Battleships Multiplayer Game

Created by:

Eric Rado (4018056)

Francisco Lozada (3606505)

1. Introduction

The main goal of this project was to effectively develop a multiplayer game (multithreaded) that implemented both Transmission Control Protocol (TCP) for its game logic communication and User Datagram Protocol (UDP) for its internal chat system to allow players to interact with each other. We chose to implement the Battleships board game for this project which is a strategic game that focuses on sinking your opponent's ships that are placed on a board and whose positions you do not know. In order to successfully complete the project and achieve the goal of utilizing different protocols for the game and chat of the game we had to implement we had to study the protocols and their process and learn the syntax for coding each in python. The completion of the final product allowed me to gain a low level understanding of socket programming, the Transfer Control Protocol, User Datagram Protocol, and multithreading programming. Overall the main objectives were met. The game server was multithreaded and a graphical user interface was created for the client using Tkinter.

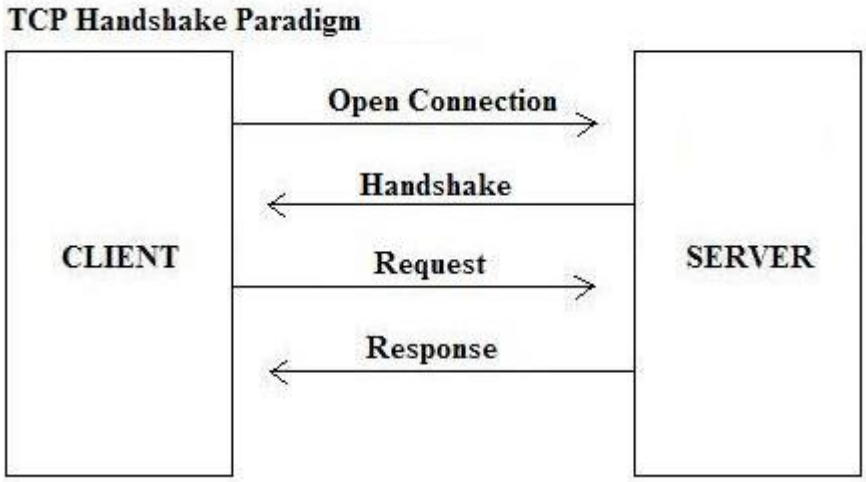
2. Problem Statement

Problems that I needed to solve:

a) Properly establish a TCP connection between both the client and server for the game logic communication
b) Multithread the game server
c) Implement UDP for game chat
d) Create Graphical User Interface for game client

3. Methodology

a) In order to properly establish a TCP connection between the client and server for the game logic communication we had to learn the protocol and syntax for the Python language.



As can be seen in the diagram above, in a TCP connection the client initiates the handshake process with the server, the listening server acknowledges the request and finally the client acknowledges that it knows it has successfully connected with the server.

b) Multithread the game server

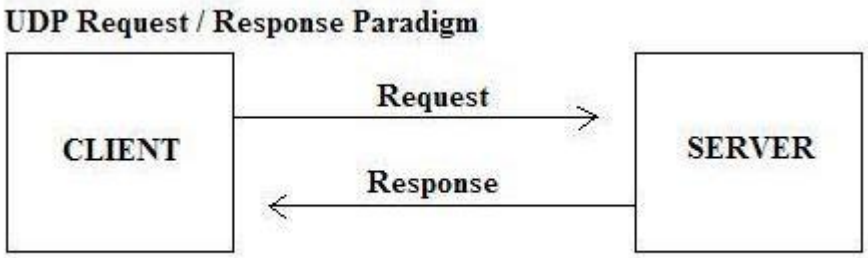
To multithread the game server we first wrote the game logic to have the user play against a computer. We slowly modified the different sections of the code to start communicating with the client. In this case the client was communicating with the server which contained the server and mimicked another player playing. Once the entire game worked well and the TCP game logic communication was set in place we modified a bit more the game logic so that in this case it was not a computer playing with the client but another client. This took some time and a couple of threading locks to allow for the proper ordering of player turns.

c) Implement UDP for game chat

In order to properly implement UDP between the clients and server to have an internal chat system for the game we had to learn the protocol and syntax for the Python language.

d) Create Graphical User Interface for game client

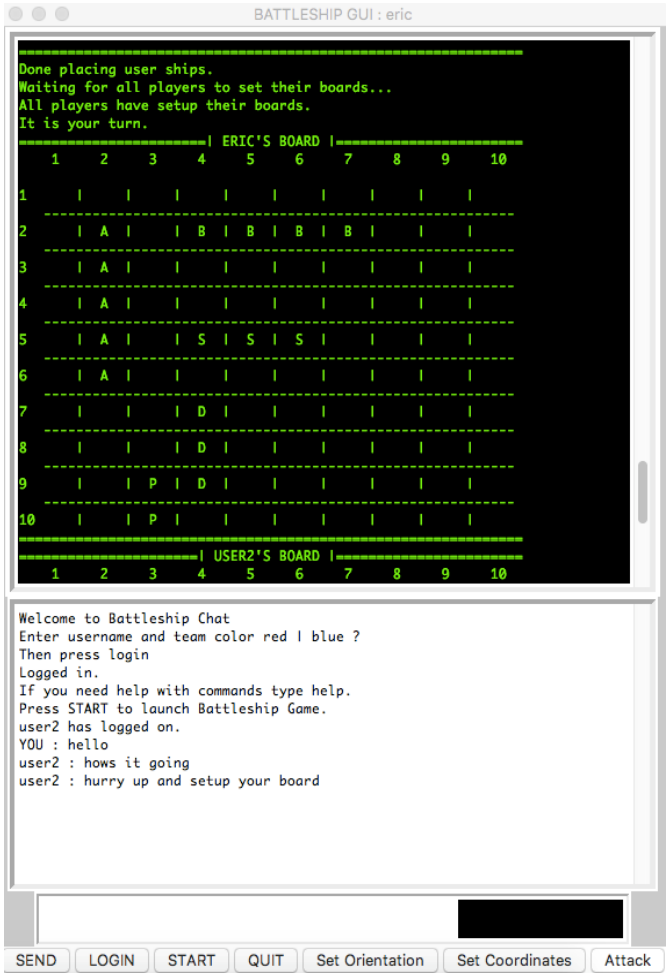
Once the multithreaded game server and the client were implemented with the internal chat system we moved on to design the game display which was previously being printed to the console via the command line. We had to do a lot of research and learning from our part to understand and implement the syntax for Tkinter. After a lot of trial and error and testing we were able to come up with the GUI design demonstrated in the results section.



With UDP there is not handshake required to establish a connection only a request must be made by the client to connect with the server which then returns a response.

4. Results

The following are snippets of the graphical user interface for the client and what the different sections mean and how they are utilized:



TOP WINDOW:

The top-most window of the GUI displays the player boards and the overall game messages and actions.

MIDDLE WINDOW:

The white, middle window of the GUI displays the messages sent by other players and is the chat system for the game.

TWO BOTTOM BOXES:

The left white box at the bottom of the GUI is used for the player to utilize the chat and send messages to other players. It is initially used for the player to input their name and team to register in the server and instantiate a blank board so that they may then place their ships on the board.

The right black box is used to both play the game by inputting the required name and coordinates of the attacks. Initially it is used to specify the coordinates and orientation of the ships you want to place on the board.

BUTTONS:

SEND - Is used to send messages once typed on the chat input box

LOGIN - Is the first button used since it's required to send your username and team name when you the game is being initially setup

START - Only pressed after the player logs in by creating a username and selecting team

QUIT - Used to end the game prematurely and surrender

Set Orientation - Used to set the orientation desired to place your ships on the board during setup

Set Coordinates - Set coordinates is used to specify the coordinates where you want to place your ship on the board during setup

Attack – Is used to send an attack to a particular player's board by specifying their name and the coordinate of where the attack will be made

The Game initially starts with and empty board but as the game progresses and attacks are made between teams on player boards the hits ($) and misses (*) are displayed. The ships and their corresponding points are as follows:

S = Submarine (3 points)

A = Aircraft carrier (5 points)

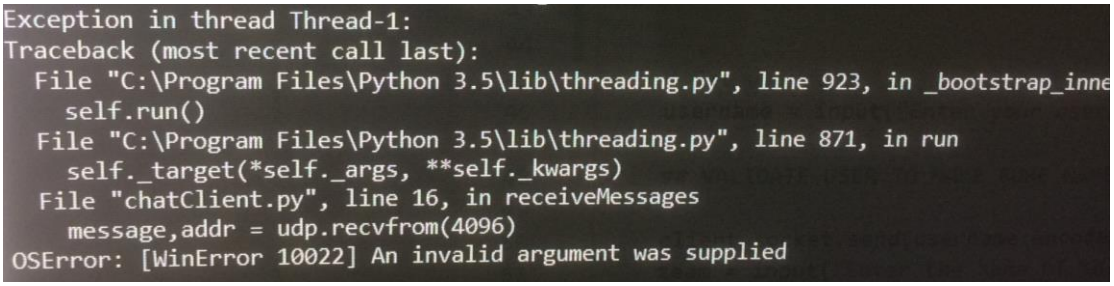
P = Patrol Boat (2 points)

D = Destroyer (3 points)

B = Battleship (4 points)

5. Analysis

Throughout the project I encountered many syntactical errors, some of which took me days to troubleshoot and detect the source of the issue. However, the most noteworthy of these was an error that stated:



This error was particularly confusing because it only appears in Windows Operating Systems. We researched the possible causes but after many attempts and trial and error we were not able to remove the error completely from the game. This is our only known glitch in Windows systems and only appears once the game is initiated at a random period in time. The nature of the errors deals with the UDP transfer protocol and its nature of being an unreliable communication protocol since some packets may never be sent or arrive and thus stalling the chat completely.

6. References

Our main reference for Python syntax was the recommended class book called:

Introducing Python – Modern Computing in Simple Packages by Bill Lubanovic

Our reference for the Tkinter syntax to implement the GUI for the client was the following link:

https://www.tutorialspoint.com/python/python_gui_programming.htm

7. Additional Notes

The general code structure for the game was to separate the game logic from the chat code as well as the graphical user interface code.