Python Project: Battleship

For this project I created a battleship game which needs pygame to be installed on the computer before it can be played. It is a single-player game where the goal is to sink all of the ships in the shortest number of turns possible. The game is played on an interactive 10x10 grid that was made with the pygame library and sounds are also included using pygame. The grid coordinates are selected by clicking the mouse. As the game is played, feedback is given on the game board with colors and text is printed out to let the user know what coordinates are being chosen, and if they are hitting or missing the ships.

To use this program, first run the main.py. This will immediately bring up the instructions for playing the game that are located in a .txt file, which has been read and printed out through the use of file IO coding. When the user is ready to start the game, they must enter “1”. The game will not start if anything else is entered. This was achieved with the use of exceptions. The program is stuck in a loop that can only be broken if the statements under try: succeed. Otherwise, the loop continues. When “1” is entered, a small 10x10 grid will appear with the title “Battleship”. This grid was created with the use of a 10-element-long list within another 10-element-long list that behaves like a 2D array in C++. The first index of the list is the row and the second is the column (ex: grid[row][column]) this creates 100 elements that make up the entire grid. Nested for loops are used to access the elements of the 2D list.

Every time this program runs, the code uses the imported random library to pick a random number between 0 and 5. There are 6 different ship configurations in the code, each corresponding to a number between 0 and 5 so the game can have 6 different configurations for the user to play with where the configuration is chosen randomly at the start of each game, so that the player doesn’t know what to expect. To play the game, the user must simply click on the white grid spaces which will result in that space changing color to either red (hit) or green (miss) and it will play a sound matching the result of the click. Every time the player clicks a space on the grid, a variable that represents the amount of turns taken is incremented by one. This keeps track of the amount of turns it takes for the player to beat the game and it will represent their score at the end. The results of each click will also be printed out along with the grid coordinates that were chosen.

In order to determine whether or not the ships have been hit and/or sunk, each ship has its own class where hits are recorded when certain grid positions are clicked and if the amount of hits that a ship receives is equal to the size of the ship, that ship returns a boolean value stating that it has been sunk. After each turn, the current ships sunk will also be printed out. There are a series of if statements that check to see which ships are currently sunk and those that are will be printed out after each turn to let the player know how they are doing. Once all four ships have been sunk, the game is over.

When the player completes the game, the game grid is no longer active and the player is prompted to enter their name. The name is given to a highscores class which takes the name and sets it as a dictionary key. Then that key is assigned a value which is a list containing the amount of turns that it took for the player to complete the game and the time that they completed the game. The time is able to be added through the use of a library called datetime. Then the program opens up a file containing the scores and appends to that file to include the player’s data on the scoreboard. This file is then opened up again, read, and printed to the screen to display all of the previous scores alongside the most recent score. The user is prompted to enter “2” to quit the program.