CS 371 Pong Project
Final Report

Caleb Fields, Ty Gordon, Abdallah Sher

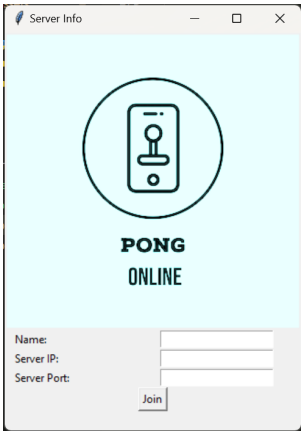
Background

The objective of this project is to create a peer-to-peer multiplayer version of the video game "Pong" using pygame and socket programming. Client-side game logic was provided by Alexander Barrera. Our responsibilities for this project include implementing a server which will sync the two clients, manage multiple pong games, and ensure smooth gameplay and performance. Additionally, opportunities for bonus points were provided. These opportunities include (those attempted in this project are bolded and italicized):

- ***Active use of a git repository***
- ***Server-side support for multiple pairs of clients***
- ***Implementation of the means by which to play again***
- ***Implementation of a leaderboard served at port 80 which displays player initials and number of games won***
- Implementation of player registration, authentication, and encryption

Design

Clients must first request connection with the server using the server IP address, port, and initials of their choosing. After the first client connects, the server should recognize that there is another client required to start the game and wait for its connection. After the connection of the second client, the server should notify both clients that the game is starting. Once this is completed, the server should continue to listen for more client connections and begin games as the client-pairs connect. As a game continues, clients should perpetually send updates to the server containing their game state (paddle information, ball information, score, and a sync variable tracking how far into the game the client is). Should one client's sync variable fall behind, the server should update the game state of both clients using whichever one is ahead. At the end of the game, the server should update the leaderboard depending on whoever won the game. Clients should then be set to a state in which they can choose to play again. Below is an image of the pong client:



Implementation

The implementation of our multiplayer Pong game is divided into two main components: the client-side and the server-side. The client-side is responsible for handling user input, rendering the game, and communicating with the server to synchronize game states with the opponent. The server-side manages connections between players, relaying game state information between them, and ensuring synchronization.

When pongClient.py is run, a start screen is created that has fields “Name”, “Server IP”, and “Server Port” and a button “Join”. The input of these fields is then fed to our join server function once the “Join” button is pressed and the screen will freeze (note: clients may connect to a server running on their machine by leaving the “Server IP” field blank). The joinServer function then attempts to create a TCP connection to the server and will wait to receive information about which side the player will play and the dimensions of the play area. If the server does not exist or there is some other issue with the connection, an error message is communicated at the bottom of the window and the window unfreezes. Otherwise, if the client is designated as the “left” player, the game will communicate that it is waiting for a second player. Once another player connects to the server, both players’ start screen will close, the game window will be created, and the game will begin.

Players control their respective paddles using the arrow keys on the keyboard and the game ends once a player reaches a score of 5. Each frame has each player send a Json dictionary containing what frame the player is on, the position of their paddle, the ball, and the score. The players then receive a Json dictionary containing the current frame, the positions of both players' paddles, and position of the ball, and the score. The clients then use this information to reset the players’ game states to be the same. After 5 seconds of the game ending, both players are returned to the start screen where they may reconnect to the server or exit.

When pongServer.py is run, the process will spin off a thread to host the leaderboard on port 80 and begin listening on port 7777 for clients. The server IP and port must be configured in the pongServer.py file. Once a client connects to the server, the username is checked for non-alphanumeric characters. If the validation fails then the server closes the connection immediately, otherwise the server creates a new thread to manage the connection between the client and the server. If there are no players waiting for a partner, the player will be designated as the left player and wait for a partner, otherwise the player will join the waiting player’s game as the right player. Once a game has begun, the players are added to the leaderboard if they are not already present and the leaderboard is refreshed.

When a client thread is created, the preliminary information is transmitted from the server to the client in a Json dictionary and they are flagged as ready to play. Once a second player has joined the lobby, the game begins and each frame the server receives a Json dictionary with gamestate data from both clients. If the server fails to receive data from a client, the connection is closed and the game ends. Otherwise, the server checks if either player is lagging behind, and

if they are then the lagging player will receive the game state from the most up-to-date player. Once a player has won the game, their number of wins on the leaderboard is incremented and the connection is closed. The server stores each players' interpretations of the game state for every instanced game in a global list. This allows data such as paddle positions to be transferred between clients, and for games to be accessible to the server.

Challenges

- Issues with Python installations hampered early development and required reinstallations and system path changes.
- Smooth gameplay was difficult to achieve, a compromise was made where the left player's ball position is used when either side needs to update. This gives the left player a slight advantage but significantly reduced rubberbanding and other synchronization issues. We deemed this an acceptable exchange as the concept of "port-priority" is present in other online games to resolve such issues.
- Division of labor was another difficulty, initial attempts to plan ahead and split the project into equal parts failed, resulting in team members identifying what needed to be done next and taking turns implementing features and resolving issues as needed.

Lessons Learned

- Our team began this project with minimal Python experience, the language itself is designed to be intuitive but small quirks of the language only became apparent once development was underway. Our team is now fairly comfortable writing Python code.
- The creation of a server-client relationship is something no one on the team had experience with prior to this project. Our team can now develop network-based solutions.
- Our team began working on the project at the end of October, although work was done on the project until the due date, having already completed significant sections of the project made completing the project significantly less difficult and stressful.
- Our team became significantly more comfortable using Git/Github for team project development as a result of this project.

Known Bugs

- The leaderboard still updates the score of a player even if they change their name.
- Some stuttering issues may arise depending on the quality of the network.
- Player disconnections can lead to undefined behavior.

Conclusions

Despite the challenges, our team was able to assemble a working peer-to-peer multiplayer pong game using TCP socket programming, python and pygame. Many of the tasks required us to use frameworks and applications that we haven't used before, and thus we had to adapt and learn. This project has broadened our skill sets and improved us as software developers.