data[**4**]

**if** player\_data[**6**] != receive\_data[**5**]:

player\_data[**6**] = receive\_data[**5**]

**if** player\_data[**7**] != receive\_data[**3**]:

player\_data[**7**] = receive\_data[**3**]

**if** player\_data[**9**] != receive\_data[**6**]:

player\_data[**9**] = receive\_data[**6**]

**print**("received and updating:", receive\_data)

# UPDATE IT

**elif** len(receive\_data) == **6**:

# DATA RECEIVED FROM ONE PLAYER

player\_data[player][**0**] = receive\_data[**0**]

player\_data[player][**1**] = receive\_data[**1**]

**if** player\_data[player][**2**] != receive\_data[**2**]:

player\_data[player][**2**] = receive\_data[**2**]

**if** player\_data[player][**3**] != receive\_data[**3**]:

player\_data[player][**3**] = receive\_data[**3**]

player\_data[player][**4**] = receive\_data[**4**]

**if** player\_data[player][**6**] != receive\_data[**5**]:

player\_data[player][**6**] = receive\_data[**5**]

**if** player == **0**:

sending\_data = player\_data[**1**], player\_data[**2**], player\_data[**3**], player\_data[**4**], player\_data[**5**], player\_data[**6**], player\_data[**7**], player\_data[**9**]

**if** player == **1**:

sending\_data = player\_data[**0**], player\_data[**2**], player\_data[**3**], player\_data[**4**], player\_data[**5**], player\_data[**6**], player\_data[**7**], player\_data[**9**]

# print("received:", receive\_data)

# print("sending:", sending\_data)

connection.sendall(str.encode(json.dumps(sending\_data)))

# SEND THE DATA OF THE OPPOSITE PLAYER AS WELL AS SOME GLOBAL VARIABLES

# SEND [OPPOSITE\_PLAYER\_LOCATION, GHOSTS, LIVES, IS\_ALIVE, LEVEL\_OVER, LEVEL]

**else**:

**break**

**except**:

**break**

# IF SOMETHING GOES WRONG BREAK AND DISCONNECT

**print**("the connection has been lost")

**try**:

**print**("Game", game\_id, "is closing")

# del game\_data[game\_id]

**except**:

**pass**

connection.close()

# CONNECTION LOST CLOSE THE SERVER

current\_player = **0**

game\_data = {}

game\_id = **0**

# INITIAL VARIABLES

**with** socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) **as** socket\_variable:

**try**:

# CREATES SERVER

socket\_variable.bind((HOST, PORT))

socket\_variable.listen()

# LISTEN FOR CONNECTION

**except** socket.error **as** e:

str(e)

# SOCKET ERROR

**print**("Server Is Up")

# OUTPUT TO SHOW THE SERVER IS UP

**while** True:

connection, address = socket\_variable.accept()

# WHEN A USER JOINS GETS CONNECTION AND ADDRESS

**print**("Conntected to:", address)

# OUTPUT TO SHOW ADDRESS

start\_new\_thread(client, (connection, current\_player, game\_id))

# START NEW PLAYER THREAD

**if** current\_player == **1**:

game\_id += **1**

current\_player = **0**

**else**:

current\_player += **1**

game\_data[game\_id] = copy.deepcopy(positions)

# MAKE A COPY OF POSITIONS LIST

# ADD 1 TO NEW CURRENT PLAYER SO WHEN A NEW PLAYER JOINS THEIR ID IS DIFFERENT

**sprite\_object\_class.py**

**import** **pygame**

**from** **game\_settings** **import** \*

vector = pygame.math.Vector2

**class** **moving\_object**:

# NEED TO CHANGE SELF.IMAGE and SELF.IMAGELOOP VARIABLES

**def** **update\_sprite**(self, images\_to\_update, speed, variable\_x\_size, variable\_y\_size):

##### update\_sprite #######

# Parameters : images\_to\_update:List, speed:float, variable\_x\_size:int, variable\_y\_size:int

# Return Type : None

# Purpose :- 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

##########################

self.image\_loop += speed

**if** self.image\_loop > len(images\_to\_update):

self.image\_loop = **0**

self.image = images\_to\_update[int(self.image\_loop)]

self.image = pygame.transform.scale(self.image, (cell\_width - variable\_x\_size, cell\_height - variable\_y\_size))

self.image.set\_colorkey(BLACK)

**def** **orientation\_images**(self, direction, right\_images, left\_images, up\_images, down\_images, speed, variable\_x\_size, variable\_y\_size):

##### orientation\_images #######

# Parameters : direction:Tuple, right\_images:List, left\_images:List, up\_images:List, down\_images:List, speed:float, variable\_x\_size:int, variable\_y\_size:int

# Return Type : None

# Purpose :-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

##########################

**if** direction == (**1**, **0**):

self.update\_sprite(right\_images, speed, variable\_x\_size, variable\_y\_size)

**elif** direction == (-**1**, **0**):

self.update\_sprite(left\_images, speed, variable\_x\_size, variable\_y\_size)

**elif** direction == (**0**, -**1**):

self.update\_sprite(up\_images, speed, variable\_x\_size, variable\_y\_size)

**elif** direction == (**0**, **1**):

self.update\_sprite(down\_images, speed, variable\_x\_size, variable\_y\_size)

**else**:

self.update\_sprite(right\_images, speed, variable\_x\_size, variable\_y\_size)

**def** **wall\_check**(self, my\_direction, wall\_list, player\_rect):

##### wall\_check #######

# Parameters : my\_direction:Tuple, wall\_list:List, player\_rect:pygame.Rect

# Return Type : None

# Purpose :- Gets the players pixel position and when they hit a wall it stops the player

##########################

# WE ALSO UPDATE PIXEL POSITION HERE BECAUSE THE ANIMATION LOOKS NICER

pixel\_position = vector((player\_rect.x // cell\_width) \* cell\_width, (player\_rect.y // cell\_width) \* cell\_width)

**if** my\_direction == (**1**, **0**):

**if** (pixel\_position.x + **28**, pixel\_position.y) **in** wall\_list:

my\_direction = (**0**, **0**)

**if** my\_direction == (-**1**, **0**):

**if** (pixel\_position.x - **28**, pixel\_position.y) **in** wall\_list:

my\_direction = (**0**, **0**)

**elif** my\_direction == (**0**, **1**):

**if** (pixel\_position.x, pixel\_position.y + **28**) **in** wall\_list:

my\_direction = (**0**, **0**)

**if** (pixel\_position.x, pixel\_position.y + **28**) == (**252**, **252**):

my\_direction = (**0**, **0**)

**if** my\_direction == (**0**, -**1**):

**if** (pixel\_position.x, pixel\_position.y - **28**) **in** wall\_list:

my\_direction = (**0**, **0**)

**return** my\_direction, pixel\_position

**client\_network.py**

**import** **socket**

**class** **client**:

**def** **\_\_init\_\_**(self):

self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.HOST = "192.168.0.4"

self.PORT = **50001**

self.data = self.connect\_to\_server()

# CLIENT VARIABLES

**def** **connect\_to\_server**(self):

**try**:

self.client.connect((self.HOST, self.PORT))

**return** self.client.recv(**2048**).decode()

# WHEN CONNECTING TO SERVER SEND AND RECEIVE DATA

**except**:

**pass**

# IF THERE IS AN ERROR PASS

**def** **send\_data**(self, data):

**try**:

self.client.send(str.encode(data))

**return** self.client.recv(**2048**).decode()

# SEND DATA AND THEN RECEIVE DATA BACK

**except** socket.error **as** e:

# IF THERE IS AN ERROR

str(e)

**def** **send\_data\_no\_return**(self, data):

**try**:

self.client.send(str.encode(data))

# SEND DATA WITHOUT RECEIVING

**except** socket.error **as** e:

# IF THERE IS AN ERROR

str(e)

**def** **return\_data**(self):

**return** self.data

# GETS INITIAL DATA WHEN A PLAYER FIRST CONNECTS

**game\_settings.py**

**import** **os**

game\_folder = os.path.dirname(\_\_file\_\_)

image\_folder = os.path.join(game\_folder, "images")

fonts\_folder = os.path.join(game\_folder, "fonts")

music\_folder = os.path.join(game\_folder, "music")

pacman\_images = os.path.join(image\_folder, "pacman")

ghost\_images = os.path.join(image\_folder, "ghosts")

fruit\_images = os.path.join(image\_folder, "fruit")

about\_images = os.path.join(image\_folder, "about")

# FOLDERS

BLACK = **0**, **0**, **0**

YELLOW = **150**, **150**, **0**

LIGHT\_YELLOW = **255**, **255**, **0**

WHITE = **255**, **255**, **255**

GRAY = **170**, **170**, **170**

RED = **150**, **0**, **0**

LIGHT\_RED = **255**, **0**, **0**

BLUE = **0**, **0**, **150**

LIGHT\_BLUE = **0**, **0**, **255**

GREEN = **0**, **150**, **0**

LIGHT\_GREEN = **0**, **255**, **0**

LIGHT\_MAZE\_BLUE = **0**, **128**, **248**

DARK\_MAZE\_BLUE = **0**, **90**, **174**

# COLOURS

screen\_width = **534**

screen\_length = **616**

width\_buffer = **26**

length\_buffer = **15**

# SCREEN SIZE

starting\_position\_x = **266**

starting\_position\_y = **350**

# SINGLEPLAYER STARTING POSITION

multiplayer\_starting\_positions = [[**294**, **350**], [**238**, **350**]]

# MULTIPLAYER STARTING POSITIONS

red\_ghost\_starting\_position = **252**, **224**

blue\_ghost\_starting\_position = **224**, **280**

black\_ghost\_starting\_position = **280**, **280**

orange\_ghost\_starting\_position = **252**, **280**

# GHOST STARTING POS

game\_title = "Pacman"

# GAME TITLE

menu\_font = os.path.join(fonts\_folder, "a\_font.ttf")

arial\_font = os.path.join(fonts\_folder, "b\_font.ttf")

arialbold\_font = os.path.join(fonts\_folder, "arial\_bold.ttf")

# FONTS

cell\_width = **28**

cell\_height = **28**

# CELL WIDTH AND HEIGHT

fps = **120**

# FPS