Pong Project Report

Tharanie Subramaniam and Joshna Kurra

Background

This project modified the classic Atari game Pong into a multiplayer game with a client-server architecture. This involved creating both client and server logic that allows players to compete against each other over a network connection. The code given in the zip file includes all of the game logic, but the server and client logic needed to be added in order to allow the game to be playable over a network.

Design

We designed our implementation by first considering what the server and client were responsible for. The client would need to be able to communicate with the server to send information about its current game state and receive information about the other player's game state. The server would need to be able to simultaneously receive data and send data back to the clients. We used a TCP connection for this game as this ensures a reliable and bidirectional connection. To ensure that the game is synchronized, the server would also have to check which client's data is most updated. This way, each client would be sent the correct game information each time the server updates the clients, keeping the two clients synchronized. We also needed to consider what data structure would be best for this communication.

Implementation

To start off, when the client program is run, it prompts the user for an IP and port. The user inputs this and clicks the join button to join the server. Once both clients are connected this way, the server uses a client handler function to first assign a paddle side for each client—the first client to join is always left whereas the second client is right. Each client then calls the play game function with the paddle side received from the server. In the function, each client sends the server its game data as a dictionary that is converted to a JSON string. The dictionary includes data such as score, player paddle location, opponent paddle location, ball location, etc. Part of this data is the sync value, which gets compared with the sync value of the second client. Whichever client has the greater sync value is used to populate the most up-to-date game data in the server. This updated game data is then sent back from the server to both clients where they update their own game state variables. This continues back and forth until one player wins. The handle client function which is passed into each thread takes care of all the back and forth communication. The diagram below in Fig. 1 was created to demonstrate the client-server interactions and the flow of our code.

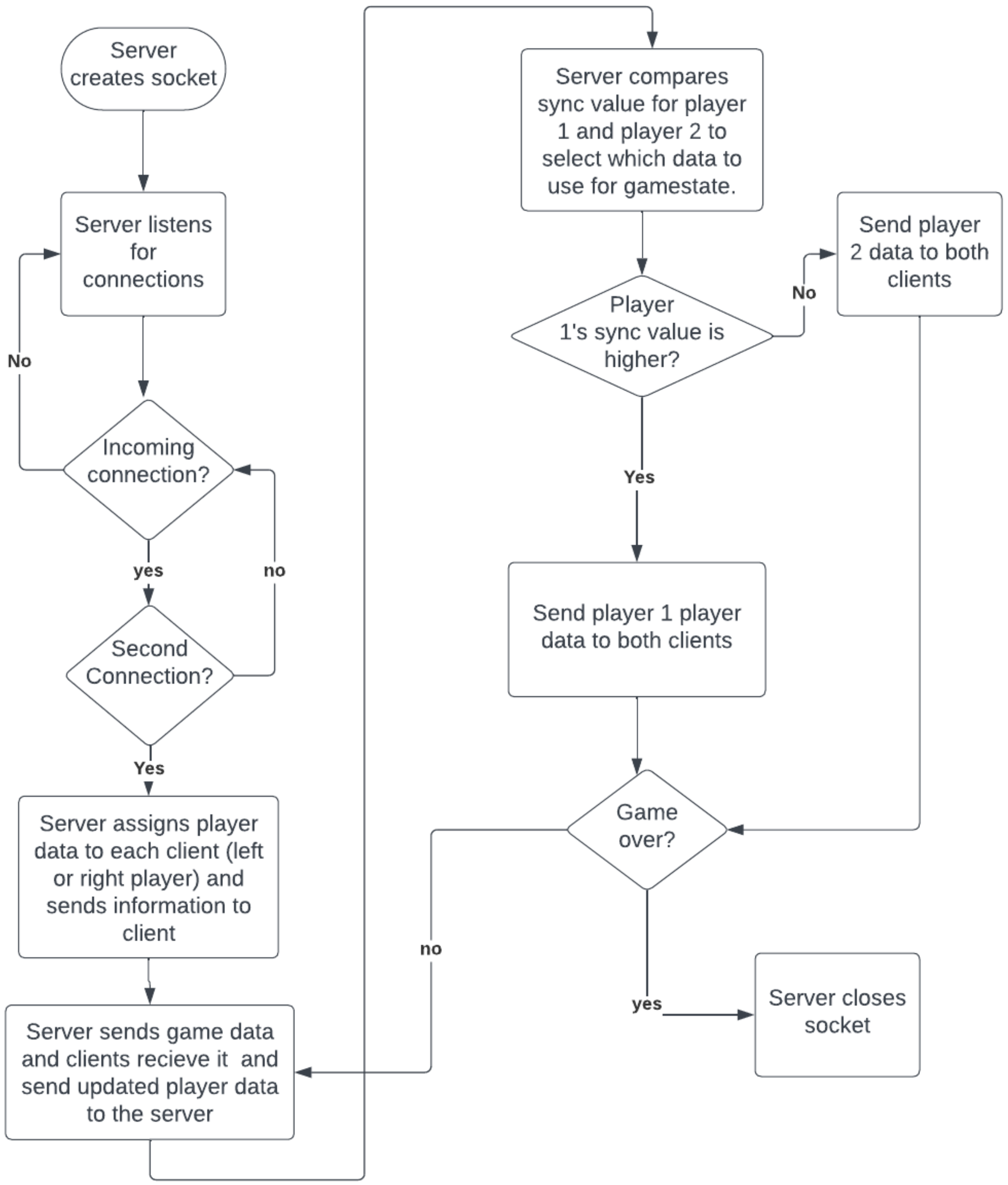
Server creates socket
Server listens for connections
Incoming connection?
No
yes
no
Second Connection?
Yes
Server assigns player data to each client (left or right player) and sends information to client
Server sends game data and clients recieve it and send updated player data to the server
Server compares sync value for player 1 and player 2 to select which data to use for gamestate.
Player 1's sync value is higher?
No
Send player 2 data to both clients
Yes
Send player 1 player data to both clients
Game over?
no
yes
Server closes socket

Fig 1. Flow chart for our implementation

Challenges

One of the main challenges that we ran into during this project was using a MAC and Windows computer to test our code. At first, we were running into graphics problems on the Windows computer we were testing with, as the screen and paddles were distorted. This was fixed by changing the pygame.display.update function in the client code. Towards the end, we ran into a problem where when one client was a Windows computer and another client was a MAC computer. Both players would be on the left side of the screen instead of one player being left and another player being on the right. Debugging this issue was initially unsuccessful, as we found the correct paddle side was being assigned but the display was simply not reflecting this fact. We then deciding to try to test our code on two Windows computers instead, and game worked as it was intended to. Therefore, we found the Windows clients can only play with other Windows computers. We have not tested this, but it can be inferred that MAC clients can play the game with other MAC clients. However, a MAC computer can host the server for Windows clients with no problems. Figuring out how these operating system incompatibilities was difficult for us, but we adapted to this problem by trying many different versions of running the code, and sending the data in order to find the source of the problem, and ultimately testing it on different devices which helped us solve it.

Another challenge included managing the threads that represent the players. Originally, we had a while loop in which both the threads get assigned. With this approach we were running into a JSONDecodeError. But, we checked everything with our JSON thoroughly and nothing seemed wrong. However, in this process, we found that sometimes there is an empty string being sent over. We wondered if this was because of the threading and decided to make two separate thread variables rather than run them in a loop. This fixed the empty string error, and the data was successfully being sent back and forth.

Lessons Learned

A major lesson that we learned from this project was that in socket programming, it’s very important to properly plan the code and make it very clear where and when data is being sent and received—to make sure that each client and server interaction is accounted for and done correctly. We did not expect there to be as much tedious planning and tracking of data flow as there was. We also learned how to use TCP and thread programming in order to connect two computers and send data over a network. We learned how to work with and handle the threads properly in order to make the game successful. From this project, we also learned how to use GitHub for collaborative programming, so that we were able to see each other’s changes as well as different version of our code as we worked through the projects. We also learned how to use Visual Studio’s debugger in order to track what’s happening in our code and find out what might be causing issues.

Known Bugs

There are a few bugs present in our code, none of which significantly impact gameplay. Every couple of frames, the ball has fluctuations that cause it to go slightly in unexpected directions. The overall trajectory of the ball does not change, resulting in the ball simply shaking a little bit as it moves across the game window. This does not make the game unplayable, and even adds a little extra challenge for the players. There also is a little bit of lag present while playing the game at times, but this lag also isn't enough to make the game unplayable. Both errors may be fixed by optimizing the code and trying to make it more efficient, which could potentially decrease lag and prevent the ball's slightly sporadic behavior. We could try to minimize the data that is sent through the network and only send the most essential information to reduce network traffic which could increase performance. We could also try to use a network protocol that has less overhead such as UDP, which could potentially be faster and prevent lag. However, UDP is less reliable than TCP, so there may be more packets lost and glitches, or the synchronization capabilities of our program would have to be relied on more heavily. Another bug in our program is that even though the game is functional, the server produces a JSONDecodeError that we were not able to debug. In order to fix this, we could continue to debug and trace where exactly this error is coming from.

Conclusion

In summary, we were able to successfully modify the pong game in order to allow 2 clients to connect to the server and play against each other. We used TCP to make the connection and send data over the network, and several server and client interactions were implemented. This allowed data to be sent and received between the clients and the server, meaning each client could receive data sent by other clients through the server's services. We implemented a synchronization feature to make sure that the two clients were seeing the same game state as they played each other. Our game is usable for 2 clients that are both Windows, or both MACs, and the game has room for improvement as the ball movement isn't always very smooth, and there is some lag present, but overall, the requirements were fulfilled. We also used GitHub while making our project, and saved our work throughout each stage through many commits. In the future, we could optimize our code to improve the game for a better user experience. This could include adding features such increasing the number of clients, adding a menu, and a start game screen, and increasing the range at which clients can connect to the network from.