Steve Hirabayashi
PID: 2247504

CNT4713 Project 2 – Battleship Game Client and Server Report

1. Introduction

In this project, I was given the task of developing a Battleship game that can host multiple players and teams that can communicate with one another using a chat feature. In order to successfully complete this assignment, I needed to become familiar with the Python programming language modules for socket programming, multi-threading, and GUI development.

Quite a bit of the knowledge needed for socket programming came from my previous experience developing the FTP client and server. For most of my questions regarding Python, I referred to the "Introducing Python" textbook, as well as the online API. For most of my questions regarding developing the GUI using TkInter came from online references.

Overall, I feel that the current implementation of the Battleship client and server successfully meets the specifications outlined by the professor, although I was not able to conduct a sufficient test on every feature of the program.

2. Problem Statement

The objective of this project was to implement a functional Battleship game client and server that implements both TCP and UDP connections. The server must be centralized and manage the game states and updating for each client (using multithreading to handle asynchronous input from multiple clients). The battle ship game must support at least 4 to 10 users and the option to form teams amongst players. There must also be a chat function that allows users to communicate amongst all players or to specific teams only. Optional requirements include implementing a GUI and adding an SSL layer or encryption for client-server communication.

3. Methodology

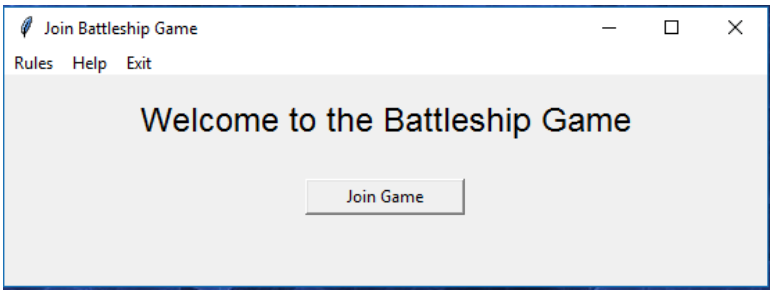
The project was started building the server and client in the same general structure as the FTP client-server program: The server implements multiple threads for each client, creating a TCP connection with each client. Then the configuration files for both the client and server were created, the program reads these files using the ConfigParser module. The configuration files contains information such as the server address, port number for TCP connection, port number range for UDP connections, and the number of players that will participate in the game.

Then on a separate file, the GUI was developed with the events stubbed, the four frames: Connect, Join, Setup, and Game were created separately. Then the game and player classes were created to represent the game state and players respectively. Afterwards, for each phase of the game, the client and server were developed in parallel so that the flow of commands and data through the TCP connection is behaving as intended. Loops were created on the server side to ensure that all players join before the setup phase begins, as well as all players have selected their coordinates before the game actually starts. Finally, the constant polling from the client to the server was created to ensure that all player's game and chat logs as well as the game board are always up to date. This is achieved by giving every player two buffers (game state and chat) that store the updates the clients need to make when a move or a chat message is sent from any client to the server.

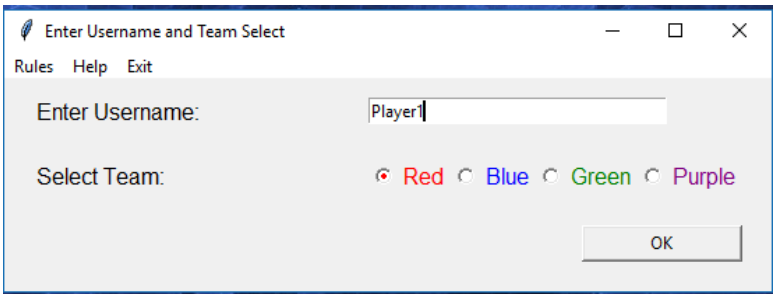
4. Results

Running a game with just two players:

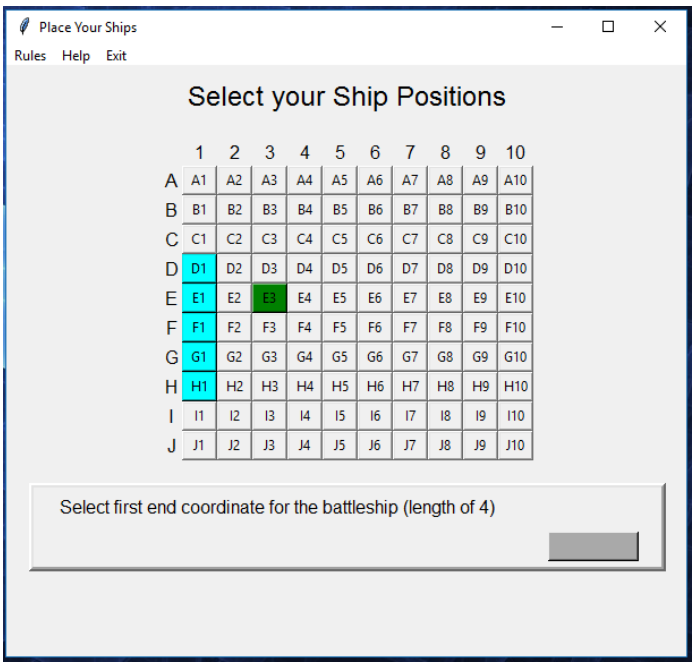
Connect to server:

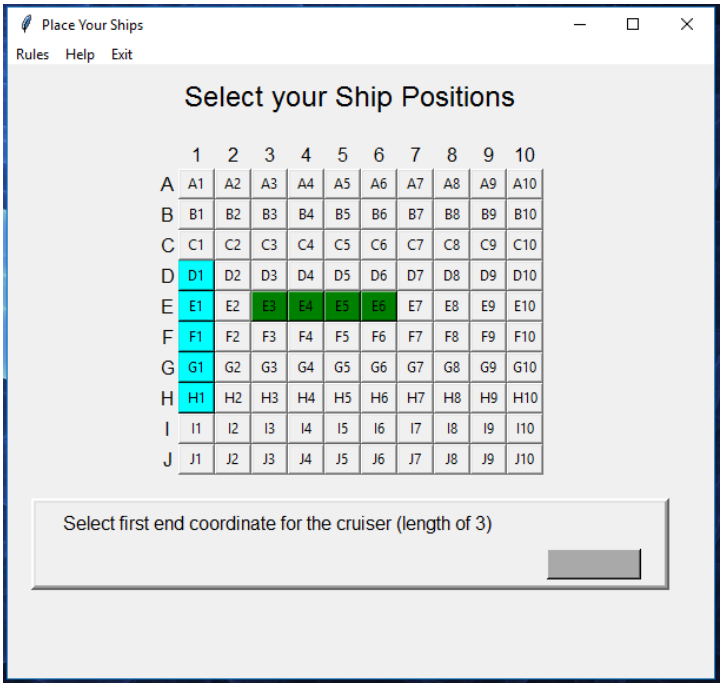


Enter username, select team, then wait for others to complete this step:



Selecting coordinates:

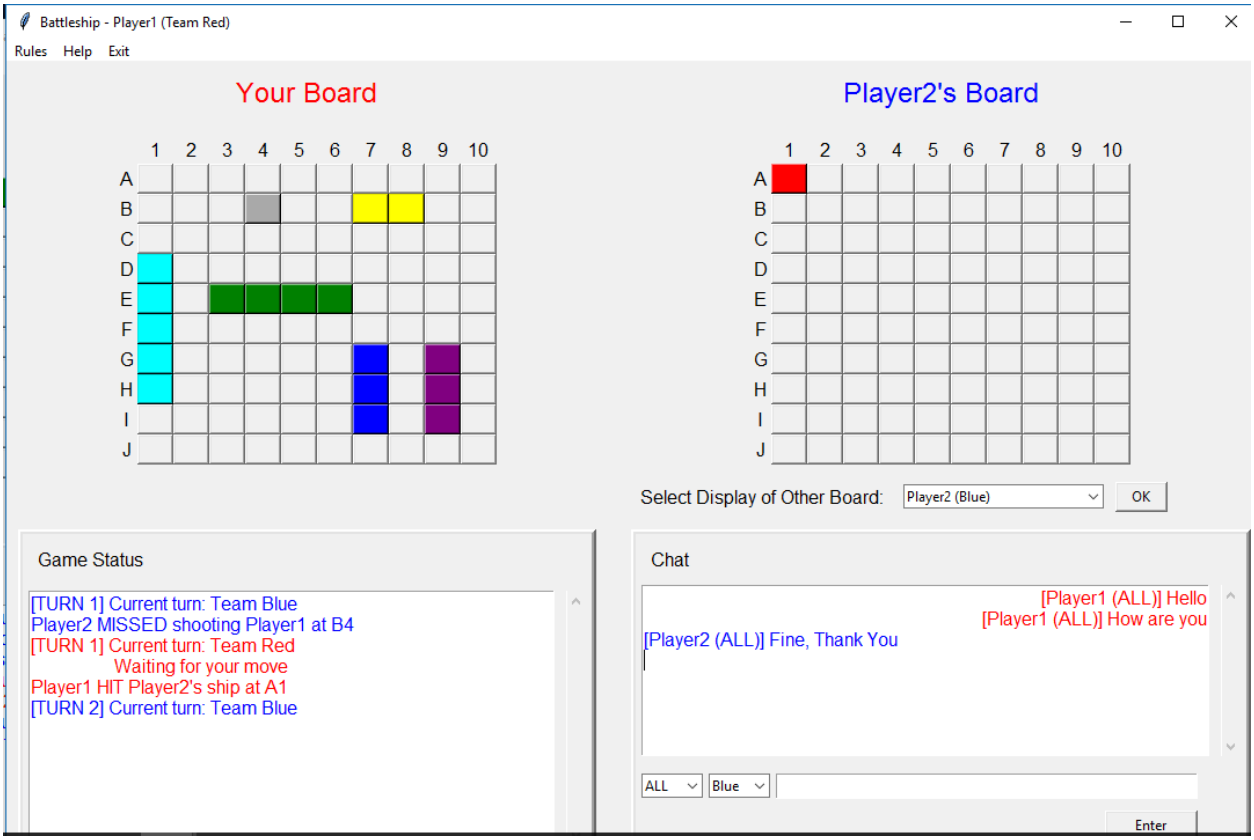




Game board: Making move (Red for a Hit, Gray for a Miss)

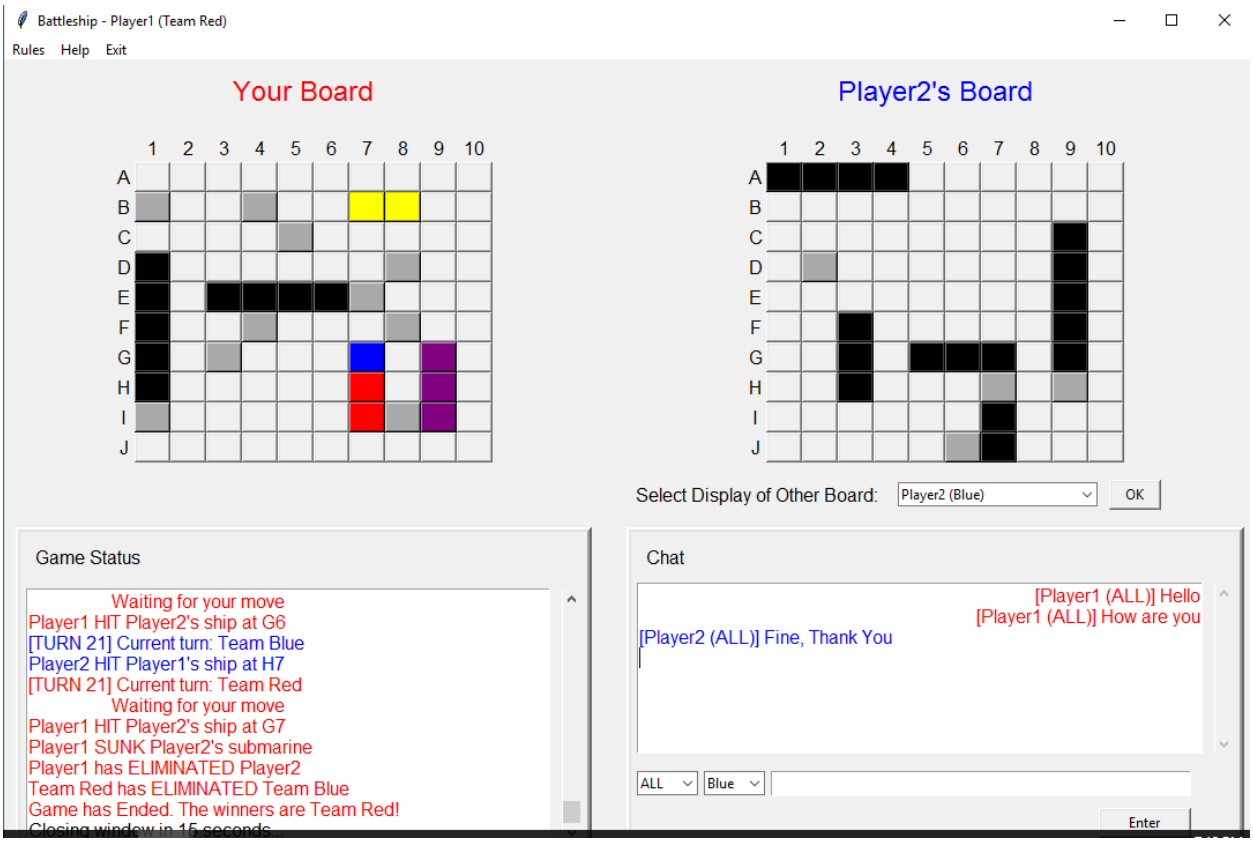
Game state messages displayed on lower left

Chat messages are displayed on lower right



Sinking a shit turns the entire ship Black.

Game ending state:



5. Analysis

As my second large scale Python project, I was able to develop a better understanding of the Python programming language, especially the Tk GUI module. Due to the asynchronous nature of the client inputs (game moves and chat function), learning how to keep the program thread-safe with locks was also important. Much of my struggles originated from unfamiliarity with the Tk widgets and GUI programming, as well as combining socket programming with asynchronous event handling from the clients.

6. References

Aside from referencing previous works involving socket programming in this course, other supplementary websites and textbooks I used:

Klein, B. (n.d.). *Python Tkinter*. Retrieved from Python Course: http://www.python-course.eu/python_tkinter.php

Lubanovic, B. (2015). *Introducing Python.* Sebastopol, CA: O'Reilly Media, Inc.

Tk Tutorial. (n.d.). Retrieved from TkDocs: http://www.tkdocs.com/tutorial/