Full Name: Nathaniel Christopher Bates

Candidate Number: 7012

Centre Name: Prince William School

Centre Number: 27136

Project Tile: Minesweeper

Qualification Code: 7517/1

Date: xx/xx/x

Minesweeper

Contents

[Analysis: 3](#_Toc101125728)

[The Problem 3](#_Toc101125729)

[Statement and Outline 3](#_Toc101125730)

[How I Researched it 3](#_Toc101125731)

[Research 4](#_Toc101125732)

[Minesweeper 4](#_Toc101125733)

[Minesweeper online 4](#_Toc101125734)

[Googles Minesweeper 5](#_Toc101125735)

[Microsoft’s Minesweeper 5](#_Toc101125736)

[Problems With Minesweeper 6](#_Toc101125737)

[Prospective Users 7](#_Toc101125738)

[User Needs 7](#_Toc101125739)

[Questionnaire 7](#_Toc101125740)

[Interviews 8](#_Toc101125741)

[User Investigation summary 8](#_Toc101125742)

[Understanding the Problem 9](#_Toc101125743)

[Modelling of the Problem 9](#_Toc101125744)

[Potential Solution 9](#_Toc101125745)

[Specific Objectives: 10](#_Toc101125746)

[Objectives 10](#_Toc101125747)

[Modules 10](#_Toc101125748)

[Main Game Procedures 10](#_Toc101125749)

[Main Algorithms 11](#_Toc101125750)

[Design Model 11](#_Toc101125751)

[Key Variable analysis 11](#_Toc101125752)

[Object Orientated Plan 12](#_Toc101125753)

[Class Diagram 12](#_Toc101125754)

[Documented Design 14](#_Toc101125755)

[Hierarchy Diagram 14](#_Toc101125756)

[SQL Database 14](#_Toc101125757)

[Entity Relationship Diagram 16](#_Toc101125758)

[Pseudocode 16](#_Toc101125759)

[Main Menu 16](#_Toc101125760)

[Main Game 17](#_Toc101125761)

[User Interface 19](#_Toc101125762)

[Description of Game Process 19](#_Toc101125763)

[Menu Phase 19](#_Toc101125764)

[Main Game Phase 19](#_Toc101125765)

[End of Game 19](#_Toc101125766)

[Algorithms 19](#_Toc101125767)

[Grid Generation 19](#_Toc101125768)

[Removing Adjacent Empty Tiles 19](#_Toc101125769)

[Leader Board 19](#_Toc101125770)

[Class Definitions 19](#_Toc101125771)

[Board Class 19](#_Toc101125772)

[Menu Class 19](#_Toc101125773)

[Button Class 19](#_Toc101125774)

[Tile Class 19](#_Toc101125775)

[HUD Class 19](#_Toc101125776)

[Technical Solution 19](#_Toc101125777)

[Overview 19](#_Toc101125778)

[Technical Skills Used 19](#_Toc101125779)

[Detailed Code Analysis 19](#_Toc101125780)

[Testing 19](#_Toc101125781)

[Introduction and Overview 19](#_Toc101125782)

[Tests Performed 19](#_Toc101125783)

[Testing Goals 19](#_Toc101125784)

[Test Data 19](#_Toc101125785)

[Expected Outcomes 19](#_Toc101125786)

[Actual Outcomes 19](#_Toc101125787)

[Evaluation 19](#_Toc101125788)

[Objectives Met 19](#_Toc101125789)

[Analysis of User Feedback 19](#_Toc101125790)

[Realistic Improvements 19](#_Toc101125791)

[Referencing 19](#_Toc101125792)

[Art Assets 20](#_Toc101125793)

[Menu Assets 21](#_Toc101125794)

[Game Assets 21](#_Toc101125795)

[Program Code 21](#_Toc101125796)

[Main Program 21](#_Toc101125797)

[Menu Procedures Module 21](#_Toc101125798)

[Game Procedures Module 21](#_Toc101125799)

[Database Module 21](#_Toc101125800)

[Analysis Questionnaire 21](#_Toc101125801)

[User Feedback Forms 21](#_Toc101125802)

[Video 21](#_Toc101125803)

# Analysis:

## The Problem

Minesweeper is a single player puzzle game in which the player must find each mine without detonating them by using clues about the number of neighbouring mines in each field. There are many versions that have been made of this game that have been created from when it was first released in 1989 and there have been many players. The game is meant to be a quick play game which is meant to be easy to get the hang of and not be overly complicated. The problem I have with many versions of the game is that some are:

* Too complicated and stem too far from the original version of the game
* Very boring and repetitive with no customisation to gameplay
* Lack basic game mechanics such as the first tile being a bomb, which ruins the experience
* Don’t include a competitive leader board which allows users to try and compete with others for the best time or score
* Large download files, ugly graphics, contains bugs and is poorly optimised

Graphical user interface, application

Description automatically generated

## Statement and Outline

I shall reprogram minesweeper so that the initial click will always be an empty tile. Further, the game shall include a leader board and custom board and my sizes.

A solution to this problem would be a new implementation of the game which is:

* **simple and easy to understand**. I could do this by keeping the number of buttons and controls to a minimum and making the UI easy to read and understand
* **Interesting and less repetitive.** I could do this by implementing a feature to make a custom board with variable number of mines and board size
* **Less Luck based.** I could do this by adding a mechanic that allows the users first click to always be an empty tile instead of a bomb tile.
* **Competitive and engaging.** I could do this by introducing a leader board to the main menu which will allow users to compare their scores and try and beat one and another
* **Easy to access and play.** I could do this by keeping graphics simplistic so that the download size isn’t large and so the game runs smoothly.

## How I Researched it

Firstly, I researched the game and how it functions. How algorithms work, standard grid sizes, game laws, I used the minesweeper wiki and several articles to collect my research for this.

Secondly, I reviewed several versions of Minesweeper by downloading them and playing them while recording notes of what I disliked and liked about the Game. The topics I looked for included:

- User Interface e.g. How complex it is, how recreate-able it is, CUSTOMISATION

- Gameplay Mechanics e.g. How Difficult it is, how score is recorded, what algorithms are used, LEADERBOARDS

- Graphics e.g. how clean it looks, how complex the images are

- Efficiency e.g. Is it slow? Does it have trouble loading? (This may depend on my PC, but it will be controlled for each game)

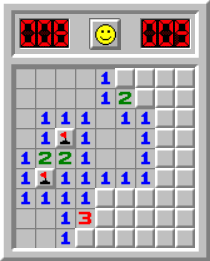
- Final Review and any extra points

Finally, I have dived into communities that competitively play Minesweeper and I read about what aspects of the game they liked and didn’t like and took it into consideration when trying to find solutions to improve the game.

## Research

### Minesweeper

Win Condition: The game is complete when all tiles that are not mines are clicked open

Loosing Condition: If you open a tile and it contains a mine the game is lost

Notes: The numbers on the board represent how many mines surround the tile (3 x 3 grid). When you reveal an empty tile, all connecting adjacent tiles are also removed. You can Flag the position of mines when you deduct, they are there so you don’t have the remember individually the places of each mine.

**2 Mines next to this Tile**

Your objective is to use the clues on the board to solve the grid as fast as possible. The faster you can solve it, the better.

**Flagged to show that a mine is under it**

Custom games can have a grid size up too 30x24 and a maximum of (x-1)(y-1) mines. When playing the game there are basic patterns which can be used to quickly help you reveal the position mines.

Graphical user interface, application, table, Teams

Description automatically generatedFor standard minesweeper games (Easy, Intermediate, Expert), there are standard board sizes and number of mines. For Beginner the board size is 8x8 to 10x10 and has 10 mines. For Intermediate the board size is 16x16 and has 40 mines. For Expert it is a 16x30 and has 99 mines.

We can calculate the percentage of mines for each difficulty:

Beginner = 10/100 \* 100 = 10% mines

Intermediate = 40/256 \* 100 = 15.625% mines

Expert = 99/480 \* 100 = 20.625%

There is a proportionality between difficulty and percentage of mines for each standard board

https://dash.harvard.edu/bitstream/handle/1/14398552/BECERRA-SENIORTHESIS-2015.pdf

https://minesweepergame.com/strategy/how-to-play-minesweeper.php#:~:text=Minesweeper%20is%20a%20game%20where,mine%20you%20lose%20the%20game!

<http://www.minesweeper.info/wiki/3BV>

https://zyxyvy.wordpress.com/2012/08/11/the-rules-of-minesweeper/

### Minesweeper online

https://minesweeperonline.com

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generatedThe interface for the main game is super simple and only shows you the time, score, and a restart button. The drop-down menus for the game settings allows customisation, set difficulty modes and a new game button.

This version of minesweeper greatly resembles the original version of minesweeper that was on windows XP. It looks and feels very retro but there are not many interesting colours.

Chart, scatter chart

Description automatically generatedThe game can be downloaded from the app store or played on browser using HTML. Its efficient but if your internet or the page refreshes all your progress is lost unless you save the export code.

It Also Includes a global leader board for daily, weekly, monthly, and all-time records. This is a great aspect as it lets players compare their scores and gives the game a competitive aspect.

### Googles Minesweeper

https://www.google.com/search?client=opera-gx&q=minesweeper&sourceid=opera&ie=UTF-8&oe=UTF-8

The interface is easy to work with and sophisticated, it displays the number of flags and the current time and the difficulty of the game. There are options at the top for the sound, sharing and closing the game. It can easily be recreated for me excluding the share button.



The gameplay ranges from Easy to medium to Hard, in this version there is no custom difficulties so there is no need for a scoring algorithm as how well you do is measured by the time taken for you to complete the game.

Chart, diagram

Description automatically generated with medium confidenceThe Graphics are minimalistic and colourful, they don’t allow the player to see through the tiles and cheat. They don’t take up much memory space due to the constant colour.

Graphical user interface

Description automatically generatedIt is run on a browser so it initially requires internet speed to load the game which will differ from person to person based on their internet connection, afterwards the processing power will determine how fast it will work. It runs quickly and smoothly due to the simplistic graphics and optimised code.

### Microsoft’s Minesweeper

https://www.microsoft.com/store/productId/9WZDNCRFHWCN

This game has a Very interesting/complex user interface in the way it has lots of options compared to the other two. It offers the 3 standard difficulty modes as well as a custom feature and Daily challenges and adventure mode. It has a larger range of gameplay but may overcomplicated the simplicity behind minesweeper.

A screenshot of a computer

Description automatically generated with medium confidenceA picture containing table

Description automatically generated

The main game looks quite ugly and boring with a very plain colour scheme and graphics. The game requires a small loading time in between games, and it freezes and crashes frequently.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, application

Description automatically generatedWhile the game offers an interesting approach on minesweeper it runs poorly and its overly complicated. Further, it requires a download of 110 MB which isn’t much but compared to versions which can be played on the browser it’s significantly more and even though it’s a downloaded version it doesn’t save your progress if the game crashes.

A picture containing text, cabinet, screenshot

Description automatically generated

### Problems With Minesweeper

I read reviews of minesweeper from <https://m.imdb.com/review/rw7532634/?ref_=tt_urv> which helped me understand what problems people had with the game and what I can do to improve it.

A screenshot of a computer

Description automatically generated with medium confidence

This Review tells me two things about the game I can improve:

* The game becomes ‘boring very quickly’ with limited game modes (Difficulty)
* The fact that the first initial click may reveal a mine and finish the game

The Solution to this could be

* Custom grid sizes and amount of mines
* Every initial click when a new game is started will reveal an Empty Tile

A screenshot of a computer

Description automatically generated

This Review tells me that the game can be confusing to understand and play. I could solve this by providing a tutorial or a user guide when you start the game. I did some more research to find out how many people know how to play minesweeper.

It turns out most people do not know how to play minesweeper.

https://imgur.com/a/vdd7TRk

## Prospective Users

The prospective users of this software are people who enjoy the occasional quick game of minesweeper when taking a break while working on their pc. I will be targeting a market of people who have played games and are above the age of 15, so I have collected a group of prospective users from my computer science class in school to answer surveys and respond to interviews to give me a greater insight into how I can create better application.

## User Needs

to gain a key understanding of the user’s needs I conducted a questionnaire towards my prospective users. They answered basics questions which helped me build a view of what they wanted from the game.

### Questionnaire

Text, letter

Description automatically generated

From this preliminary survey I have determined that there is a justification for a new minesweeper due to the lack of and demand for a less repetitive, competitive, and improved version. I made a more intricate and detailed question based of the results I received in the first one.

Graphical user interface, text, application

Description automatically generated

The results for this survey suggest that minesweeper should be played competitively with a scoring system as wells as a reduced influence of luck that influences the game. Additionally, the majority agreed that too many features would not be as enjoyable as it would bare less resemblance to the original. The results of the style where to spread out to make a clear decision so for time purposes I will go with what is most simple.

### Interviews

I interview my prospective users on a 1 on 1 discussion to gain a more detailed insight into what features the users would appreciate. Some ideas discussed where as follows:

* A quick restart button to avoiding returning to the main menu each time
* An online live action leader board for the users to compare scores/times
* A method of creating an account to view the players statistics to see.

## Understanding the Problem

### Modelling of the Problem

Diagram

Description automatically generated

### Potential Solution

My solution to this problem will involve using python and a library in python called pygame which is a set of python modules used for game development. This combined with an Object-Oriented Programming approach would allow me to make the game create a new board object using methods making it very efficient and much easier to navigate through the program code. Unfortunately, pygame isn’t as optimised as other game focused programming languages such as C# and to run it you must have installed pygame to run it. However, I’m using Pygame as it works well with 2D graphics and its simple to use. Additionally, it’s very lightweight only being 12 megabytes which makes it easy to access with a fast download. Its modular and has truly portable working on many operating systems.

https://www.pygame.org/wiki/about

A screenshot of a computer

Description automatically generated with medium confidence



Pygame includes lots of features which will help me finish the project in the given timeframe. Below I list some of the modules and how I would implement them in my project.

mouse – I can use this to get the users mouse position to see which button they have clicked

display – I can use this to display information and the Interface to the user

draw – I can use this to update the display to show information to the user

rect – I can use this along with the mouse to see which button the mouse has collided with

time – I can use this for the digital clock used in minesweeper

font – I can use this along with the draw to display strings as images on the display

transform – I can use this to change the images loaded in from the Assets folder so that they fit the correct dimensions

## Specific Objectives:

With the help of the results from the interviews with my potential users, I have laid down some initial objectives which will help me record my progress and maintain a clear vision of my next steps while programming the game.

### Objectives

1. A main menu which lets you choose your Name, Game mode and Settings
2. A live action leader board which fetches and displays the top scores
3. Methods of randomly placing mines of the field
4. Method that puts the numbers around the mines to represent how many surrounds it
5. Clock that records time on main game display
6. Score that shows the player what percentage of the board is complete
7. Tile feature that hides the board and that can be removed
8. Method that detects which tile has been clicked
9. Make deployable flags that can be placed and removed
10. A restart button that restarts the game
11. Display the number of flags placed and the number of bombs on the board on the screen (HUD)
12. Display the complete grid after they win or lose and add their score to the Database if they win
13. Make a Help option which will help the user understand how to play the game if needed.
14. Make the clock start after the initial click
15. Make the first click always not a Mine

Here I make note of what Modules/Procedures/Algorithms I will need; I will demonstrate a possible solution for algorithms of them in the Pseudocode section in the Documented Design

### Modules

My code will be split into 4 main modules to help me keep things organised. The first module will be an image loading and transforming module which will load the images and transform them into the desired size. The second module holds all the Game procedures which allow the Game to function. The third module is the main menu procedures which handles the main menu and fetches the players choices. The Final module will control the Database and its functions

### Main Game Procedures

|  |  |  |  |
| --- | --- | --- | --- |
| Name of Procedure | Function or Subroutine | Purpose | Output/Returned Value |
| DisplayAll | Subroutine | Displays the Grid under the Tiles | - |
| RemoveTiles | Function | Find which Tiles has been pressed, checks if it’s a bomb and removes the tile from the screen | Boolean which represents if play clicked a mine or not |
| AddFlag | Subroutine | Checks which tile has been right clicked and sets the Tiles flag status to on. | - |
| RemoveAdjacentTiles | Function | Removes all adjacent connecting empty tiles | List of Empty Tiles |
| CreateGrid | Method Subroutine | Creates a 2D array based of the requested board size. They are all set to ‘0’ as default | - |
| PlaceMines | Method Subroutine | Places Mines randomly avoiding the Initial click position | - |
| BombAlerts | Method Subroutine | For each mine placed it goes around it and increments the number by 1 | - |
| DisplayTime | Method Subroutine | Displays the Time on the HUD | - |
| DisplayScore | Method Subroutine | Displays the Score on the HUD | - |

### Main Algorithms

Main Menu Algorithm – This algorithm repeats until the player selects Play or Quit. It updates the display with the users’ settings and checks for any player inputs (button presses)

Game Algorithm – This algorithm repeats until the play either wins, restarts or looses the game. It checks and deals with the players decisions, displays time, score and additional things on the Hud.

## Design Model

### Key Variable analysis

This table shows a collection of important variables related to the game which will determines its functionality.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Purpose | Type | Example |
| Mine | Stores the selected number of Mines to be placed during a Game | Integer | 10 |
| BoardWidth | Store the number of Tiles for the width | Integer | 16 |
| BoardHeight | Store the number of Tiles for the height | Integer | 30 |
| Difficulty | A Number that shows what difficulty the user played on when it writes to the file | String | 2 |
| FlagLock | Stores whether or not the flags stop you clicking on a tile or not | Boolean | True |
| SaveScore | Stores whether your score will be recorded or not | Boolean | True |
| StartTile | Stores whether the first tile you click has an 100% chance of not being a bomb | Boolean | False |
| MainMenuLoop | Stores whether the main menu loop is finished or not | Boolean | True |
| GameLoop | Stores whether the Game loop is finished or not | Boolean | True |
| AllTiles | A list that stores several objects made from the Tile class which represent the tile buttons on board | Array of objects | [Tile1, Tile2, Tile3, Tile4] |
| Grid | A 2D array that stores a grid of the values of the board e.g. bomb positions, numbers | 2D array of strings | [[0, 0, 1, 2, X], [X, 1, 1, 0, 0]] |
| TileX | The dimensions of the Tile in the horizontal plane on the screen | integer | 20 |
| TileY | The dimensions of the Tile in the vertical plane on the screen | integer | 20 |

### Object Orientated Plan

I will be utilising object orientated programming to create my program. The classes I will need to create are:

**Menu –** This class will control how the menu operates and deals with the players settings/inputs

**Board –** This class will be the blueprint that creates each unique board (object) based of the 2 inputs its given from the main menu: board size, number of mines. Each game its deleted and recreated

**HUD** – This class is what controls the HUD display on the main game. It shows the Clock, Score and the Number of Flags and the Number of Mines. It also includes a restart button used to restart the game.

**Button –** This class is how the main menu will function. The buttons are used for the user to select which difficulty they want and what settings they prefer in the main menu.

**Tile –** This class holds individual information about Tiles which overlap the Board on the Display. The inherit from the button class and can be clicked. They contain 2 Boolean attributes which determines whether the flag is display on the tile and whether the tile is display.

### Class Diagram

Diagram

Description automatically generated

# Documented Design

## Hierarchy Diagram

Diagram

Description automatically generated

## SQL Database

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, website

Description automatically generated

This creates a Table called Games which has all the details about the game played and the Foreign Key player ID. It gives each game played a unique identifier and stores the date of which it was played.

Graphical user interface, text, application

Description automatically generated

This Creates a Table called Players which stores the Details of the Player Including their Player ID (Primary key), the players name and the date the account was created.

Table

Description automatically generated

### Entity Relationship Diagram

Diagram

Description automatically generated

## Pseudocode

In this Pseudocode simply show the purpose of the loop and what it should do, I will use an object orientated style in the main program to increase efficiency.

### Main Menu



### Main Game



## User Interface

### Game Interface

A picture containing text, whiteboard

Description automatically generated

### Main Menu Interface

Text

Description automatically generated

## Description of Game Process

### Menu Phase

When the user starts up the program, they will be presented will a large selection of choices. The default game settings are Intermediate with tile lock, save score and easy reset all on. They can pick from four difficulties: Easy, Intermediate, Expert or Custom And three settings: tile lock, save score and easy reset. If they pick custom the Displays that show how many mines and board size for each mode become buttons so that the user can input their choices (Their choices must be in the valid region, less than 50 and greater than 10).

Settings Description:

- Tile Lock = Stops the user pressing on a tile if there is a flag on it

- Save Score = Saves the score to the database when they win or loose.

- Easy Reset = The player can quickly Reset the game without going back to the main menu, Settings stay the same

They will also have the options to view the Leader board and sign in and create an account. The leader board is a separate display that filters through the database to show either the most recently played games or the best scored time games allowing the players to optionally competitively compare their best scores. The Create an account feature allows users to add an account to the database which will allow them to sign in with the sign in features which lets the player keep track of things such as games won, games lost and their game history.

If they sign in they will have the option to press Account statistics which will let them view their game statics:

* Player Name
* Total Games Played
* Games Won
* Games Lost
* Account Created Date
* Win to Loss ratio

There will also be a game History button at the bottom which will let them view their most recent games.

### Main Game Phase

When they player has decided on their difficulty and settings and has pressed the play button the board is generated and displayed in a fraction of a second. The timer then begins and keeps going until the game is won or lost. The score constantly shows the player how much they have completed. They can place flags down to symbolise where bombs might be with left click and remove Tiles with right click. All the information Is displayed on the HUD on the right side of game screen. They can restart at any time.

### End of Game

When they game is won or lost, the timer stops. The screen flashes Red if you lost or Green if you won and a message saying ‘win’ or ‘loose’ is displayed. All the remaining tiles are stripped from the board so you can see it. The players name, the boards difficulty, the score they got and the time are all recorded into the games table which will be utilised in the leader board and player statistics queries.

## Algorithms

### Grid Generation

Step 1) based of the requested board size created a 2D array which will represent a Grid. Fill the grid up with 0s for now as they will act as empty tiles

Step 2) Randomly place the requested number of mines around the grid avoiding the initially clicked tile and tiles with mines already underneath them. These will be represented on the Grid as ‘X’s

Step 3) For each mine go around its surface (above, below, right, left, top right, top left, bottom right, bottom left) and increment the number there by 1.

### Removing Adjacent Empty Tiles

Step 1) Create an Empty Tile Queue and a Checked Tiles List

Step 2) Add the initial empty tile that was clicked into the queue

Step 3) while the queue is not empty, select the first item in the queue, if its empty, turns the tile off then add it to the checked list and check the tiles above it, below it, to the right of it and to the left of it.

Step 4) If any of these are empty tiles add them to the end of the queue UNLESS they are in checked (this means they have already been checked and don’t need to be added to queue otherwise it would go on forever.

Step 6) Remove the first value from the Queue (move the pointer up)

Step 5) every time you check a tile, if the tile is Empty (on the grid it is a 0) turn the tile off so that it doesn’t display on the board anymore.

start

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [1, 1] | [0, 1] | [1, 0] | [1, 1] | [0, 0] |  |  |  |  |  |

End

### Leader Board

Step 1) Get the Difficulty and Setting the user wants to display (Time or Date played)

Step 2) Select the player ID and either Time or Date from the Games Table the where the difficulty is the selected difficulty and order by either Time or Date

Step 3) Using the Players ID and the players Table find the players Name and return the Name and the Time/Date

Step 4) Filter the data and render it to the screen

## Class Definitions

### Board Class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Attribute Name | Attribute Type | Initial Value | Description |
| public | boardSize | Tuple |  | Stores the dimensions of the Grid to be created in terms of how many tiles. |
| public | Mines | Integer |  | Stores how many mines should be placed on the Grid |
| public | startTile | Array | [] | Stores the position of the first click to make sure it cant be a mine |
| public | Grid | Array | [] | Stores the Grid values: mines, numbers |
| public | RecordedMinePositions | Array | [] | Stores the positions of the mines placed on the board |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Method Name | Parameters | Return Value | Description |
| public | createGird | - | - | Creates an empty grid as a 2D array with 0s |
| public | placeMines | - | - | Places mines randomly around the grid |
| public | bombAlerts | - | - | Increments the numbers around each mine |

### Menu Class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Attribute Name | Attribute Type | Initial Value | Description |
| public | Difficulty | String | ‘Intermediate’ | Stores the playes Chosen/default Difficulty |
| public | Settings | Array | - | An Array consisting of Boolean values which resemble the players chosen settings |
| public | Images | Array | - | An array with all the images that may be needed |
| public | Buttons | Array | - | An array filled with objects for buttons that will be displayed |
| public | Screen | Display | - | Pygame Display where everything is displayed too |
| public | Name | String | ‘Guest’ | The players name |
| public | PlayersID | Integer | -1 | The playersID in the players Table |
| public | NameDisplay | Image | - | Image of the players name that’s been rendered |
| public | Texts | Array | - | An Array of Texts that have been rendered |
| public | Mines | Integer | 40 | Number of mines that changes based of Difficulty |
| public | Width | Integer | 16 | Width of the board (in tiles) that changes based of Difficulty |
| public | Height | Integer | 16 | Height of the board (in tiles) that changes based of Difficulty |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Method Name | Parameters | Return Value | Description |
| public | DifficultyToValues | - | - | Turns the users difficulty (if not custom) into values for the game |
| public | CreateTexts | - |  | Creates/Updates texts to be rendered |
| public | DisplayMenu | - |  | Displays Texts, buttons and other images |
| public | CheckForClicks | - | Continue, LeaderBoardDisplayOn, CreatedAccount | Checks if any buttons have been clicked. Returns what main button has been pressed |
| public | GetCustomSizes | - | - | If the difficulty is custom it allows the player to input numbers in for the game values |

### Button Class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Attribute Name | Attribute Type | Initial Value | Description |
| public | YPosition | Integer | - | The buttons Y coordinate |
| public | XPosition | Integer | - | The buttons X coordinate |
| public | Scale | Float | - | How much the button should be scaled by |
| public | Unscaledimage | Image | - | Passed in Image |
| public | Image | Image | - | The buttons scaled image |
| public | Rect | Object | - | The buttons hit box |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Method Name | Parameters | Return Value | Description |
| public | checkForClick | - | Clicked | Returns True if the users mouse collides with the buttons Rect |
| public | Draw | Screen | - | Displays the button at the Rects top left coordinate |

### HUD Class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Attribute Name | Attribute Type | Initial Value | Description |
| Public | TotalTiles | Integer | - | Stores how many total tiles there are on the board |
| Public | EmptyTiles | Integer | 0 | Stores how many tiles the player has removed from the board |
| Public | Flags | Array | [] | Stores all the flag positions |
| Public | Mines | Integer | - | Stores how many mines there are |
| Public | Score | float | - | Stores the Score which is a percentage how far the player has completed the game |
| Public | milliseconds | Integer | 0 | Stores how many milliseconds have passed since the game started |
| Public | secs | Integer | 0 | Stores how many seconds have passed since the game started |
| Public | Mins | Integer | 0 | Stores how many minutes have passed since game started |
| Public | ScreenWidth | Integer | - | Stores the Screens Surface Width |
| Public | ScreenHeight | Integer | - | Stores the Screens Surface Height |
| Public | Difficulty | String | - | Stores the difficulty for when its saved to games table |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Method Name | Parameters | Return Value | Description |
| Public | DisplayClock | GameScreen, started, BackGroundImg | - | When the game is started is displays the time on a backgroundImg on the game screens HUD |
| Public | DisplayAmountOfFlagsandBombs | GameScreen | - | Displays how many flags are on the boards tiles and how many mines are on the boards grid |
| Public | CalculateScore | Value | - | Calculates the score using Value which is how many tiles have been removed |
| Public | DisplayScore | GameScreen, BackgroundImg | - | Displays the Score on the Game Screens HUD |
| Public | RecordScore | MainMenu | - | Records score to the Games Table using the players ID from the Main Menu Class |

### Tile Class ( Inherits from Button)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Attribute Name | Attribute Type | Initial Value | Description |
| Public | TileNumber | Integer | - | Each tile is given a unique number to identify it |
| Public | FlagOn | Boolean | - | Can be True or False, Can only be True if the Tile is also on. |
| Public | TileOn | Boolean | - | If the users click it, its turns off. If its True it is displayed on the display |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Type | Method Name | Parameters | Return Value | Description |
| Public | DisplayFlags | GameScreen | - | If FlagOn is True, it displays a flag at the coordinates |
| Public | DisplayTiles | GameScreen | - | If TileOn is True it displays a Tile at the coordinates. If its off it makes sure the flag is also off. |

# Testing

## Tests Table

These are the tests I performed on the program. These tests are measured by trying all types of data types and user inputs the user can give and whether or not it deals with it correctly.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Description | Test Data Type | Expected Result | Pass/Fail | Screen-shot |
| 1 | Creating an Account with a Name Already taken | Valid | Flashes red resets the name box. Tells you the name is taken in console | Pass | 1 |
| 2 | Trying to enter Spaces or Symbols in the username when signing in or creating account. | Erroneous | Doesn’t let you enter symbols or spaces, only numbers and letters | Pass | 2 |
| 3 | Creating an account with no username or no password | Extreme | Flashes red Tells you it’s too short in the console | Pass | 3 |
| 4 | Standard Game Difficulty generates board correctly | Valid | Generates a unique board to the correct number of tiles with the correct amount of mines | Pass | 4 |
| 5 | Custom Game Difficulty generates board correctly | Valid | Generates a unique board to the correct user inputted Height and width with the correct user inputted amount of mines | Pass | 5 |
| 6 | Placing a flag on the board | Valid | A flag appears on the tile the player clicked | Pass | 6 |
| 7 | Clicking a tile on board | Valid | If the tile is empty, it removes it and all adjacent empty ones. If it’s a number tile it removes only that tile | Pass | 7 |
| 8 | Right clicking a Flag | Valid | Removes the flag from the clicked tile | Pass | 8 |
| 9 | Clicking on a Flagged Tile when Tile lock is on | Valid | It doesn’t remove the tile | Pass | 9 |
| 10 | Clicking on a Flagged Tile when Flag lock is off | Valid | It removes the tile | Pass | 10 |
| 11 | player clicks a mine | Valid | The game is lost and it displays the loose message | Pass | 11 |
| 12 | every non mine tile is removed | Valid | The game is won and it displays the win message | Pass | 12 |
| 13 | Tapping the Account statistics buttons navigate through the data | Valid | The user can successful move back and forth through the menus | Pass | 13 |
| 14 | Signing in with correct username and password | Valid | It signs the player in and allows them to view their account statistics | Pass | 14 |
| 15 | Signing in with correct username but wrong password | Valid | Resets the n name back to guest and tells you the password was incorrect in the console | Pass | 15 |
| 16 | Leader board button is clicked | Valid | Displays 50 results which are filtered through the users choices | Pass | 16 |
| 17 | Restart Button is Clicked | Valid | Restarts the game back to the menu screen | Pass | 17 |
| 18 | Middle mouse button is clicked when Easy reset is on | Valid | Restarts the game skipping the menu screen and keeping the settings the same | Pass | 18 |
| 19 | enter letters into custom number of Mines, width, height | Erroneous | Doesn’t accept the letters and waits for numbers | Pass | 19 |
| 20 | Creating an Account by Entering a password and re-entering the password wrong | Valid | Flashes red resets the Password boxes but not the name box. Tells you the passwords don’t match in console | Pass | 20 |
| 21 | Enters nothing into Width, Height when custom | Erroneous | Sets both to the minimum accepted data | Pass | 21 |
| 22 | Enters a Massive or small Values into Width and Height | Extreme | Sets it to the maximum or minimum acceptable data depending on which its closest too | Pass | 22 |

## Screenshots

|  |  |  |
| --- | --- | --- |
| Test Number Screenshot | Screenshot | Comment |
| 1 |  | The name nat is already in the players table |
| 2 |  | Only accepts alpha and digit characters |
| 3 |  | you must enter a password and username over three characters |
| 4 |  | There are 10 mines |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  | Before and after right clicking middle square |
| 9 |  | You cant press any of the tiles |
| 10 |  | If you press a flag with tile/flag lock off it doesn’t matter |
| 11 |  | If you hit a mine, you loose the game |
| 12 |  | If there are no more tiles with no mines under them you win the game |
| 13 |  |  |
| 14 |  | When you log in you can see the account stats button |
| 15 |  | If the password is incorrect |
| 16 |  | Leader board display |
| 17 |  | Displays all |
| 18 |  | Skips the main menu features and keeps the difficulty and settings the same and immediately generates another unique board |
| 19 |  |  |
| 20 |  | Passwords do not match |
| 21 |  | If Nothing is entered it returns the smallest accepted value |
| 22 |  | If you put a too large of a number, it goes back to the largest accepted value and if you enter a value too small it rounds to the smallest accepted value |

# Evaluation

## Objectives Met

|  |  |  |  |
| --- | --- | --- | --- |
|  | Objective | Met | Comment |
| 1 | A Main Menu that allows the user to input their preferred game settings | Yes | There is well made and accessible menu which allows the user to easily interact with the game. |
| 2 | A live action leader board | Yes | The user can press the leader board button which will display the top 50 times for intermediate by default but allows them to filter by difficulty and by Time or Date created. It does this by selecting all from the games data base where the score is 100.0 |
| 3 | Random board mine generation | Yes | Each board has an incredibly small chance to be the same as the last. |
| 4 | Method of placing the bomb alerts | Yes | The number around each mine adds up correctly |
| 5 | Working accurate clock timer | Yes | The clock is accurate to two decimal places |
| 6 | Working accurate score tracker | Yes | The Score is accurate to one decimal place |
| 7 | Tile feature that hides the board and that can be removed | Yes | The tiles work as intended. They can be removed, Flagged, and removed when connected to a clicked empty tile if also empty (or 1 next to the last empty tile) |
| 8 | Method that detects which tile has been clicked | Yes | It works very accurately detecting which tiles the mouse is colliding with using the rect function. |
| 9 | Make deployable flags that can be placed and removed | Yes | The flags deploy on the tile you right click and can be removed by right clicking again |
| 10 | A restart button that restarts the game | Yes | The restart button works and brings you back to the main menu. |
| 11 | Display the number of flags placed and the number of bombs on the board on the screen (HUD) | Yes | The HUD constantly shows and updates how many flags are on the board and how many mines are on the board |
| 12 | Display the complete grid after they win or lose and add their score to the Database | Yes | When you loose or win it displays the complete grid under the tiles and your player ID, score, and name and the date it was played are added to the Games table in the database. |
| 13 | Display a Help menu to the user if they need help understanding how to play the game | Yes | The program prints out game instructions and operating instructions in the console when the program is started. |
| 14 | Make the clock start after the initial click | Yes | The clock only starts after the grid has been generated after the initial click |
| 15 | Make the first click always not a Mine | Yes | The grid and numbers are generated after the first click so that it avoids the first click being a mine |

## User Feedback

I sent user feedback forms to the same users who answered the questionnaire in the analysis stage.

Graphical user interface, text, application, email

Description automatically generated

The answered forms can be seen in the User Feedback Forms Results Section under User Communication.

## Analysis of User Feedback

The users disliked the limitations of names. My program gives each player a unique ID which you identify them with but on the score board the names would show up the same. I could fix this by Adding the players ID to the leader board.

The users feedback suggested that I could improve the leader board by filtering the scores by different time frames e.g. Today, This Week, This Month and All time. This would make users more active as each day the leader board would reset so they could get to the top without having to beat the best of all time score.

A Continue Feature which allows the user to continue playing the game after they have hit a mine. It would simply reverse their last move by turning the tiles back on that were turned off during the move. This would be unfair to include in the games table as it would be cheating so it would skip adding it to the database.

My users complained that when the game is lost and the grid is displayed, you can’t see your flag placements to see where you went wrong to help you improve at the game. I could do this by displaying flags at all the tiles with flag set to on after the board is displayed.

Finally, they also suggested changing the menu interface by making the custom sizes sliders instead of having to type inputs in and some of the main menu buttons bigger as they can be tricky to select. This would make the game more visually appealing and makes it quicker and easier to change the sizes.

## Realistic Improvements

To extend this project and improve it I could:

* Give the user the option to save and reopen games by saving the time and creating a text file which stores all the information about the Grid and which tiles are on and which flags are on. This would allow me to re-open the game in the exact same conditions so the user could continue it without having to waste time. This would mainly be used for expert and custom games as they can take a while.
* To increase the players entertainment, I could add sound effects to make it more engaging. I could easily implement sounds using pygame, but I would need to create them or find some non-copyright free ones online
* Additionally, I could add more animation to the game to make it livelier and increase the visual appeal of the game. I could do this by creating more images for when the player removes tiles, places flags, wins/loses the game.
* I could also include more statistics such as Average Time, Efficiency, Clicks, and 3BV (Bechtel’s Board Benchmark Value). This is the smallest number of clicks to win the game, it changes from board to board. I could do this by using an algorithm that solves the board in every possible traversal and returns the smallest value of clicks possible.

# Referencing

<https://m.imdb.com/title/tt0285729/reviews> - Reviews on Minesweeper

<https://minesweeper.online/help/efficiency> - 3BV explained

First Mine Click

<https://www.reddit.com/r/AskReddit/comments/djnck/minesweeper_debate_can_you_can_lose_on_the_first/>

<http://imgur.com/GRB5H.jpg>

Rules of Minesweeper

<https://zyxyvy.wordpress.com/2012/08/11/the-rules-of-minesweeper/>

<https://minesweepergame.com/strategy/how-to-play-minesweeper.php>

Working with pygame

<https://www.pygame.org/docs/>

<https://inventwithpython.com/pygame/chapter2.html>

<https://www.adamsmith.haus/python/answers/how-to-sort-a-multidimensional-list-by-column-in-python> - 2D Array sorting

SQL

<https://www.mysql.com/downloads/>

<https://www.w3schools.com/sql/sql_datatypes.asp>

<https://stackoverflow.com/questions/29772337/python-mysql-connector-unread-result-found-when-using-fetchone>

[https://www.pixilart.com/draw#](https://www.pixilart.com/draw) - Pixel art maker

# Art Assets

I made all the Assets for the game using PowerPoint and an online pixel art maker (https://www.pixilart.com). Pixel art is simple and suits the style of game my Prospective users recommended.

Graphical user interface, diagram

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

## Menu Assets

Diagram

Description automatically generated

## Game Assets

A picture containing diagram

Description automatically generated

# Program Code

## Main Program

1. **import** time, pygame, random, os, sys, math # Imports
2. # constants
3. TileX **=** 20
4. TileY **=** 20
5. backGroundColour **=** (212, 212, 212) # Sets background colour, will be used a lot
7. Rules **=** '''
8. Rules:
9. The board is divided into cells, with mines randomly distributed. To win, you need to open all the cells.
10. The number on a cell shows the number of mines adjacent to it.
11. Using this information, you can determine cells that are safe, and cells that contain mines.
12. Cells suspected of being mines can be marked with a flag using the right mouse button.
14. Game Features:
15. The smiley face button is the restart button, this will bring you back to the main menu.
16. The blue number at the top is your score, this tells you what percentage of the board you have completed.
17. The red number below it is the time since you created the game.
18. If easy reset is on from the main menu settings, you can press middle mouse button to instantly restart with the same board settings
20. Menu Features:
21. You can create Custom games by pressing the custom difficulty and then entering in your values into the display buttons (press ENTER for each one after inputting).
22. The bar at the top allows you to sign in with your username then password.
23. The Create account buttons lets you create an account.
24. When you have signed into your account you can view your match history and your account statistics
25. '''
27. print(Rules) # Displays Rules to the User in case they need them
29. #load modules
30. sys.path.insert(0, '/Users/sovie/Documents/Minesweeper Final/Modules')
31. **from** Data\_Base **import** AddToTable, SelectFromTable, clearAllFromTable, LogIn, CheckAvailability
32. **from** Load\_Images **import** Load, Transform, loadHUDImage
33. **from** Menu\_Procedures **import** setUpMenu, CreateButtons, FetchInputText, LeaderBoard, CreateNewLogIn, AccountStatstics
34. **from** Game\_Procedures **import** DisplayAll, RemoveTiles, AddFlag, RemoveAdjacentTiles, AddExtraNumberTiles
35. ListImages **=** Load() # Loads images
36. Images **=** Transform(ListImages, TileX, TileY) # Transforms Images to correct dimensions
38. **class** Board():
39. # Class that generates a Binary Bit Map Of a Minesweeper Board, Also stores Time, Score and Name
40. **def** \_\_init\_\_(self, boardSize, Mines):
41. # Passed In Attributes
42. self.\_boardSize **=** boardSize
43. self.\_Mines **=** Mines
44. self.\_startTile **=** []
45. # Starting Empty List Attributes, Used to store information about the game: Mines, Flags, Bomb Alerts
46. self.\_Grid **=** []
47. self.\_RecordedMinePositions **=** []
48. **def** \_createGrid(self):
49. # Creates A Empty Binary Bit Map which will be Used to work with map, Reversed [DEPTH FROM 0, DISTANCE FROM 0]
50. **for** row **in** range(self.\_boardSize[1]):
51. list **=** []
52. **for** unit **in** range(self.\_boardSize[0]):
53. list.append('0')   # 0 on the Grid Symbolises an Empty Square, For now they all will All be Empty
54. self.\_Grid.append(list)
55. **def** \_placeMines(self):
56. # Randomly Places Mines on the Grid
57. MinePositions **=** [] # Records Positons of all Mines Created as a 2D Array
58. **if** self.\_Mines >**=** self.\_boardSize[0] **\*** self.\_boardSize[1]:
59. self.\_Mines **=** (self.\_boardSize[0] **\*** self.\_boardSize[1] ) **-** 1
60. **for** i **in** range(self.\_Mines): # Loops for the amount of Mines Selected
61. MineNotPlaced **=** True
62. **while** MineNotPlaced: # Continues Until Mine is Placed
63. x **=** random.randint(0, self.\_boardSize[0] **-** 1) # Gets a Random Number between 0 and the BoardWidth
64. y **=** random.randint(0, self.\_boardSize[1] **-** 1) # Gets a Random Number between 0 and the BoardHeight
65. position **=** [x, y]
66. **if** position **in** MinePositions **or** position **==** self.\_startTile: # Makes Sure Mines arent Placed in same Place Twice
67. **pass**
68. **else**:
69. MineNotPlaced **=** False
70. MinePositions.append(position)
71. **for** i **in** range(self.\_Mines): # Updates The Grid with each Mines Position
72. MinePosition **=** MinePositions[i]
73. MinePositionY **=** MinePosition[0]
74. MinePositionX **=** MinePosition[1]
75. self.\_Grid[MinePositionX][MinePositionY] **=** 'X' # Displays them on Grid as X, Graph of Grid X/Y is inverted (Grid[Y][X])
76. self.\_RecordedMinePositions **=** MinePositions # Records them for Later Use
77. **def** \_bombAlerts(self):
78. # Adds/Increment Numbers on The Grid Depending on how Many Mines surround it
79. **for** i **in** range(len(self.\_RecordedMinePositions)): # Loops For the Amount Of Mines Created
80. MinePosition **=** self.\_RecordedMinePositions[i]
81. MinePositionY **=** MinePosition[1]
82. MinePositionX **=** MinePosition[0]
83. **if** MinePositionX > 0: # Checks Tile to the Left
84. **if** self.\_Grid[MinePositionY][MinePositionX **-** 1] !**=** 'X': # If The left tile isnt a bomb
85. num **=** self.\_Grid[MinePositionY][MinePositionX **-** 1] # Gets the Number, if its the first iteration it will be 0, otherwise it could be 1-7
86. newNum **=** str(int(num) **+** 1) # Increments Number by 1
87. self.\_Grid[MinePositionY][MinePositionX **-** 1] **=** newNum # Updates NewNumber to Grid
88. **if** MinePositionX < self.\_boardSize[0] **-** 1: # Same For Tile Right of Mine
89. **if** self.\_Grid[MinePositionY][MinePositionX **+** 1] !**=** 'X':
90. num **=** self.\_Grid[MinePositionY][MinePositionX **+** 1]
91. newNum **=** str(int(num) **+** 1)
92. self.\_Grid[MinePositionY][MinePositionX **+** 1] **=** newNum
93. **if** MinePositionY > 0: # Same For Tile Above Mine
94. **if** self.\_Grid[MinePositionY **-** 1][MinePositionX] !**=** 'X':
95. num **=** self.\_Grid[MinePositionY **-** 1][MinePositionX]
96. newNum **=** str(int(num) **+** 1)
97. self.\_Grid[MinePositionY **-** 1][MinePositionX] **=** newNum
98. **if** MinePositionY < self.\_boardSize[1] **-** 1: # Same For Tile Below Mine
99. **if** self.\_Grid[MinePositionY **+** 1][MinePositionX] !**=** 'X':
100. num **=** self.\_Grid[MinePositionY **+** 1][MinePositionX]
101. newNum **=** str(int(num) **+** 1)
102. self.\_Grid[MinePositionY **+** 1][MinePositionX] **=** newNum
103. **if** MinePositionX < self.\_boardSize[0] **-** 1 **and** MinePositionY < self.\_boardSize[1] **-** 1: # Same For Tile Bottom Right of Mine
104. **if** self.\_Grid[MinePositionY **+** 1][MinePositionX **+** 1] !**=** 'X':
105. num **=** self.\_Grid[MinePositionY **+** 1][MinePositionX **+** 1]
106. newNum **=** str(int(num) **+** 1)
107. self.\_Grid[MinePositionY **+** 1][MinePositionX **+** 1] **=** newNum
108. **if** MinePositionX > 0 **and** MinePositionY < self.\_boardSize[1] **-** 1: # Same For Tile Bottom Left of Mine
109. **if** self.\_Grid[MinePositionY **+** 1][MinePositionX **-** 1] !**=** 'X':
110. num **=** self.\_Grid[MinePositionY **+** 1][MinePositionX **-** 1]
111. newNum **=** str(int(num) **+** 1)
112. self.\_Grid[MinePositionY **+** 1][MinePositionX **-** 1] **=** newNum
113. **if** MinePositionY > 0 **and** MinePositionX < self.\_boardSize[0] **-** 1: # Same For Tile Top Right of Mine
114. **if** self.\_Grid[MinePositionY **-** 1][MinePositionX **+** 1] !**=** 'X':
115. num **=** self.\_Grid[MinePositionY **-** 1][MinePositionX **+** 1]
116. newNum **=** str(int(num) **+** 1)
117. self.\_Grid[MinePositionY **-** 1][MinePositionX **+** 1] **=** newNum
118. **if** MinePositionY > 0 **and** MinePositionX > 0: # Same For Tile Top Left of Mine
119. **if** self.\_Grid[MinePositionY **-** 1][MinePositionX **-** 1] !**=** 'X':
120. num **=** self.\_Grid[MinePositionY **-** 1][MinePositionX **-** 1]
121. newNum **=** str(int(num) **+** 1)
122. self.\_Grid[MinePositionY **-** 1][MinePositionX **-** 1] **=** newNum
124. **class** HUD():
125. **def** \_\_init\_\_(self, boardSize, Mines, Difficulty, startingScore, ScreenSize):
126. # Constants Attributes
127. self.\_Difficulty **=** Difficulty
128. self.\_TotalTiles **=** boardSize[0] **\*** boardSize[1]
129. self.\_EmptyTiles **=** 0
130. self.\_Flags **=** []
131. self.\_Mines **=** Mines
132. # Static Attributes
133. self.\_Score **=** float(startingScore)
134. self.\_milliseconds **=** 0
135. self.\_secs **=** 0
136. self.\_mins **=** 0
137. self.\_ScreenWidth **=** ScreenSize[0]
138. self.\_ScreenHeight **=** ScreenSize[1]
140. **def** \_DisplayClock(self, started, GameScreen, BackGroundImg):
141. # Displays The Clock Digital (Minutes, Seconds)
142. pygame.font.init() # Initalises Font
143. **if** started **==** True:
144. self.\_milliseconds **+=** 1 # Increments Miliseconds Each time the clocks Ticks (100/s)
145. **if** self.\_milliseconds **==** 100: # 10 Miliseconds in a Second
146. self.\_secs **+=** 1
147. self.\_milliseconds **=** 0 # Resets
148. **if** self.\_secs **==** 60: # Converts Seconds to Minutes
149. self.\_mins **+=** 1
150. self.\_secs **=** 0
151. text **=** str(self.\_mins) **+**':'**+** str(self.\_secs)
152. Font **=** pygame.font.Font('freesansbold.ttf', 25)
153. Text **=** Font.render(text, True, (255, 0, 0)) # Renders Font
154. GameScreen.blit(BackGroundImg, (self.\_ScreenWidth **-** 100, 70))
155. GameScreen.blit(Text, (self.\_ScreenWidth **-** 95, 70)) # Displays The Text to the GameScreen
156. **def** \_DisplayAmountOfFlagsandBombs(self, GameScreen):
157. # Displays the Amount of Mines left and Amount of Flags on the Board at the Top of the GameScreen
158. Font **=** pygame.font.Font('freesansbold.ttf', 15)
159. FlagNum **=** Font.render(str(len(self.\_Flags)), True, (255, 0, 0))
160. MineNum **=** Font.render(str(self.\_Mines), True, (255, 0, 0))
161. GameScreen.blit(FlagNum, (self.\_ScreenWidth**-**70, 105))
162. GameScreen.blit(MineNum, (self.\_ScreenWidth**-**70, 130))
163. GameScreen.blit(Images['FlagImg'], (self.\_ScreenWidth**-**100, 100)) # Displays Images of Flag
164. GameScreen.blit(Images['BombMenuImg'], (self.\_ScreenWidth**-**100, 125))  # Displays Images of Bomb
165. **def** \_CalculateScore(self, Value):
166. # Calculates Score, Called For each tile that is taken of the board, the Score resembles the percentage of the board complete, when it is 100% that means all the non-Mine tiles have been removed
167. TilesRemoved **=** self.\_TotalTiles **-** Value # Value is the amount of remaning Tiles
168. self.\_Score **=** round((TilesRemoved **/** (self.\_TotalTiles **-** int(self.\_Mines)))**\***100, 1)
169. **def** \_DisplayScore(self, GameScreen, BackGroundImg):
170. # Displays Score on GameScreen, Had to be Seperate to Calculate Score otherwise each time its displayed it would Increment Score
171. score **=** str(self.\_Score)
172. Font **=** pygame.font.Font('freesansbold.ttf', 25)
173. score **=** Font.render(score, True, (0, 0, 255))
174. GameScreen.blit(BackGroundImg, (self.\_ScreenWidth **-** 100, 40))
175. GameScreen.blit(score, (self.\_ScreenWidth**-**100, 40))
176. **def** \_RecordScore(self, MainMenu):
177. # Called When game is Won
178. Time **=** (self.\_mins **\*** 60) **+** self.\_secs **+** self.\_milliseconds**/**100 # Calculates Time
179. **if** self.\_Difficulty !**=** 'Custom':
180. AddToTable(self.\_Difficulty, MainMenu.\_PlayersID, Time, self.\_Score) # Adds to the Database Table Games
181. **class** Menu():
182. # Class for the Menu Screen
183. **def** \_\_init\_\_(self, Settings, Images, Buttons, Screen):
184. self.\_Difficulty **=** 'Intermediate' # Sets default starting Difficulty
185. self.\_Settings **=** Settings
186. self.\_Images **=** Images
187. self.\_Buttons **=** Buttons
188. self.\_Screen **=** Screen
189. self.\_Name **=** 'Guest' # Sets Starting Default name as Guest
190. self.\_PlayersID **=** **-**1
191. pygame.font.init()
192. Font **=** pygame.font.Font('freesansbold.ttf', 15)
193. self.\_NameDisplay **=** Font.render(self.\_Name, True, (0, 0, 0)) # Renders Name
194. self.\_Texts **=** []
195. self.\_Mines **=** 40
196. self.\_Width **=** 16
197. self.\_Height **=** 16
198. **def** \_DifficultyToValues(self):
199. # Gets values of Mines and Width and Height depending on the Difficulty
200. **if** self.\_Difficulty **==** 'Beginner':
201. self.\_Mines **=** 10
202. self.\_Width **=** 10
203. self.\_Height **=** 10
204. **if** self.\_Difficulty **==** 'Intermediate':
205. self.\_Mines **=** 40
206. self.\_Width **=** 16
207. self.\_Height **=** 16
208. **if** self.\_Difficulty **==** 'Expert':
209. self.\_Mines **=** 99
210. self.\_Width **=** 16
211. self.\_Height **=** 30
212. **def** \_CreateTexts(self):
213. # Creates Texts to be Displayed on the Screen
214. pygame.font.init()
215. Font **=** pygame.font.Font('freesansbold.ttf', 15)
216. Text1 **=** Font.render('Mines:     '**+**str(self.\_Mines), True, (0, 0, 0))
217. Text2 **=** Font.render('Width:     '**+**str(self.\_Width), True, (0, 0, 0))
218. Text3 **=** Font.render('Height:    '**+**str(self.\_Height), True, (0, 0, 0))
219. Text4 **=** Font.render('Beginner', True, (0, 0, 0))
220. Text5 **=** Font.render('Intermediate', True, (0, 0, 0))
221. Text6 **=** Font.render('Expert', True, (0, 0, 0))
222. Text7 **=** Font.render('Custom', True, (0, 0, 0))
223. Text8 **=** Font.render('Tile LOCK', True, (0, 0, 0))
224. Text9 **=** Font.render('Easy Reset', True, (0, 0, 0))
225. Text10 **=** Font.render('Save Score', True, (0, 0, 0))
226. self.\_NameDisplay **=** Font.render(self.\_Name, True, (0, 0, 0))
227. self.\_Texts **=** [Text1, Text2, Text3, Text4, Text5, Text6, Text7, Text8, Text9, Text10] # Returns them as an Array for simplicity
228. **def** \_DisplayMenu(self):
229. # Displays Images
230. **for** i **in** range(len(self.\_Buttons)): # Draws all Buttons on Screen
231. self.\_Buttons[i].\_Draw(self.\_Screen)
232. # Updates Dots on the Screen
233. **if** self.\_Difficulty **==** 'Beginner':
234. self.\_Screen.blit(self.\_Images['DotImg'], (20, 50))
235. **if** self.\_Difficulty **==** 'Intermediate':
236. self.\_Screen.blit(self.\_Images['DotImg'], (20, 70))
237. **if** self.\_Difficulty **==** 'Expert':
238. self.\_Screen.blit(self.\_Images['DotImg'], (20, 90))
239. **if** self.\_Difficulty **==** 'Custom':
240. self.\_Screen.blit(self.\_Images['DotImg'], (20, 110))
241. # Displays All the Texts
242. self.\_Screen.blit(self.\_Texts[0], (150, 200))
243. self.\_Screen.blit(self.\_Texts[1], (10, 180))
244. self.\_Screen.blit(self.\_Texts[2], (10, 200))
245. j **=** 50
246. **for** i **in** range(3, 7):
247. self.\_Screen.blit(self.\_Texts[i], (40, j))
248. j **+=** 20
249. j **=** 50
250. **for** i **in** range(7, 10):
251. self.\_Screen.blit(self.\_Texts[i], (180, j))
252. j **+=** 20
253. self.\_Screen.blit(self.\_NameDisplay, (25, 15))
254. j **=** 50
255. **for** i **in** range(0, len(self.\_Settings)):
256. **if** self.\_Settings[i] **==** True:
257. self.\_Screen.blit(Images['CrossImg'], (150, j))
258. j**+=**20
259. **def** \_CheckForClicks(self):
260. # Checks which Buttons have been pressed
261. TileLock **=**  self.\_Settings[0]
262. EasyReset **=** self.\_Settings[1]
263. SaveScore **=** self.\_Settings[2]
264. LeaderBoardDisplayOn **=** False
265. Continue **=** True
266. CreatedAccount **=** False
267. **for** i **in** range(len(self.\_Buttons)):
268. **if** self.\_Buttons[i].\_checkForClick() **==** True:
269. time.sleep(0.2) # Delay so you dont instantly repress the button
270. **if** i **==** 0:
271. self.\_Difficulty **=** 'Beginner' # If the Beginner Button is pressed it sets the Difficulty to Beginner
272. **if** i **==** 1:
273. self.\_Difficulty **=** 'Intermediate'
274. **if** i **==** 2:
275. self.\_Difficulty **=** 'Expert'
276. **if** i **==** 3:
277. self.\_Difficulty **=** 'Custom'
278. **if** i **==** 4:
279. **if** TileLock **==** True: # Makes it act as a Button, If its already on and it is clicked it turns it off
280. TileLock **=** False
281. **else**:
282. TileLock **=** True
283. **if** i **==** 5:
284. **if** SaveScore **==** True:
285. SaveScore **=** False
286. **else**:
287. SaveScore **=** True
288. **if** i **==** 6:
289. **if** EasyReset **==** True:
290. EasyReset **=** False
291. **else**:
292. EasyReset **=** True
293. **if** i **==** 7: # Play Game
294. self.\_Screen.fill((255, 255, 255)) # Refreshes Screen
295. self.\_Screen.blit(self.\_Images['PlayButtonPressedImg'], (300, 60)) # Animation for button
296. pygame.display.update()
297. time.sleep(0.5)
298. Continue **=** False
299. **if** i **==** 8: # Quits Game
300. self.\_Screen.fill((255, 255, 255))
301. self.\_Screen.blit(self.\_Images['QuitButtonPressedImg'], (300, 160))
302. pygame.display.update()
303. time.sleep(0.2)
304. pygame.quit()
305. sys.exit()
306. **if** i **==** 12:
307. NameDisplay **=** ''
308. # Name can only have 15 Characters
309. self.\_Name **=** FetchInputText('Both', 15, [25, 15], 'Name', self.\_Texts, self.\_Screen, self.\_Images, self.\_NameDisplay, self.\_Buttons) # Fetches User input for Name
310. Exists **=** CheckAvailability(self.\_Name)
311. **if** Exists **==** False: # If the name Exists then there is an account
312. Password **=** FetchInputText('Both', 20, [25, 18], 'Password', self.\_Texts, self.\_Screen, self.\_Images, self.\_NameDisplay, self.\_Buttons) # Fetches User input for Password
313. PlayerID, Username, CorrectDetails **=** LogIn(self.\_Name, Password) # trys to log into account
314. self.\_PlayersID **=** PlayerID
315. **if** CorrectDetails **==** False: # if they get the username and password wrong it resets the name back to guest
316. print('Password incorrect')
317. self.\_Name **=** 'Guest'
318. **else**:
319. print('Name doesnt exist') # if there is no account it resets the name
320. self.\_Name **=** 'Guest'
322. **if** i **==** 13: # Opens up leaderboard if leaderboard button pressed
323. self.\_Screen.fill((255, 255, 255))
324. self.\_Screen.blit(self.\_Images['LeaderBoardPressedImg'], (300, 110))
325. pygame.display.update()
326. time.sleep(0.5)
327. LeaderBoardDisplayOn **=** True
328. **if** i **==** 14:
329. CreateNewLogIn(Images, Button) # Opens new account menu
330. pygame.quit()
331. CreatedAccount **=** True
332. self.\_Settings **=** [TileLock, EasyReset, SaveScore] # Simplifies all the Settings into one Array
333. **return** Continue, LeaderBoardDisplayOn, CreatedAccount
334. **def** GetCustomSizes(self):
335. # Gets Custom size for size for mines, height and width
336. Values **=** [50, 10] # Sets max and min size for both Height and Width
337. **if** self.\_Buttons[9].\_checkForClick() **==** True:
338. self.\_Height **=** FetchInputText('Digits', 2, [80, 203], 'Height', self.\_Texts, self.\_Screen, self.\_Images, self.\_NameDisplay, self.\_Buttons)
339. **if** len(self.\_Height) > 0: # If they have inputed something:
340. **if** int(self.\_Height) > 50 **or** int(self.\_Height)  < 10:
341. self.\_Height **=** min(Values, key**=lambda** x:abs(x**-**int(self.\_Height))) # Rounds it to closest value, either max or min, if they enter a value that is not acceptapted within the bounds
342. **else**:
343. self.\_Height **=** 10
344. **if** self.\_Buttons[10].\_checkForClick() **==** True:
345. self.\_Width **=** FetchInputText('Digits', 2, [80, 179], 'Width', self.\_Texts, self.\_Screen, self.\_Images, self.\_NameDisplay, self.\_Buttons)
346. **if** len(self.\_Width) > 0:
347. **if** int(self.\_Width) > 50 **or** int(self.\_Width)  < 8:
348. self.\_Width **=** min(Values, key**=lambda** x:abs(x**-**int(self.\_Width)))
349. **else**:
350. self.\_Width **=** 8
351. **if** self.\_Buttons[11].\_checkForClick() **==** True:
352. self.\_Mines **=** FetchInputText('Digits', 3, [210, 200], 'Mines', self.\_Texts, self.\_Screen, self.\_Images, self.\_NameDisplay, self.\_Buttons)
353. **if** len(self.\_Mines) **==** 0: # if its empty make it 0
354. self.\_Mines **=** 0
355. **class** Button():
356. # Simple Button Class Needed for user input in User Interface using Pygame
357. **def** \_\_init\_\_(self, Xpos, Ypos, image, scale):
358. self.\_YPosition **=** Ypos
359. self.\_XPosition **=** Xpos
360. self.\_scale **=** float(scale)
361. self.\_Unscaledimage **=** image
362. width **=** image.get\_width() # Gets Image Width
363. height **=** image.get\_height() # Gets Image Height
364. self.\_image **=** pygame.transform.scale(self.\_Unscaledimage, (int(width **\*** self.\_scale), int(height **\*** self.\_scale))) # Transform Passed in Image to new Dimensions Based on the Scale, Doesnt Change Image Shape Only the Size on Screen
365. self.\_rect **=** self.\_image.get\_rect() # Defines the Rectangle (hitbox) of Image, Used in collisions with Mouse Position
366. self.\_rect.topleft **=** (Xpos, Ypos) # When Displaying the Image it will Display from the Top left of the Rectangle where the coordinates specify
367. **def** \_checkForClick(self):
368. # Very Simple Method that checks if the user clicks it
369. Clicked **=** False
370. MousePosition **=** pygame.mouse.get\_pos() # Gets Position of Mouse
371. **if** self.\_rect.collidepoint(MousePosition): # Checks if the Mouse collides with the Objects ( Button ) Rect ( Hitbox)
372. **if** pygame.mouse.get\_pressed()[0] **==** 1: # Checks that if the Left Mouse Button is pressed [0] = Left [1] = Middle [2] = Right
373. Clicked **=** True
374. **return** Clicked
375. **def** \_Draw(self, Screen):
376. # Draws Button to Screen, Screen is passed in because there are Buttons need to appear on the main menu and on the gamescreen
377. Screen.blit(self.\_image, (self.\_rect.x, self.\_rect.y))
379. **class** Tile(Button):
380. # A Tile Inherits from Button class and uses shared attributes
381. **def** \_\_init\_\_(self, x, y, image, scale, TileNumber, FlagOn, TileOn):
382. super().\_\_init\_\_(x, y, image, scale)
383. # Each Tile Has Unique Identifier
384. self.\_TileNumber **=** TileNumber
385. self.\_FlagOn **=** FlagOn
386. self.\_TileOn **=** TileOn
387. **def** \_DisplayFlags(self, GameScreen):
388. # If Flag is Set to True it means a Flag should be Displayed on the Tile
389. **if** self.\_FlagOn **==** True:
390. GameScreen.blit(Images['FlagImg'], (self.\_XPosition, self.\_YPosition))
391. **def** \_DisplayTiles(self, GameScreen):
392. # If the tileOn is set to True it means the tile hasnt be clicked ( or adjacent empty tile hasnt been) so it should be displayed
393. **if** self.\_TileOn **==** True:
394. GameScreen.blit(self.\_image, (self.\_XPosition, self.\_YPosition))
395. **else**: # If the Tile is off, the flag must also be off
396. self.\_FlagOn **=** False

399. **def** AccountStatsTextDisplay(GlobalStats, PlayerStats, AcountStatisticsMenu, PlayerName, Difficulty):
400. pygame.font.init()
401. Font **=** pygame.font.Font('freesansbold.ttf', 9)
402. # Prints Texts in different colours to show which option is selected
403. **if** Difficulty **==** 'Beginner':
404. BeginnerText **=** Font.render('Beginner', True, (0, 200, 0))
405. **else**:
406. BeginnerText **=** Font.render('Beginner', True, (0, 0, 0))
407. **if** Difficulty **==** 'Intermediate':
408. IntermediateText **=** Font.render('Intermediate', True, (0, 200, 0))
409. **else**:
410. IntermediateText **=** Font.render('Intermediate', True, (0, 0, 0))
412. **if** Difficulty **==** 'Expert':
413. ExpertText **=** Font.render('Expert', True, (0, 200, 0))
414. **else**:
415. ExpertText **=** Font.render('Expert', True, (0, 0, 0))
417. AcountStatisticsMenu.blit(BeginnerText, (165, 15))
418. AcountStatisticsMenu.blit(IntermediateText, (245, 15))
419. AcountStatisticsMenu.blit(ExpertText, (325, 15))
421. # Organises Data for Player and Global stats
422. GlobalTotalGamesPlayed **=** Font.render('Global Games Played '**+**str(GlobalStats[0]), True, (0, 0, 0))
423. GlobalTotalGamesWon **=** Font.render('Global Games Won: '**+**str(GlobalStats[1]), True, (0, 0, 0))
424. GlobalTotalGamesLost **=** Font.render('Global Games Lost: '**+**str(GlobalStats[0] **-** GlobalStats[1]), True, (0, 0, 0))
425. Font **=** pygame.font.Font('freesansbold.ttf', 15)
426. PlayerName **=** Font.render('PlayerName: '**+**PlayerName, True, (0, 0, 0))
427. AcountStatisticsMenu.blit(GlobalTotalGamesPlayed, (395, 5))
428. AcountStatisticsMenu.blit(GlobalTotalGamesWon, (395, 15))
429. AcountStatisticsMenu.blit(GlobalTotalGamesLost, (395, 25))
430. AcountStatisticsMenu.blit(PlayerName, (20, 10))
432. Font **=** pygame.font.Font('freesansbold.ttf', 12)
433. PlayerTotalGamesPlayed **=** Font.render('Total Games Played: '**+**str(PlayerStats[2]), True, (0, 0, 0))
434. PlayerTotalGamesWon **=** Font.render('Games Won: '**+**str(PlayerStats[0]), True, (0, 0, 0))
435. PlayerTotalGamesLost **=** Font.render('Games Lost: '**+**str(PlayerStats[1]), True, (0, 0, 0))
436. AccountCreated **=** str(PlayerStats[3])
437. AccountCreated **=** AccountCreated.replace('(datetime.datetime(', '')
438. AccountCreated **=** AccountCreated.replace('),)', '')
439. PlayerAccountCreated **=** Font.render('Account Created: '**+**str(AccountCreated), True, (0, 0, 0))
440. **if** PlayerStats[2] > 0 **and** PlayerStats[1] > 0:
441. Ratio **=** round((PlayerStats[0]**/**PlayerStats[2]**\***100), 1)
442. PlayerWinRatio **=** Font.render('Win:Loss Ratio: '**+**str(Ratio), True, (0, 0, 0))
443. AcountStatisticsMenu.blit(PlayerWinRatio, (20, 210))
445. # Displays info on screen
446. AcountStatisticsMenu.blit(PlayerTotalGamesPlayed, (20, 50))
447. AcountStatisticsMenu.blit(PlayerTotalGamesWon, (20, 90))
448. AcountStatisticsMenu.blit(PlayerTotalGamesLost, (20, 130))
449. AcountStatisticsMenu.blit(PlayerAccountCreated, (20, 170))
451. # Draw a pie chart of Wins and Losses
453. **if** PlayerStats[2] > 0 **and** PlayerStats[1] > 0:
455. CentreX, CentreY, Radius **=** 390, 190, 75 # sets raidus and centre of pi chart
456. pygame.draw.circle(AcountStatisticsMenu, (255, 0, 0), (CentreX, CentreY), Radius) # draws RED circle at centre coorddinates
457. Ratio **=** Ratio **/** 100
458. angle **=** int(round(360 **\*** Ratio, 0)) # calculates angle, How much of the circle represents the win/loss percentage ( circle = 360 degrees)
459. p **=** [(CentreX, CentreY)] # sets centre
460. **for** n **in** range(0, angle):
461. x **=** CentreX **+** int(Radius**\***math.cos(n**\***math.pi**/**180)) # figures out X coordinate
462. y **=** CentreY **+** int(Radius**\***math.sin(n**\***math.pi**/**180)) # figures out X coordinate
463. p.append((x, y))
464. p.append((CentreX, CentreY))
465. **if** len(p) > 2: # points argument must contain more than 2 points
466. pygame.draw.polygon(AcountStatisticsMenu, (0, 255, 0), p) # draws several polynomials FROM centre so it looks like a Sector
468. **def** LeaderBoardTextDisplay(LeaderBoardButtons, LeaderBoardSettings, LeaderBoardDisplay):
469. # Displays Info on the LeaderBoard
470. **for** i **in** range(len(LeaderBoardButtons)):
471. LeaderBoardButtons[i].\_Draw(LeaderBoardDisplay) # Draws all leader board buttons
472. **if** LeaderBoardButtons[0].\_checkForClick() **==** True: # Checks for clicks for each of the buttons and then carries out function
473. LeaderBoardSettings[0] **=** 'Beginner'
474. **if** LeaderBoardButtons[1].\_checkForClick() **==** True:
475. LeaderBoardSettings[0] **=** 'Intermediate'
476. **if** LeaderBoardButtons[2].\_checkForClick() **==** True:
477. LeaderBoardSettings[0] **=** 'Expert'
478. **if** LeaderBoardButtons[3].\_checkForClick() **==** True:
479. LeaderBoardSettings[1] **=** 'Time'
480. **if** LeaderBoardButtons[4].\_checkForClick() **==** True:
481. LeaderBoardSettings[1] **=** 'Date'
482. Font **=** pygame.font.Font('freesansbold.ttf', 10)   # This chunck of code makes the text green if the button has been clicked or is default button
483. **if** LeaderBoardSettings[0] **==** 'Beginner':
484. BeginnerText **=** Font.render('Beginner', True, (0, 200, 0))
485. **else**:
486. BeginnerText **=** Font.render('Beginner', True, (0, 0, 0))
487. **if** LeaderBoardSettings[0] **==** 'Intermediate':
488. IntermediateText **=** Font.render('Intermediate', True, (0, 200, 0))
489. **else**:
490. IntermediateText **=** Font.render('Intermediate', True, (0, 0, 0))
491. **if** LeaderBoardSettings[0] **==** 'Expert':
492. ExpertText **=** Font.render('Expert', True, (0, 200, 0))
493. **else**:
494. ExpertText **=** Font.render('Expert', True, (0, 0, 0))
496. **if** LeaderBoardSettings[1] **==** 'Time':
497. TimeText **=** Font.render('Time', True, (0, 200, 0))
498. **else**:
499. TimeText **=** Font.render('Time', True, (0, 0, 0))
501. **if** LeaderBoardSettings[1] **==** 'Date':
502. DateText **=** Font.render('Date', True, (0, 200, 0))
503. **else**:
504. DateText **=** Font.render('Date', True, (0, 0, 0))
505. LeaderBoardDisplay.blit(TimeText, (283, 15)) # Displays all Texts
506. LeaderBoardDisplay.blit(DateText, (363, 15))
507. LeaderBoardDisplay.blit(BeginnerText, (25, 15))
508. LeaderBoardDisplay.blit(IntermediateText, (103, 15))
509. LeaderBoardDisplay.blit(ExpertText, (190, 15))
511. **def** Main(Images, ListImages, TileX, TileY):
512. Games **=** []
513. x **=** **-**1
514. MainGameLoop **=** True
515. QuickReset **=** False
516. MainMenuLoop **=** True
517. Buttons, LeaderBoardButtons, AccountStatisticButtons **=** CreateButtons(Images, Button) # Creates Main Menu Button objects, only needs to be done once
518. Settings **=** [True, True, True] # Sets the default settings outside the Menu Loop so it doesnt saves the changed settings each time you restart the game
519. Continue **=** True
520. MenuScreen **=** setUpMenu()
521. MainMenu **=** Menu(Settings, Images, Buttons, MenuScreen)
522. AcountStatisticsButton **=** Button(300, 12, Images['AccountStatsImg'], 0.1) # Cant be in create Buttons procedure as that displays all the buttons as well
523. GameHistoryButton **=** Button(20, 250, Images['GameHistroyImg'], 0.14)
524. **while** MainGameLoop: # Main game Loop
525. x **=** x **+** 1
526. pygame.quit()
527. MenuScreen **=** setUpMenu()
528. MainMenu.\_Screen **=** MenuScreen
529. **if** QuickReset **==** True: # Resets the Quick Reset
530. MainMenuLoop **=** False # Skips main menu
531. QuickReset **=** False
532. **else**:
533. MainMenuLoop **=** True
534. **while** MainMenuLoop: # Main Menu Loop
535. MainMenu.\_DifficultyToValues() # Converts the Difficulty into Values ( Mines, Height and Width )
536. MainMenu.\_CreateTexts() # Creates/Updates Texts
537. MenuScreen.fill((255, 255, 255)) # Refreshes Screen
538. MainMenu.\_DisplayMenu() # Displays Things on the Menu
539. Continue, LeaderBoardDisplayOn, CreatedAccount  **=** MainMenu.\_CheckForClicks()
540. **if** CreatedAccount **==** True:
541. MenuScreen **=** setUpMenu()
542. MainMenu.\_Screen **=** MenuScreen # Redefines Menu
543. **if** MainMenu.\_Difficulty **==** 'Custom':
544. MainMenu.GetCustomSizes() # if the difficulty is custom, it calls this procedure which checks if any custom buttons are pressed
545. **if** MainMenu.\_Name !**=** 'Guest':
546. AcountStatisticsButton.\_Draw(MainMenu.\_Screen)
547. **if** AcountStatisticsButton.\_checkForClick():
548. AcountStatisticsMenu **=** pygame.display.set\_mode((550, 300))
549. AcountStatisticsLoop **=** True
550. AccountStatisticSetting **=** 'Beginner'
551. **while** AcountStatisticsLoop:
552. AcountStatisticsMenu.fill((255, 255, 255)) # refreshes screen
553. **for** i **in** range(len(AccountStatisticButtons)): # Draws all Statistic menu buttons
554. AccountStatisticButtons[i].\_Draw(AcountStatisticsMenu) # Draws all leader board buttons
555. **if** AccountStatisticButtons[0].\_checkForClick() **==** True: # Checks for clicks for each of the buttons and then carries out function
556. AccountStatisticSetting **=** 'Beginner'
557. **if** AccountStatisticButtons[1].\_checkForClick() **==** True:
558. AccountStatisticSetting **=** 'Intermediate'
559. **if** AccountStatisticButtons[2].\_checkForClick() **==** True:
560. AccountStatisticSetting **=** 'Expert'
561. pygame.display.set\_caption('Account Statistics') # Window Caption
562. GameHistoryLoop **=** False
563. GameHistoryButton.\_Draw(AcountStatisticsMenu)
564. GlobalStats, PlayerStats **=** AccountStatstics(MainMenu.\_PlayersID, AccountStatisticSetting)
565. AccountStatsTextDisplay(GlobalStats, PlayerStats, AcountStatisticsMenu, MainMenu.\_Name, AccountStatisticSetting)
566. **if** GameHistoryButton.\_checkForClick() **==** True:
567. GameHistoryLoop **=** True
568. pygame.display.set\_caption('GameHistory') # Changes Window Caption
569. **while** GameHistoryLoop:
570. AcountStatisticsMenu.fill((255, 255, 255))
571. pygame.font.init()
572. Font **=** pygame.font.Font('freesansbold.ttf', 12)
573. done **=** False
574. GameHistory **=** PlayerStats[4]
575. GameHistory **=** GameHistory.replace("')", "")
576. # Organise One long string into List
577. Data **=** []
578. i **=** 0
579. Loops **=** GameHistory.count("datetime.datetime(") # there is 1 of these per row
580. **if** Loops > 0:
581. **for** j **in** range(Loops**+**1):
582. Temp **=** ''
583. i **=** i **+** 2
584. **while** GameHistory[i] !**=** ')' **and** GameHistory[i**+**1] !**=** ')': # loops through the long singular string and seperates it when it finds '))'
585. Temp **=** Temp **+** str(GameHistory[i])
586. i **=** i **+** 1
587. Temp **=** Temp.replace("Decimal('", "")
588. Temp **=** Temp.replace("datetime.datetime", "") # Clears unneeded text
589. Temp **=** Temp.replace("(", "")
590. Data.append(Temp)
591. filter(None, Data)
592. filter(None, Data)
593. **for** i **in** range(len(Data)):
594. Text **=** Font.render(str(Data[i]), True, (0, 0, 0))
595. AcountStatisticsMenu.blit(Text, (10, (20**\***i)**+**10)) # displays data on new lines on display
596. **for** event **in** pygame.event.get():
597. **if** event.type **==** pygame.QUIT: # if the player quits the menu via the top right cross
598. GameHistoryLoop **=** False
599. pygame.display.update()
600. pygame.display.update()
601. **for** event **in** pygame.event.get():
602. **if** event.type **==** pygame.QUIT: # if the player quits the menu via the top right cross
603. AcountStatisticsLoop **=** False
604. pygame.quit() # Closes the window
605. MenuScreen **=** setUpMenu() # Sets up Menu again
606. MainMenu.\_Screen **=** MenuScreen # Redefines Menu
607. **if** LeaderBoardDisplayOn **==** True:
608. LeaderBoardDisplay **=** pygame.display.set\_mode((450, 600)) # sets up leaderboard display
609. LeaderBoardSettings **=** ['Beginner', 'Time'] # Default starting Settings
610. pygame.display.set\_caption('LeaderBoard') # Window Caption
611. **while** LeaderBoardDisplayOn **==** True: # leaderboard Loop
612. LeaderBoardDisplay.fill((255, 255, 255))
613. LeaderBoardTextDisplay(LeaderBoardButtons, LeaderBoardSettings, LeaderBoardDisplay) # Deals with buttons and Text, also displays them on the menu
614. Data **=** SelectFromTable(LeaderBoardSettings[0], LeaderBoardSettings[1]) # Which Difficulty and what to Order them by ( time or Date played )
615. LeaderBoard(LeaderBoardSettings, Data, LeaderBoardDisplay) # Displays The leaderboard on the Leaderboard Display, Allows them to order by last played or how fast the player won (main/default) and which Difficulty
616. pygame.display.update()
617. **for** event **in** pygame.event.get():
618. **if** event.type **==** pygame.QUIT: # if the player quits the menu via the top right cross
619. LeaderBoardDisplayOn **=** False
620. pygame.quit() # Closes the window
621. MenuScreen **=** setUpMenu() # Sets up Menu again
622. MainMenu.\_Screen **=** MenuScreen # Redefines Menu
623. **if** Continue **==** False:
624. MainMenuLoop **=** False
625. **for** event **in** pygame.event.get():
626. **if** event.type **==** pygame.QUIT:
627. pygame.quit()
628. sys.exit()
629. pygame.display.update()
630. MainMenu.\_DifficultyToValues() # Converts back from the settings Array into the individual variables
631. FlagLock **=** MainMenu.\_Settings[0]
632. EasyReset **=** MainMenu.\_Settings[1]
633. SaveScore **=** MainMenu.\_Settings[2]
634. pygame.quit() # Closes Menu Screen
635. MenuTick **=** pygame.time.get\_ticks() # Gets how long you spent in the Menu to make the clock work
636. boardSize **=** (int(MainMenu.\_Width), int(MainMenu.\_Height))
637. Game **=** Board(boardSize, int(MainMenu.\_Mines)) # creates Game object, this will be used for creating and setting up the game
638. Games.append(Game)
639. Games[x].\_createGrid() # creates the empty grid for the board
640. # Setting up Game Screen Based of how many Tiles there are
641. ScreenWidth **=** ((Games[x].\_boardSize[0]) **\*** 24) **+** 80
642. ScreenHeight **=** ((Games[x].\_boardSize[1]) **\*** 21) **+** 5
643. ScreenSize **=** (ScreenWidth, ScreenHeight)
644. HUDImg, BackGroundImg **=** loadHUDImage(ScreenWidth, ScreenHeight, MainMenu.\_Difficulty) # Loads images for the HUD
645. GameScreen **=** pygame.display.set\_mode(ScreenSize) # Defines the GameScreen
646. **if** MainMenu.\_Difficulty **==** 'Beginner':
647. ScreenWidth **=** ScreenWidth **+** 10
648. GameHud **=** HUD(boardSize, MainMenu.\_Mines, MainMenu.\_Difficulty, 0, (ScreenWidth, ScreenHeight)) # Creates the HUD, this is used for display things on the GameScreen to the player e.g clock, score
649. RestartButton **=** Button((ScreenWidth**-**100), 150, Images['RestartImg'], 0.05) # Creates Restart Button
650. GameScreen.fill(backGroundColour) # Back Ground Colour
651. pygame.display.set\_caption('Minesweeper!') # Window Caption
652. AllTiles **=** []
653. **for** i **in** range(Games[x].\_boardSize[0]):
654. **for** j **in** range(Games[x].\_boardSize[1]):
655. AllTiles.append(Tile((TileX**\***i)**+**5, (TileY**\***j)**+**10,Images['TileImg'],1, (i, j), False, True)) # Creates and adds Tile objects (which act as buttons) to a list
656. Clock **=** pygame.time.Clock() # Creates Clock
657. GameLoop **=** True
658. started **=** False # Doesnt start timer until first click
659. Clicks **=** 0 # Sets amount of clicks to 0, this will be used to create the mine positions after the first click to avoid the first click being a mine
660. **while** GameLoop:
661. GameScreen.blit(HUDImg, ((boardSize[0]**\***21), 10)) # Displays HUD background
662. AddFlag(AllTiles, GameHud) # Checks if the player adds/removes a Flag
663. Status, TileNum, Clicks, FirstClick **=** RemoveTiles(AllTiles, FlagLock, GameHud, Games[x], Clicks, Button) # Removes a Tile based of where the user clicks
664. **if** FirstClick **==** True: # if the its the users first click it generates the mines avoiding the starting tile
665. Games[x].\_startTile **=** TileNum
666. Games[x].\_placeMines() # places the mines on the grid
667. Games[x].\_bombAlerts() # increments the numbers around every mine on the grid#
668. started **=** True # starts clock
669. **if** Status **==** False: # if the user clicked a Tile with a bomb under it
670. # Loose Condition
671. **if** SaveScore **==** True:
672. GameHud.\_RecordScore(MainMenu)
673. GameScreen.fill((255, 0, 0)) # Flashes Red
674. pygame.display.update()
675. time.sleep(0.3)
676. GameScreen.fill(backGroundColour) # Back Ground Colour
677. GameScreen.blit(HUDImg, ((boardSize[0]**\***21), 10))
678. GameScreen.blit(Images['LooseImg'], ((boardSize[0]**\***21)**+**15, 150)) # Displays Loose Image on the HUD
679. DisplayAll(Game, GameScreen, Images, TileX, TileY) # Displays the grid under the Tiles
680. GameHud.\_DisplayClock(started, GameScreen, BackGroundImg) # shows the player the time they got
681. GameHud.\_DisplayScore(GameScreen, BackGroundImg) # shows the player the score they got
682. GameHud.\_DisplayAmountOfFlagsandBombs(GameScreen) # shows the player how many flags they had placed and how many bombs there were
683. pygame.display.update()
684. GameLoop **=** False
685. NextGameLoop **=** True
686. **while** NextGameLoop:
687. **for** event **in** pygame.event.get():
688. **if** event.type **==** pygame.KEYDOWN: # if they press a button on the keyboard it restarts
689. NextGameLoop **=** False
690. **elif** pygame.mouse.get\_pressed()[1] **==** True: # if they press the middle mouse button it restarts without going to the main menu and keeps the same settings
691. NextGameLoop **=** False
692. QuickReset **=** True
693. **elif** event.type **==** pygame.QUIT: # if they press the red cross it closes the program
694. pygame.quit()
695. quit() # quits whole program
696. **break**
697. **else**:
698. RecordedEmptyTiles **=** RemoveAdjacentTiles(Game.\_Grid, AllTiles, TileNum, Games[x], GameHud) # Removes adjacent empty Tiles
699. RemainingTiles **=** []
700. **for** i **in** range(len(AllTiles)):
701. **if** AllTiles[i].\_TileOn **==** True:
702. RemainingTiles.append([AllTiles[i].\_TileNumber[0], AllTiles[i].\_TileNumber[1]]) # Adds all Tiles that are on into an array
703. Value **=** len(RemainingTiles)
704. GameHud.\_CalculateScore(Value) # calculates score (percent completed) based on how many tiles are left to click
705. # Order Both Arrays and Check if they are the same
707. Games[x].\_RecordedMinePositions **=** sorted(Game.\_RecordedMinePositions)
708. RemainingTiles **=** sorted(RemainingTiles)
710. **if** Game.\_RecordedMinePositions **==** RemainingTiles: # if they are the same, it means the only remaining tiles that are on are the Mines so the player wins
711. # Win Condition
712. **if** SaveScore **==** True:
713. GameHud.\_RecordScore(MainMenu)
714. GameScreen.fill((0, 255, 0)) # Flashes Green
715. pygame.display.update()
716. time.sleep(0.3)
717. GameScreen.fill(backGroundColour) # Back Ground Colour
718. GameScreen.blit(HUDImg, ((boardSize[0]**\***21), 10))
719. GameScreen.blit(Images['WinImg'], ((boardSize[0]**\***21)**+**5, 150)) # Displays Win Image on the HUD
720. DisplayAll(Game, GameScreen, Images, TileX, TileY)
721. GameHud.\_DisplayClock(started, GameScreen, BackGroundImg)
722. GameHud.\_DisplayScore(GameScreen, BackGroundImg)
723. GameHud.\_DisplayAmountOfFlagsandBombs(GameScreen)
724. GameLoop **=** False
725. pygame.display.update()
726. GameLoop **=** False
727. NextGameLoop **=** True
728. **while** NextGameLoop:
729. **for** event **in** pygame.event.get():
730. **if** event.type **==** pygame.KEYDOWN:
731. NextGameLoop **=** False
732. **elif** pygame.mouse.get\_pressed()[1] **==** True:
733. NextGameLoop **=** False
734. QuickReset **=** True
735. **elif** event.type **==** pygame.QUIT:
736. NextGameLoop **=** False
738. **break**
739. AddExtraNumberTiles(AllTiles, RecordedEmptyTiles) # Displays the Tiles above, below, right and left of each empty tile to show the number tile
740. RestartButton.\_Draw(GameScreen) # draws the restart button
741. # Display Grid then overlaps the tiles that are ON above it so you can see gaps in it
742. DisplayAll(Games[x], GameScreen, Images, TileX, TileY)
743. **for** i **in** range(len(AllTiles)):
744. AllTiles[i].\_DisplayTiles(GameScreen) # Display Tiles
745. AllTiles[i].\_DisplayFlags(GameScreen) # Display Flags
746. GameHud.\_DisplayClock(started, GameScreen, BackGroundImg) # Display clock
747. GameHud.\_DisplayScore(GameScreen, BackGroundImg) # Display Score
748. GameHud.\_DisplayAmountOfFlagsandBombs(GameScreen) # Displays Flags placed and Amount of bombs in grid
749. pygame.display.update() # Updates Display
750. Clock.tick(100) # Clock Tick
751. **for** event **in** pygame.event.get():
752. **if** Clicks >**=** 0:
753. **if** pygame.mouse.get\_pressed()[1] **==** True: # Checks that if the Left Mouse Button is pressed [0] = Left [1] = Middle [2] = Right
754. **if** EasyReset **==** True:
755. **if** Clicks > 0 : # if they restart the game and havent clicked anything, there wont be any mines under the tiles as the grid is only created after the first click to avoid the first click being a mine
756. DisplayAll(Games[x], GameScreen, Images, TileX, TileY) # Displays all as game is over so you can see where the mines were
757. pygame.display.update()
758. time.sleep(2)
759. GameLoop **=** False
760. QuickReset **=** True # Sets QuickReset to TRUE meaning you skip the Menu selection part and all your chosen settings remain the same and you are start a new game right away
761. **if** RestartButton.\_checkForClick() **==** True:
762. **if** Clicks > 0:
763. DisplayAll(Games[x], GameScreen, Images, TileX, TileY) # Displays all as game is over so you can see where the mines were
764. pygame.display.update()
765. time.sleep(1)
766. GameLoop **=** False
767. **if** event.type **==** pygame.QUIT: # Closes Program
768. GameLoop **=** False
769. pygame.quit()
770. sys.exit
772. **if** \_\_name\_\_ **==** '\_\_main\_\_':
773. Main(Images, ListImages, TileX, TileY) # Passes in Images

## Menu Procedures Module

1. **import** pygame, time, sys
3. **from** Data\_Base **import** CheckAvailability, AddPlayer, SelectFromPlayer, SelectFromGames
5. **def** setUpMenu():
6. # Sets up the Menu
7. MenuTick **=** 0 # Restarts the Menu Ticks
8. MenuScreenSize **=** (450, 250) # Sets the Size
9. MenuScreen **=** pygame.display.set\_mode(MenuScreenSize) # Creates Screen
10. pygame.display.set\_caption('Menu') # Gives the Screen a Caption
11. **return** MenuScreen # Returns the Screen
13. **def** CreateNewLogIn(Images, Button):
14. CreateAccountMenu **=** pygame.display.set\_mode((300, 200))
15. CreateAccountLoop **=** True
16. UsernameButton **=** Button(120, 10, Images['InputNameImg'], 1)
17. PasswordButton **=** Button(120, 60, Images['InputNameImg'], 1)
18. Password2Button **=** Button(120, 110, Images['InputNameImg'], 1)
19. CreateAccount2Button **=** Button(205, 150, Images['CreateAccountImg'], 0.1)
20. LogInButtons **=** [UsernameButton, PasswordButton, Password2Button, CreateAccount2Button]
21. pygame.font.init()
22. Font **=** pygame.font.Font('freesansbold.ttf', 15)
23. UserNameText **=** Font.render('Username: ', True, (0, 0, 0))
24. PasswordeText **=** Font.render('Password: ', True, (0, 0, 0))
25. ReEnterPasswordText **=** Font.render('ReEnter Pass: ', True, (0, 0, 0))
26. Username **=** ''
27. Password **=** ''
28. Password2 **=** ''
29. **while** CreateAccountLoop:
30. CreateAccountMenu.fill((255, 255, 255))
31. CreateAccountMenu.blit(UserNameText, (20, 15))
32. CreateAccountMenu.blit(PasswordeText, (20, 65))
33. CreateAccountMenu.blit(ReEnterPasswordText, (20, 115))
34. UsernameDisplay **=** Font.render(Username, True, (0, 0, 0))
35. PasswordHidden **=** ''
36. Password2Hidden **=** ''
37. **for** i **in** range (len(Password)):
38. PasswordHidden **=** PasswordHidden **+** '\*'
39. **for** i **in** range (len(Password2)):
40. Password2Hidden **=** Password2Hidden **+** '\*'
41. PasswordDisplay **=** Font.render(PasswordHidden, True, (0, 0, 0))
42. Password2Display **=** Font.render(Password2Hidden, True, (0, 0, 0))
43. Texts **=** [UsernameDisplay, PasswordDisplay, Password2Display]
44. **for** i **in** range(len(LogInButtons)):
45. LogInButtons[i].\_Draw(CreateAccountMenu)
46. **if** UsernameButton.\_checkForClick():
47. Username **=** FetchInputText('Both', 15, [120, 10], 'NewUsername', Texts, CreateAccountMenu, Images, '', LogInButtons)
48. Available **=** CheckAvailability(Username)
49. **if** Available **==** True:
50. print('free')
51. **else**:
52. print('taken')
53. time.sleep(0.1)
54. CreateAccountMenu.fill((255, 0, 0))
55. pygame.display.update()
56. time.sleep(0.3)
57. Username **=** ''
58. **if** PasswordButton.\_checkForClick():
59. Password **=** FetchInputText('Both', 15, [120, 60], 'NewPassword', Texts, CreateAccountMenu, Images, '', LogInButtons)
60. **if** Password2Button.\_checkForClick():
61. Password2 **=** FetchInputText('Both', 15, [120, 110], 'NewPassword2', Texts, CreateAccountMenu, Images, '', LogInButtons)
62. **if** CreateAccount2Button.\_checkForClick():
63. **if** len(Username) > 1 **and** len(Password) > 3:
64. **if** Password **==** Password2:
65. AddPlayer(Username, Password)
66. time.sleep(0.1)
67. CreateAccountMenu.fill((0, 255, 0))
68. pygame.display.update()
69. time.sleep(0.3)
70. CreateAccountLoop **=** False
71. print('Account Created')
73. **else**:
74. print('Passwords do not match')
75. time.sleep(0.1)
76. CreateAccountMenu.fill((255, 0, 0))
77. pygame.display.update()
78. time.sleep(0.3)
79. Password **=** ''
80. Password2 **=** ''
81. **else**:
82. print('Username/password too short')
83. time.sleep(0.1)
84. CreateAccountMenu.fill((255, 0, 0))
85. pygame.display.update()
86. time.sleep(0.3)
87. Username **=** ''
88. Password **=** ''
89. Password2 **=** ''
90. **else**:
91. CreateAccountMenu.blit(UsernameDisplay, (125, 15))
92. CreateAccountMenu.blit(PasswordDisplay, (125, 65))
93. CreateAccountMenu.blit(Password2Display, (125, 115))
94. pygame.display.update()
95. **for** event **in** pygame.event.get():
96. **if** event.type **==** pygame.QUIT:
97. pygame.quit()
98. CreateAccountLoop **=** False
100. **def** FetchInputText(InputType, MaxSize, POS, whichInput, Texts, Screen, Images, NameDisplay, Buttons):
101. # Gets the users text input
102. Done **=** False
103. Text **=** '' # Empty Starting string
104. **while** Done **==** False **and** len(Text) < MaxSize: # Limits the characters the user can enter
105. **for** event **in** pygame.event.get():
106. **if** event.type **==** pygame.KEYDOWN:
107. **if** event.key **==** pygame.K\_BACKSPACE: #
108. **if** len(Text) > 0: # Makes sure there is text to get rid of
109. Text **=** Text.rstrip(Text[**-**1]) # Removes the last character of the string
110. **if** event.key **==** pygame.K\_KP\_ENTER **or** event.key **==** pygame.K\_RETURN:
111. Done **=** True # Finsihes Loop
112. **if** event.key **==** pygame.K\_SPACE **and** InputType **==** 'Alpha': # for the name, allows you to enter a space
113. Text **=** Text **+** ' '
114. **else**:
115. **if** InputType **==** 'Alpha':
116. **if** event.unicode.isalpha(): # Returns True if they key pressed is an alphabetic character
117. Text **=** Text **+** event.unicode
119. **elif** InputType **==** 'Both':
120. **if** event.unicode.isalpha():
121. Text **=** Text **+** event.unicode
122. **elif** event.unicode.isdigit():
123. Text **=** Text **+** event.unicode
124. **else**:
125. **if** event.unicode.isdigit():
126. Text **=** Text **+** event.unicode
127. **if** event.type **==** pygame.QUIT:
128. pygame.quit()
129. sys.exit()
130. # Displays rest of import information on screen while in Loop
131. **if** whichInput !**=** 'NewUsername' **and** whichInput !**=** 'NewPassword' **and** whichInput !**=** 'NewPassword2':
132. Buttons[9].\_Draw(Screen)
133. Buttons[10].\_Draw(Screen)
134. Buttons[11].\_Draw(Screen)
136. **if** whichInput **==** 'Name':
137. Screen.blit(Images['SelectedNameImg'], (20, 10))
138. **elif** whichInput **==** 'Password':
139. Screen.blit(Images['SelectedNameImg2'], (20, 10))
140. **else**:
141. Screen.blit(Images['InputNameImg'], (20, 10))
142. **else**:
143. Buttons[0].\_Draw(Screen)
144. Buttons[1].\_Draw(Screen)
145. Buttons[2].\_Draw(Screen)
146. Screen.blit(Images['SelectedNameImg'], (POS[0], POS[1]))
148. **if** whichInput **==** 'Name':
149. Screen.blit(Texts[0], (150, 200))
150. Screen.blit(Texts[1], (10, 180))
151. Screen.blit(Texts[2], (10, 200))
152. **if** whichInput **==** 'Mines':
153. Screen.blit(Texts[1], (10, 180))
154. Screen.blit(Texts[2], (10, 200))
155. Screen.blit(NameDisplay, (25, 15))
156. **if** whichInput **==** 'Width':
157. Screen.blit(Texts[0], (150, 200))
158. Screen.blit(Texts[2], (10, 200))
159. Screen.blit(NameDisplay, (25, 15))
160. **if** whichInput **==** 'Height':
161. Screen.blit(Texts[0], (150, 200))
162. Screen.blit(Texts[1], (10, 180))
163. Screen.blit(NameDisplay, (25, 15))
164. **if** whichInput **==** 'Password':
165. Screen.blit(Texts[0], (150, 200))
166. Screen.blit(Texts[1], (10, 180))
167. Screen.blit(Texts[2], (10, 200))
168. **if** whichInput **==** 'NewUsername':
169. Screen.blit(Texts[1], (125, 65))
170. Screen.blit(Texts[2], (125, 115))
171. **if** whichInput **==** 'NewPassword':
172. Screen.blit(Texts[0], (125, 15))
173. Screen.blit(Texts[2], (125, 115))
174. **if** whichInput **==** 'NewPassword2':
175. Screen.blit(Texts[0], (125, 15))
176. Screen.blit(Texts[1], (125, 65))
177. # Constanlty Updates the Text to the screen so you can see what you are typing
178. pygame.font.init()
179. Font **=** pygame.font.Font('freesansbold.ttf', 15)
180. **if** whichInput **==** 'Password' **or** whichInput **==** 'NewPassword2' **or** whichInput **==** 'NewPassword':
181. Text2 **=** ''
182. **for** i **in** range(len(Text)):
183. Text2 **=** Text2 **+** '\*'
184. TextDisplay **=** Font.render(Text2, True, (0, 0, 0))
185. **else**:
186. TextDisplay **=** Font.render(Text, True, (0, 0, 0))
188. **if** whichInput **==** 'NewUsername' **or** whichInput **==** 'NewPassword' **or** whichInput **==** 'NewPassword2':
189. Screen.blit(TextDisplay, (POS[0]**+**5, POS[1]**+**5))
190. **else**:
191. Screen.blit(TextDisplay, (POS[0], POS[1]))
192. pygame.display.update() # Updates Display as is out of main Loop
193. **return** Text # Returns the User input
195. **def** CreateButtons(Images, Button):
196. # Creates Buttons for the Main Menu, Only has to be run once
197. CreateAccountButton **=** Button(200, 12, Images['CreateAccountImg'], 0.1)
198. BeginnerButton **=** Button(20, 50, Images['SelectBoxImg'], 0.08)
199. IntermediateButton **=** Button(20, 70, Images['SelectBoxImg'], 0.08)
200. ExpertButton **=** Button(20, 90, Images['SelectBoxImg'], 0.08)
202. CustomButton **=** Button(20, 110, Images['SelectBoxImg'], 0.08)
203. TileLockOnButton **=** Button(150, 50, Images['SelectBoxImg2'], 0.08)
204. SaveScoreButton **=** Button(150, 90, Images['SelectBoxImg2'], 0.08)
205. EasyResetButton **=** Button(150, 70, Images['SelectBoxImg2'], 0.08)
206. PlayButton **=** Button(300, 60, Images['PlayButtonImg'], 1)
207. QuitButton **=** Button(300, 160, Images['QuitButtonImg'], 1)
208. BoardButton **=** Button(300, 110, Images['LeaderBoardImg'], 1)
209. HeightInputButton **=** Button(70, 198, Images['InputImg'], 1)
210. WidthInputButton **=** Button(70, 175, Images['InputImg'], 1)
211. MineInputButton **=** Button(205, 195, Images['InputImg'], 1)
212. InputNameButton **=** Button(20, 10, Images['InputNameImg'], 1)
213. # Buttons for leaderboard
214. ShowBeginnerButton **=** Button(20, 10, Images['LeaderboardBackgroundButton'], 0.06)
215. ShowIntermediateButton **=** Button(100, 10, Images['LeaderboardBackgroundButton'], 0.06)
216. ShowExpertButton **=** Button(180, 10, Images['LeaderboardBackgroundButton'], 0.06)
217. ShowBeginnerButton2 **=** Button(160, 10, Images['LeaderboardBackgroundButton'], 0.06)
218. ShowIntermediateButton2 **=** Button(240, 10, Images['LeaderboardBackgroundButton'], 0.06)
219. ShowExpertButton2 **=** Button(320, 10, Images['LeaderboardBackgroundButton'], 0.06)
220. OrderByTimeButton **=** Button(280, 10, Images['LeaderboardBackgroundButton'], 0.06)
221. OrderByDateButton **=** Button(360, 10, Images['LeaderboardBackgroundButton'], 0.06)
222. Buttons **=** [BeginnerButton, IntermediateButton, ExpertButton, CustomButton, TileLockOnButton, SaveScoreButton, EasyResetButton, PlayButton, QuitButton, HeightInputButton, WidthInputButton, MineInputButton, InputNameButton, BoardButton, CreateAccountButton]
223. LeaderBoardButtons **=** [ShowBeginnerButton, ShowIntermediateButton, ShowExpertButton, OrderByTimeButton, OrderByDateButton]
224. AccountStatisticButtons **=** [ShowBeginnerButton2, ShowIntermediateButton2, ShowExpertButton2]
225. **return** Buttons, LeaderBoardButtons, AccountStatisticButtons
227. **def** LeaderBoard(Settings, Data, Screen):
228. # Organises and Displays the top 50 leaderboard results for each Difficulty
229. pygame.font.init()
230. Font **=** pygame.font.Font('freesansbold.ttf', 10)
231. **if** len(Data) > 50:
232. TopScores **=** 50 # Makes sure only 50 can be displayed but allows it to function if there arent more than 50 results
233. **else**:
234. TopScores **=** len(Data)
235. j **=** 0
236. i **=** 0
237. count **=** 1
238. **while** count < TopScores**+**1: # Loops through each results displaying it
239. **if** j > 24 **and** i **==** 0: # When it reaches the bottom
240. i **=** i **+** 1 # Makes it go to the next line
241. j **=** 0 # AND Makes it go back to the top
242. # Organises the Text
243. Text **=** str(Data[j])
244. Text **=** Text.rstrip(")")
245. Text **=** Text.replace("datetime.datetime(", " | ")
246. Text **=** Text.replace("Decimal(", " ")
247. Text **=** Text.replace("('", "")
248. Text **=** Text.replace("'", "")
249. Text **=** Text.replace(",", "")
250. Text **=** Text.replace(")", "  ")
251. Top **=** False
252. **if** Settings[1] **==** 'Time': # If its in the Time order, the top 3 times get colourful titles
253. **if** count **==** 1:
254. Text **=** Font.render((str(count) **+**' '**+** Text), True, (201, 176, 55)) # renders GOLD
255. Top **=** True
256. **if** count **==** 2:
257. Text **=** Font.render((str(count) **+**' '**+** Text), True, (215, 215, 215)) # renders SILVER
258. Top **=** True
259. **if** count **==** 3:
260. Text **=** Font.render((str(count) **+**' '**+** Text), True, (106, 56, 5)) # renders BRONZE
261. Top **=** True
262. **if** count > 3:
263. Text **=** Font.render((str(count) **+**' '**+** Text), True, (0, 0, 0)) # renders plain black
264. **else**:
265. Text **=** Font.render((str(count) **+**' '**+** Text), True, (0, 0, 0)) # renders plain black
266. Screen.blit(Text, ((i**\***200) **+** 20, (j **\*** 17)**+**60))
267. j **=** j **+** 1
268. count **=** count **+** 1
270. **def** AccountStatstics(PlayerID, Difficulty):
271. PlayerDateCreated **=** SelectFromPlayer(PlayerID)
272. PlayerGamesWon, PlayerGamesLost, PlayerGamesPlayed, GameHistory, TotalGamesPlayed, TotalGamesWon **=** SelectFromGames(PlayerID, Difficulty)
273. PlayerStats **=** [PlayerGamesWon, PlayerGamesLost, PlayerGamesPlayed, PlayerDateCreated, GameHistory]
274. GlobalStats **=** [TotalGamesPlayed, TotalGamesWon]
275. **return** GlobalStats, PlayerStats

## Game Procedures Module

1. # Procedure Module
2. **import** pygame, time, sys
4. **def** DisplayAll(Game, Screen, Images, TileX, TileY):
5. Grid **=** Game.\_Grid
6. # When game is over AND during each iteration of maingame loop, shows everything under the tiles
7. Tiles **=** [Images['EmptyImg'], Images['OneImg'], Images['TwoImg'], Images['ThreeImg'], Images['FourImg'], Images['FiveImg'], Images['SixImg'], Images['SevenImg'], Images['EightImg']] # list of images from dictionary
8. **for** i **in** range(0, Game.\_boardSize[0]):
9. **for** j **in** range(0, Game.\_boardSize[1]): # Loops through each value of 2Darray Grid
10. **if** Grid[j][i] **==** 'X': # Checks if its a Bomb
11. Screen.blit(Images['BombImg'], ((TileX **\*** i)**+**5, (TileY **\*** j) **+** 10)) # If its a bomb it displays Bomb
12. **for** x **in** range(9): # Loops through every possible number 1-8, if its equal to x ( 1-8 ) then it displays Xs position value of Images ( 1-8 )
13. **if** Grid[j][i] **==** str(x):
14. Screen.blit(Tiles[x], ((TileX**\***i)**+**5, (TileY**\***j) **+** 10)) # Displays Number Image
16. **def** RemoveTiles(AllTiles, FlagLock, Hud, Game, clicks, Button):
17. **for** i **in** range(len(AllTiles)):
18. **if** AllTiles[i].\_checkForClick() **==** True: # Checks What Remaining On Tiles Collides with Mouse Position using Button Method checkForClick
19. time.sleep(0.04) # Delay to stop user getting rid of every tile on accident
20. **if** FlagLock **==** True: # If flag lock is on, the flag status must be FALSE in order for the user to click the Tile
21. **if** AllTiles[i].\_FlagOn **==** False:
22. **if** AllTiles[i].\_TileOn **==** True:
23. clicks **=** clicks **+** 1
24. AllTiles[i].\_TileOn **=** False
25. X **=** AllTiles[i].\_TileNumber[0]
26. Y **=** AllTiles[i].\_TileNumber[1]
27. TileNum **=** [X, Y]
28. **if** Game.\_Grid[Y][X] **==** 'X': # If the tile removed has a Mine underneath it it returns False which tells the program you have lost the game
29. **return** False, TileNum, clicks, False
30. **if** clicks **==** 1: # If its the first initial click, Used for making sure the first click cannot be a Mine
31. **return** True, TileNum, clicks, True
32. **else**:
33. **return** True, TileNum, clicks, False
34. **elif** FlagLock **==** False: # Repeat of code but for if flag lock is off, meaning it doesnt have to check if the flag is on or off
35. **if** AllTiles[i].\_TileOn **==** True:
36. clicks **=** clicks **+** 1
37. AllTiles[i].\_TileOn **=** False
38. X **=** AllTiles[i].\_TileNumber[0]
39. Y **=** AllTiles[i].\_TileNumber[1]
40. TileNum **=** [X, Y]
41. **if** Game.\_Grid[Y][X] **==** 'X':
42. **return** False, TileNum, clicks, False
43. **if** clicks **==** 1:
44. **return** True, TileNum, clicks, True
45. **else**:
46. **return** True, TileNum, clicks, False
47. TileNum **=** [**-**1, **-**1]
48. **return** True, TileNum, clicks, False
50. **def** AddFlag(AllTiles, Hud):
51. MousePosition **=** pygame.mouse.get\_pos()
52. **if** pygame.mouse.get\_pressed()[2] **==** 1: # If right mouse button is pressed
53. time.sleep(0.15) # Delay So it doesnt immediatly Turn it back off
54. **for** i **in** range(len(AllTiles)):
55. **if** AllTiles[i].\_rect.collidepoint(MousePosition) **and** AllTiles[i].\_TileOn **==** True:
56. **if** AllTiles[i].\_FlagOn **==** True:
57. AllTiles[i].\_FlagOn **=** False
58. Hud.\_Flags.remove(AllTiles[i].\_TileNumber)
59. **else**:
60. AllTiles[i].\_FlagOn **=** True
61. Hud.\_Flags.append(AllTiles[i].\_TileNumber)
63. **def** RemoveAdjacentTiles(Grid, AllTiles, MousePos, Game, Hud):
64. boardSize **=** Game.\_boardSize
65. # Find Empty Tiles
66. EmptyTiles **=** []  # Acts as A Queue
67. Checked **=** [] # Stores Tiles that have been Checked
68. RecordedEmptyTiles **=** [] # Stores all Empty Tiles
69. **if** MousePos[0] > **-**1 **and** MousePos[1] > **-**1:
70. EmptyTiles.append(MousePos)
71. **if** len(EmptyTiles) > 0:
72. x **=** EmptyTiles[0][0]
73. y **=** EmptyTiles[0][1]
74. # All Connecting Ones
75. **if** Grid[y][x] **==** '0':
76. **while** len(EmptyTiles) > 0:
77. x **=** EmptyTiles[0][0]
78. y **=** EmptyTiles[0][1]
79. **if** Grid[y][x] **==** '0' **and** [x, y] **not** **in** Checked:
80. **for** i **in** range(len(AllTiles)**-**1):
81. **if** AllTiles[i].\_TileNumber[0] **==** x **and** AllTiles[i].\_TileNumber[1] **==** y:
82. AllTiles[i].\_TileOn **=** False
83. Checked.append([x, y])
84. EmptyTiles.append([x, y])
85. RecordedEmptyTiles.append([x, y])
86. **if** x **-** 1 > **-**1:
87. **if** Grid[y][x**-**1] **==** '0' **and** [x**-**1, y] **not** **in** Checked:
88. **for** i **in** range(len(AllTiles)**-**1):
89. **if** AllTiles[i].\_TileNumber[0] **==** x**-**1 **and** AllTiles[i].\_TileNumber[1] **==** y:
90. AllTiles[i].\_TileOn **=** False
91. Checked.append([x**-**1, y])
92. EmptyTiles.append([x**-**1, y])
93. RecordedEmptyTiles.append([x**-**1, y])
94. **if** x **+** 1 < boardSize[0]:
95. **if** Grid[y][x**+**1] **==** '0' **and** [x**+**1, y] **not** **in** Checked:
96. **for** i **in** range(len(AllTiles)**-**1):
97. **if** AllTiles[i].\_TileNumber[0] **==** x**+**1 **and** AllTiles[i].\_TileNumber[1] **==** y:
98. AllTiles[i].\_TileOn **=** False
99. Checked.append([x**+**1, y])
100. EmptyTiles.append([x**+**1, y])
101. RecordedEmptyTiles.append([x**+**1, y])
102. **if** y **-** 1 > **-**1:
103. **if** Grid[y**-**1][x] **==** '0' **and** [x, y**-**1] **not** **in** Checked:
104. **for** i **in** range(len(AllTiles)**-**1):
105. **if** AllTiles[i].\_TileNumber[0] **==** x **and** AllTiles[i].\_TileNumber[1] **==** y**-**1:
106. AllTiles[i].\_TileOn **=** False
107. Checked.append([x, y**-**1])
108. EmptyTiles.append([x, y**-**1])
109. RecordedEmptyTiles.append([x, y**-**1])
110. **if** y **+** 1 < boardSize[1]:
111. **if** Grid[y**+**1][x] **==** '0' **and** [x, y**+**1] **not** **in** Checked:
112. **for** i **in** range(len(AllTiles)**-**1):
113. **if** AllTiles[i].\_TileNumber[0] **==** x **and** AllTiles[i].\_TileNumber[1] **==** y**+**1:
114. AllTiles[i].\_TileOn **=** False
115. Checked.append([x, y**+**1])
116. EmptyTiles.append([x, y**+**1])
117. RecordedEmptyTiles.append([x, y**+**1])
118. EmptyTiles.pop(0)
119. **return** RecordedEmptyTiles
120. **def** AddExtraNumberTiles(AllTiles, RecordedEmptyTiles):
121. ExtraNumberTiles **=** []
122. **for** i **in** range(len(RecordedEmptyTiles)):
123. x **=** RecordedEmptyTiles[i][0]
124. y **=** RecordedEmptyTiles[i][1]
125. ExtraNumberTiles.append([x**+**1,y])
126. ExtraNumberTiles.append([x**-**1,y])
127. ExtraNumberTiles.append([x,y**+**1])
128. ExtraNumberTiles.append([x,y**-**1])
129. **for** i **in** range(len(AllTiles)):
130. **if** [AllTiles[i].\_TileNumber[0], AllTiles[i].\_TileNumber[1]] **in** ExtraNumberTiles:
131. AllTiles[i].\_TileOn **=** False

## Database Module

1. # Database Module
3. **import** mysql.connector
4. **from** datetime **import** datetime
6. # Creates Player Table
7. #cursor.execute("CREATE TABLE Players (PlayerID int PRIMARY KEY NOT NULL AUTO\_INCREMENT, PlayerName VARCHAR(15) NOT NULL, PlayerPassword VARCHAR(20) NOT NULL, DateCreated datetime NOT NULL)")
8. #cursor.execute("CREATE TABLE Games (PlayersID int, Difficulty ENUM('Beginner', 'Intermediate', 'Expert') NOT NULL, Time DECIMAL(5, 1) NOT NULL, Score DECIMAL(4, 1) NOT NULL, Created datetime NOT NULL)")

11. **def** AddToTable(Difficulty, PlayersID, Time, Score):
12. **if** PlayersID !**=** **-**1: # If they arent guest and have logged in
13. # %s allows me to pass in variables
14. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
15. cursor **=** DATABASE.cursor(buffered**=**True)
16. cursor.execute('INSERT INTO Games (PlayersID, Difficulty, Time, Score, Created) VALUES (%s, %s, %s, %s, %s)', (PlayersID, Difficulty, Time, Score, datetime.now())) # Adds info and the playerID into the game table
17. DATABASE.commit()
19. **def** SelectFromTable(Difficulty, Order):
20. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
21. cursor **=** DATABASE.cursor(buffered**=**True)
22. **if** Order **==** 'Time':
23. cursor.execute("SELECT PlayersID, Time FROM Games WHERE Difficulty = '%s' AND Score = '100.0' ORDER BY Time" **%**(Difficulty))
24. **if** Order **==** 'Date':
25. cursor.execute("SELECT PlayersID, Created FROM Games WHERE Difficulty = '%s' AND Score = '100.0' ORDER BY Created" **%**(Difficulty))
26. data **=** []
27. PlayerIDs **=** []
28. # Probably a smarter way of doing this
29. **for** x **in** cursor:
30. data.append(x[1])
31. PlayerIDs.append(x[0])
32. PlayerNames **=** []
33. **for** i **in** range(len(PlayerIDs)):
34. **if** PlayerIDs[i] **==** **-**1:
35. PlayerNames.append("('Guest')")
36. **else**:
37. cursor.execute("SELECT PlayerName FROM Players WHERE PlayerID = '%s'" **%**(PlayerIDs[i]))
38. **for** x **in** cursor:
39. PlayerNames.append(x)
40. info **=** []
41. **for** i **in** range(len(PlayerNames)):
42. info.append(str(PlayerNames[i]) **+** str(data[i]))
43. print(info)
44. **return** info
46. **def** SelectFromPlayer(PlayerID):
47. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
48. cursor **=** DATABASE.cursor(buffered**=**True)
49. cursor.execute("SELECT DateCreated FROM Players WHERE PlayerID = '%s'" **%**(PlayerID))
50. **for** x **in** cursor:
51. DateCreated **=** x
52. **return** DateCreated

55. **def** SelectFromGames(PlayerID, Difficulty):
56. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
57. cursor **=** DATABASE.cursor(buffered**=**True)
58. cursor.execute("SELECT Difficulty, Score, Time, Created FROM Games WHERE PlayersID = '%s' AND Difficulty = '%s' ORDER BY Created DESC" **%**(PlayerID, Difficulty))
59. GamesPlayed **=** 0
60. PlayerGameHistory **=** ''
61. **for** x **in** cursor:
62. PlayerGameHistory **=** PlayerGameHistory **+** str(x)
63. GamesPlayed **+=** 1
64. cursor.execute("SELECT \* FROM Games WHERE PlayersID = '%s' AND Score = '100.0' AND Difficulty = '%s'" **%**(PlayerID, Difficulty))
65. GamesWon **=** 0
66. **for** x **in** cursor:
67. GamesWon **+=** 1
68. GamesLost **=** GamesPlayed **-** GamesWon
70. cursor.execute("SELECT \* FROM Games")
71. TotalGamesPlayed **=** 0
72. **for** x **in** cursor:
73. TotalGamesPlayed **+=** 1
74. cursor.execute("SELECT \* FROM Games WHERE Score = '100.0'")
75. TotalGamesWon **=** 0
76. **for** x **in** cursor:
77. TotalGamesWon **+=** 1
79. **return** GamesWon, GamesLost, GamesPlayed, PlayerGameHistory, TotalGamesPlayed, TotalGamesWon

82. **def** clearAllFromTable():
83. # Used for manually clearing tables
84. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
85. cursor **=** DATABASE.cursor(buffered**=**True)
86. cursor.execute("TRUNCATE TABLE Players")
87. cursor.execute("TRUNCATE TABLE Games")

90. **def** CheckAvailability(Username):
91. # Checks if the Username is already in use to avoid people stealing peoples names
92. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
93. cursor **=** DATABASE.cursor(buffered**=**True)
94. cursor.execute("SELECT PlayerName, PlayerPassword FROM Players WHERE PlayerName = '%s'" **%**(Username))
95. y **=** 0
96. **for** x **in** cursor:
97. y **=** y **+** 1
98. **if** y **==** 0:
99. **return** True # Not taken
100. **else**:
101. **return** False # Taken
103. **def** AddPlayer(Username, Password):
104. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
105. cursor **=** DATABASE.cursor(buffered**=**True)
106. cursor.execute("INSERT INTO Players (PlayerName, PlayerPassword, DateCreated) VALUES (%s, %s, %s)", (Username, Password, datetime.now()))
107. DATABASE.commit()

110. **def** LogIn(Username, Password):
112. # Logs Into account to not appear as guest
114. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
115. cursor **=** DATABASE.cursor(buffered**=**True)
116. cursor.execute("SELECT PlayerID, PlayerName, PlayerPassword FROM Players WHERE PlayerName = '%s' AND PlayerPassword = '%s'" **%**(Username, Password))
117. y **=** 0
118. **for** x **in** cursor: # Checks it exsits
119. y **=** y **+** 1
120. PlayerID **=** x[0]
121. **if** y **==** 0:
122. PlayerID **=** 0
123. print('Password wrong')
124. **return** PlayerID, Username, False # Returns the players ID
125. **else**:
126. print('logged in')
127. **return** PlayerID, Username, True

## Load Images Module

1. # Database Module
3. **import** mysql.connector
4. **from** datetime **import** datetime
6. # Creates Player Table
7. #cursor.execute("CREATE TABLE Players (PlayerID int PRIMARY KEY NOT NULL AUTO\_INCREMENT, PlayerName VARCHAR(15) NOT NULL, PlayerPassword VARCHAR(20) NOT NULL, DateCreated datetime NOT NULL)")
8. #cursor.execute("CREATE TABLE Games (PlayersID int, Difficulty ENUM('Beginner', 'Intermediate', 'Expert') NOT NULL, Time DECIMAL(5, 1) NOT NULL, Score DECIMAL(4, 1) NOT NULL, Created datetime NOT NULL)")

11. **def** AddToTable(Difficulty, PlayersID, Time, Score):
12. Time **=** round(int(Time), 3)
13. **if** PlayersID !**=** **-**1: # If they arent guest and have logged in
14. # %s allows me to pass in variables
15. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
16. cursor **=** DATABASE.cursor(buffered**=**True)
17. cursor.execute('INSERT INTO Games (PlayersID, Difficulty, Time, Score, Created) VALUES (%s, %s, %s, %s, %s)', (PlayersID, Difficulty, Time, Score, datetime.now())) # Adds info and the playerID into the game table
18. DATABASE.commit()
20. **def** SelectFromTable(Difficulty, Order):
21. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
22. cursor **=** DATABASE.cursor(buffered**=**True)
23. **if** Order **==** 'Time':
24. cursor.execute("SELECT PlayersID, Time FROM Games WHERE Difficulty = '%s' AND Score = '100.0' ORDER BY Time" **%**(Difficulty))
25. **if** Order **==** 'Date':
26. cursor.execute("SELECT PlayersID, Created FROM Games WHERE Difficulty = '%s' AND Score = '100.0' ORDER BY Created" **%**(Difficulty))
27. data **=** []
28. PlayerIDs **=** []
29. # Probably a smarter way of doing this
30. **for** x **in** cursor:
31. data.append(x[1])
32. PlayerIDs.append(x[0])
33. PlayerNames **=** []
34. **for** i **in** range(len(PlayerIDs)):
35. **if** PlayerIDs[i] **==** **-**1:
36. PlayerNames.append("('Guest')")
37. **else**:
38. cursor.execute("SELECT PlayerName FROM Players WHERE PlayerID = '%s'" **%**(PlayerIDs[i]))
39. **for** x **in** cursor:
40. PlayerNames.append(x)
41. info **=** []
42. **for** i **in** range(len(PlayerNames)):
43. info.append(str(PlayerNames[i]) **+** str(data[i]))
44. print(info)
45. **return** info
47. **def** SelectFromPlayer(PlayerID):
48. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
49. cursor **=** DATABASE.cursor(buffered**=**True)
50. cursor.execute("SELECT DateCreated FROM Players WHERE PlayerID = '%s'" **%**(PlayerID))
51. **for** x **in** cursor:
52. DateCreated **=** x
53. **return** DateCreated

56. **def** SelectFromGames(PlayerID, Difficulty):
57. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
58. cursor **=** DATABASE.cursor(buffered**=**True)
59. cursor.execute("SELECT Difficulty, Score, Time, Created FROM Games WHERE PlayersID = '%s' AND Difficulty = '%s' ORDER BY Created DESC" **%**(PlayerID, Difficulty))
60. GamesPlayed **=** 0
61. PlayerGameHistory **=** ''
62. **for** x **in** cursor:
63. PlayerGameHistory **=** PlayerGameHistory **+** str(x)
64. GamesPlayed **+=** 1
65. cursor.execute("SELECT \* FROM Games WHERE PlayersID = '%s' AND Score = '100.0' AND Difficulty = '%s'" **%**(PlayerID, Difficulty))
66. GamesWon **=** 0
67. **for** x **in** cursor:
68. GamesWon **+=** 1
69. GamesLost **=** GamesPlayed **-** GamesWon
71. cursor.execute("SELECT \* FROM Games")
72. TotalGamesPlayed **=** 0
73. **for** x **in** cursor:
74. TotalGamesPlayed **+=** 1
75. cursor.execute("SELECT \* FROM Games WHERE Score = '100.0'")
76. TotalGamesWon **=** 0
77. **for** x **in** cursor:
78. TotalGamesWon **+=** 1
80. **return** GamesWon, GamesLost, GamesPlayed, PlayerGameHistory, TotalGamesPlayed, TotalGamesWon

83. **def** clearAllFromTable():
84. # Used for manually clearing tables
85. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
86. cursor **=** DATABASE.cursor(buffered**=**True)
87. cursor.execute("TRUNCATE TABLE Players")
88. cursor.execute("TRUNCATE TABLE Games")

91. **def** CheckAvailability(Username):
92. # Checks if the Username is already in use to avoid people stealing peoples names
93. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
94. cursor **=** DATABASE.cursor(buffered**=**True)
95. cursor.execute("SELECT PlayerName, PlayerPassword FROM Players WHERE PlayerName = '%s'" **%**(Username))
96. y **=** 0
97. **for** x **in** cursor:
98. y **=** y **+** 1
99. **if** y **==** 0:
100. **return** True # Not taken
101. **else**:
102. **return** False # Taken
104. **def** AddPlayer(Username, Password):
105. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
106. cursor **=** DATABASE.cursor(buffered**=**True)
107. cursor.execute("INSERT INTO Players (PlayerName, PlayerPassword, DateCreated) VALUES (%s, %s, %s)", (Username, Password, datetime.now()))
108. DATABASE.commit()

111. **def** LogIn(Username, Password):
113. # Logs Into account to not appear as guest
115. DATABASE **=** mysql.connector.connect(host**=**"localhost",user**=**"root",passwd**=**"root",database**=**"minesweeperscores")
116. cursor **=** DATABASE.cursor(buffered**=**True)
117. cursor.execute("SELECT PlayerID, PlayerName, PlayerPassword FROM Players WHERE PlayerName = '%s' AND PlayerPassword = '%s'" **%**(Username, Password))
118. y **=** 0
119. **for** x **in** cursor: # Checks it exsits
120. y **=** y **+** 1
121. PlayerID **=** x[0]
122. **if** y **==** 0:
123. PlayerID **=** 0
124. print('Password wrong')
125. **return** PlayerID, Username, False # Returns the players ID
126. **else**:
127. print('logged in')
128. **return** PlayerID, Username, True

# User Communication

## Analysis Questionnaire Results

## 





## User Feedback Forms Results







# Video

