Multi Pong

A Python Game with Chat

Pachev Joseph
5699044

Alicia Fernanda Rodriguez Taboada
5162522

1. Introduction

In this project, we implemented a multi-user/multi-player game environment with a client-server architecture. This was accomplished using Python; an interpreted, object-oriented, programming language with dynamic bindings [1]. To do this, the classic arcade game Pong was chosen and improved upon to include multiple players. Additional research was necessary into three subjects to make this project successful. First, implementing Secure Sockets Layer (SSL) for secure communications, including generating self-signed certificates [2]. Next, using Pygame, a library for making multimedia applications [3]. Lastly, Tkinter, another graphical user interface for python [4].

The result is a robust server that handles multiple clients simultaneously, while also supporting chat and score tracking in real-time without the need for turn-by-turn interactions. The server handles client connections through SSL, using python's built-in library. Beneath this layer, the server connects with each client using TCP and UDP for live updates of the ball, chat, and player positions. Similarly, the clients can handle the sending up player updates, while re-rendering ball and other player positions.

2. Problem Statement

The main task of this project was implementing a multi-user/multi-player game environment with a client-server architecture, that is able to receive request from up to ten clients. The clients are users that want to connect with their friends over the internet in order to play the game of pong. The server keeps track of each player that wants to interact with one another. The problem was to implement these two applications in accordance with the guidelines from a project specification, all done in a two week period.

3. Methodology

We implemented the multi pong game with Python3, Tkinter, and Pygame. Tkinter and Pygame were used for GUI. The simplest place to start on this project was the multi-pong client. The client is the game itself. Knowing what the client is doing gave us insights into what would be necessary for the server. The client is written in python using the Pygame library. With pygame, we were able to quickly create graphical elements on the screen that are shareable across many systems. Some of the elements of the client implementation were referenced from GitHub user *smoross*[5].

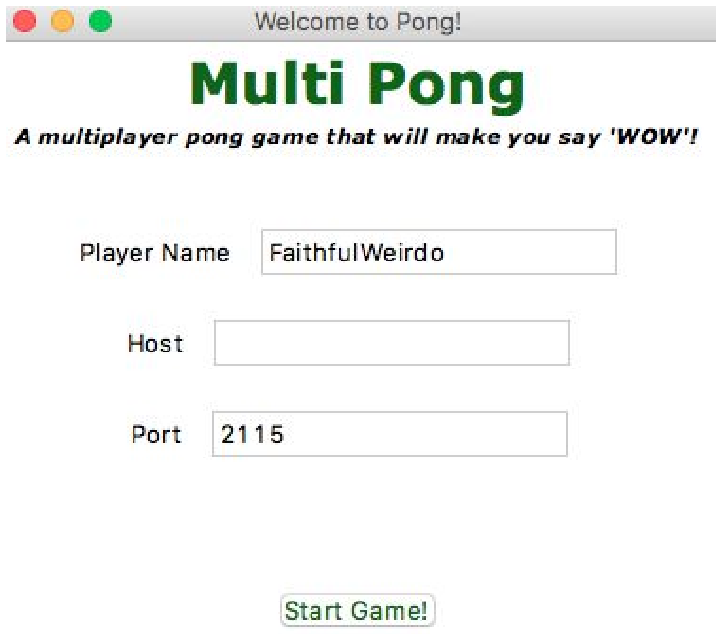
For communicating with the server, we used both TCP and UDP sockets. From the client side, the communication involves creating a socket, connecting to a remote connection, sending the data, receiving the data and lastly closing the connection. In addition to communicating with the server, the client needed to process the different packets that were sent by the server. These include player positions, ball position, score, and chat. To do this efficiently, the client uses threads that handle the interaction with the server for the ball, and other events. Threads are a small task within a process that execute code[6]

The server is also written in python and uses sockets for communication. The server needs to create a socket, bind to a specific address and port , listen and handle requests from multiple clients. We used threads to keep track of the ball, and player movements. By using threads, multiple clients and their requests could be handled simultaneously. Each time a user connects, they are given and id along with other details that are broadcasted to all other clients. From this information, the other clients are able to render this new player. The server also needed to handle a socket for the UDP player updates that were being sent from each client. Finally, we decided to move the ball logic to the server for consistency between the clients. The ball position is kept on the server, and is broadcasted to all the clients that are connected.

The final implementation was the menu and the chat system between the users. The logic for the chat system was already in place since the server and the client were already communicating with one another. However, due to the limitations of Pygame, we needed to use another GUI library: Tkinter. This allowed us to have a chat window on the side of the pong game that proved to be less intrusive to the players as they can choose to chat whenever they wanted. The menu allowed each user to enter the host and the port that they wished to connect with.

4. Results

A live video demonstration can be found at: https://youtu.be/1KAFB7u7vf4. This demonstrates the functions of the server and the client. The client can be used as a stand alone program provided that there is a server that is ready to accept and respond to its request. Figure1 shows the main screen the user sees when they first start the client. This screen allows the user to input the host, port, and choose a name if they wish. A random name is generated for each user for quick matches. Figure2 shows a two-player game between clients. The ball and player positions are being updated simultaneously on the clients and on the server. The rest of the figures show different elements of the game.


Welcome to Pong!

Multi Pong

A multiplayer pong game that will make you say 'WOW'!

Player Name: FaithfulWeirdo

Host:

Port: 2115

Figure1: Intro screen

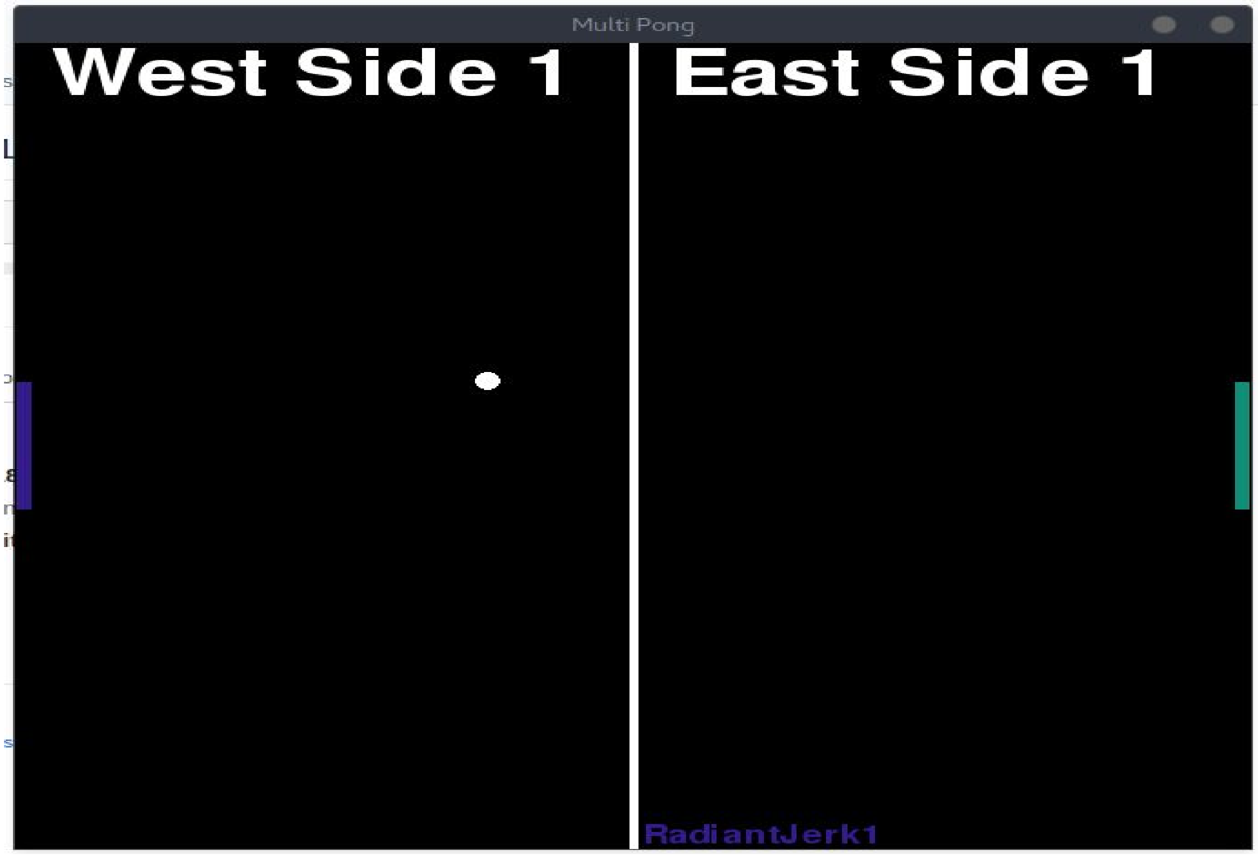


Figure2: Two-Player game

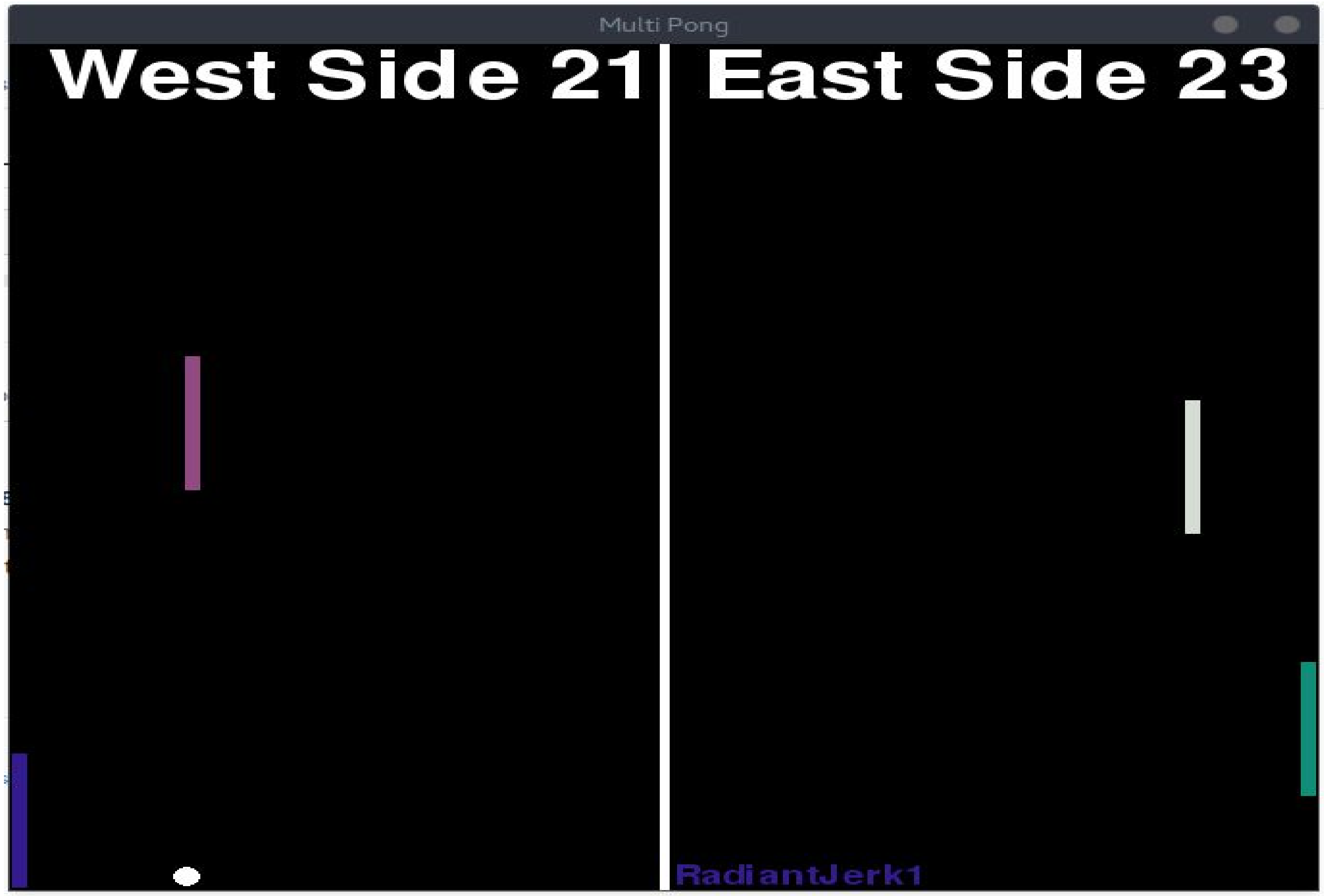


Figure3: Multi-player functionality

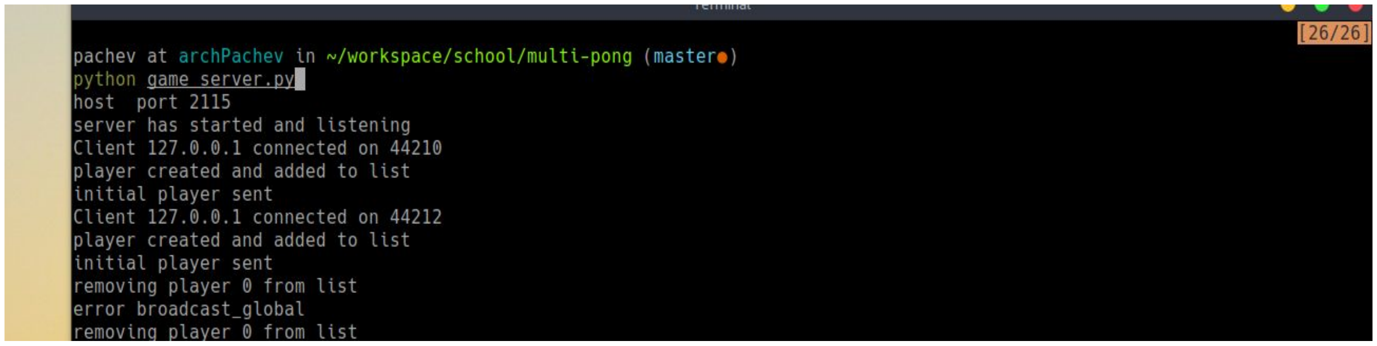

```
pachev at archPachev in ~/workspace/school/multi-pong (master●)
python game_server.py
host  port 2115
server has started and listening
Client 127.0.0.1 connected on 44210
player created and added to list
initial player sent
Client 127.0.0.1 connected on 44212
player created and added to list
initial player sent
removing player 0 from list
error broadcast_global
removing player 0 from list
```

Figure4: Server Handling player interactions

5. Analysis

The process of designing, writing, and testing the two applications took two weeks. In that time, we encountered problems dealing with the graphical elements of the game. Both of us had never worked with Python GUI tools such as Pygame and Tkinter. We were finding out the limitations of the the two as the project progressed. Initially, we were not concerned with the communication aspect of the game since we just finished an FTP server/client from the previous assignment. However, there were still problems for us to solve.

Unlike the FTP server and client, we needed to broadcast our messages to multiple clients at once. This broadcasting was essential to the game. For both the chat, and the game itself. To do this, we stored all of the clients that were connected and looped through them while sending the necessary messages. After starting with just pygame for the project, we thought the rest of the implementation would be easy. However, we ran into the problem of text input in a Pygame application. This proved to be tougher than we expected. To solve this, we introduced a new library called Tkinter to handle the chat system outside of the game.

At the start of the project, we kept the ball movement inside of each client and sent the updates to the server to rebroadcast to the clients. This led to really glitchy play due to the ball position being updated by multiple clients. So the ball physics inside of one client would try to correct the physics inside the other client. To fix this, we moved the logic for the ball inside the server. Then the server broadcasted the the ball position as it should stay consistent, given that each game arena is the same size.

The project gave us a good insight on the multiplayer aspect of online communication. Sending and receiving messages are interesting enough, but doing it in real time can be a tougher challenge. The experience of writing these two application reinforces everything that was taught in class. These include sockets, threads, tcp, udp, and ip. Overall, the project was a great learning experience in solving problems in an unfamiliar area.

6. References

[1] "Python.org", Python.org, 2017. [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed: 19- Apr- 2017].

[2] "What is SSL?", Info.ssl.com, 2017. [Online]. Available: http://info.ssl.com/article.aspx?id=10241. [Accessed: 19- Apr- 2017].

[3] "About - pygame wiki", Pygame.org, 2017. [Online]. Available: https://www.pygame.org/wiki/about. [Accessed: 18- Apr- 2017].

[4] "TkInter - Python Wiki", Wiki.python.org, 2017. [Online]. Available: http://wiki.python.org/moin/TkInter [Accessed: 20- Apr- 2017].

[5] "smoross/Pygame", GitHub, 2017. [Online]. Available: https://github.com/smoross/Pygame. [Accessed: 21- Apr- 2017].

[6] K. W. R. James F. Kurose, Computer Networking: A Top Down Approach , 6th Edition. Pearson Ed, 2013.