Annie Spahn

112 Design Proposal

**Project Description**

The name of my project is Battleship. The way game is played is two boards of size 10 by 10 are displayed, with numbers 1-10 displayed across the top and letter A-J displayed down the left side; one board is yours and the other mirrors your opponents. You have 5 ships of varying lengths, from 2-5 spaces, to place anywhere on the board, your opponent will do the same. Then by saying the row and col, A1, you will try to “hit” your opponents ships. After you make your move you will be told if it is a hit or miss. If you sink your opponents ship, you will be notified. The game is over when one person has sunk all the other person’s ships. On your board, your ships will be displayed and all the places your opponent guessed will be marked with pegs, a white peg for a miss and a red one for a hit. On the board that mirrors your opponents will be all of your guesses, once again marked by pegs, white for a miss and a red for a hit.

**Competitive Analysis**

My Battleship game will be played by the same rules as the classic Battleship board game. However, there will be the added option to play the Salvo variation of Battleship. Most of the rules are the same for the Salvo variation except each player guesses as many locations as the have ships at a time instead of one and then is told after which were hits and which were misses. For example, at the beginning of the game each player has five guesses since players start with five ships. Once again, the game ends once one player has sunk all the ships of the other players. My Battleship will also have the option to ask for a hint, which will result in the player getting the location for a square containing an opponent’s ship. Another difference between my game and other Battleship games is that players will have the option to send messages through the game to communicate with each other.

**Structural Plan**

I plan to use sockets to allow my game to be multiplayer, so I will have a separate fill for the code pertaining to the server and the code for the clients. The code for the server will be similar to the code in the dots\_client.py shown in the sockets optional lecture. The code for the client will consist of code to connect to the server, a function (handleMsgFromServer) which will handle receiving the messages and putting them in the queue. The client code will also have the basic tkinter starter code, in which timerFired will be used to check for incoming opponent moves and mouseDragged will encode and send player messages. These two files will cover the socket part of my project.

For the actual implementation of the game Battleship, I will have a file Battleship which will be the main game fil. It will get the messages from the client file, call on other classes in order to run the game, and keep track of the state of the game. There will be 4 states that the game can be in; the start state, player1 state, opponent state and end state. Player1 state is when the player can make their move and opponent state it when no moves are allowed while the player waits for their opponent to make a move. There will be a Board class, in its own file, which will draw the two boards, along with the pegs, and the ships for the player. A 2-d array that will contain which spots on the board contain what will be an attribute of the Board class; there will be an array for each board. There will be a Ship class, in its own file, which will have the location of the ship and the status of each square, hit or not hit, as attributes. The Ship class will have an explodeShip method which will make an explosion animation every time a part of a ship is hit. The Ship class will also have the method sinkShip which will return True and a statement telling the opponent and player which ship has been sunk or False if that ship has not been fully sunk yet.

Then I will have a Player class, in its own file, which will have the methods makeMove, checkLegalMove, checkLose, and returnMessage. makeMove will allow the player to add their designated coordinates to the player 2-d list and have an animation of the player dragging a peg onto the board. checkLegalMove will make sure the player is making a move on the board that they haven’t already made. checkLose checks if all the players ships are sunk yet and will set the state of the game to end if the player has lost. returnMessage will be the function that allows the player to send messages to the opponent. The Opponent class will inherit from the player class but will be its own file.

**Algorithmic Plan**

As of right now the hardest part of my project would be the sockets aspect of it. If I have enough time I would like to implement something more complex, such as better graphics or an AI feature. How I plan to implement the socket code is discussed in the structural plan above.

**Timeline**

4/17 - Have board display done, be able to place ships, and move pegs (maybe have start and end screen done too)

4/22 – Have battleship game working (checking collisions, have ships remove correctly, win statement working)

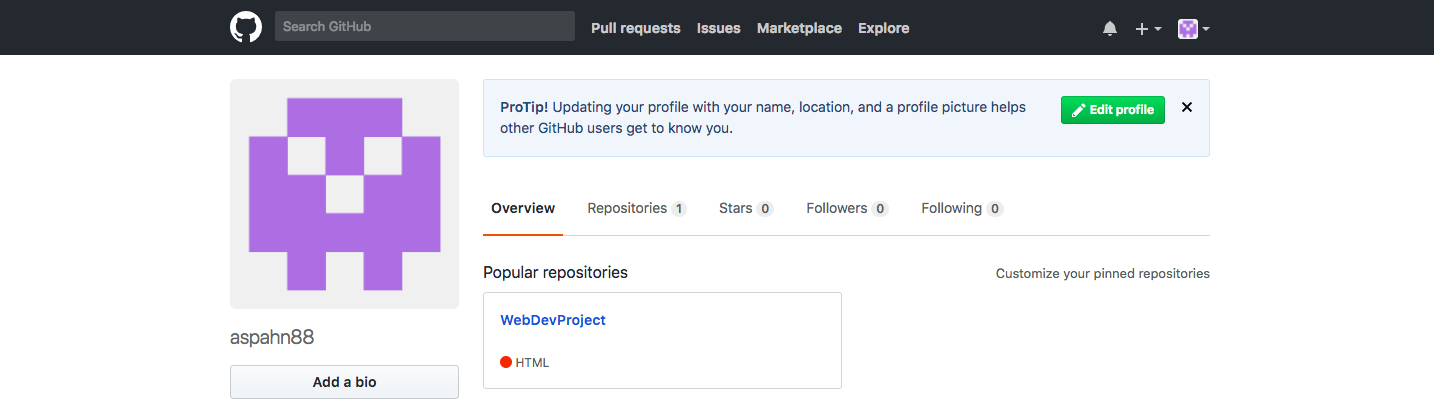
4/25 – Have battleship game with sockets working (be able to play a game against another person)

4/29 – Have messages working, graphics better, hint function implemented, entire game running smoothly. Add extra features.

5/3 – Turn in final project

**Version Control Plan**

I will be using GitHub for my version control. I have used GitHub in the past, so I am already familiar with how to use it.



**Module List**

I will be using Pygame and sockets for this project.