CS 371 Pong Project

Will White, Luke Olsen

Background

This project implements a multiplayer version of the classic Atari game Pong using a client-server architecture. Pong is a two-player sports game that simulates table tennis, where players control paddles and try to score points by hitting a ball past the opponent's paddle. In this project, the traditional Pong game is modified to allow players to compete against each other over a network connection. The solution involves designing and implementing both the client and server logic using socket programming techniques.

Design

In the client-server architecture we designed for this game, the client is responsible for sending the local player's paddle position to the server and receiving information about the opponent's paddle, ball location, and the game score. The server, on the other hand, handles communication with two clients simultaneously, utilizing threads to manage the parallel interactions. It relays information about the opponent's paddle, ball, and the score to each connected client, ensuring a synchronized multiplayer gaming experience.

We designed the server to allow two clients to connect seamlessly and initiate their own thread within the program. This approach ensures that the server can effectively manage concurrent interactions with both clients, providing a robust foundation for parallel communication. The design not only allows for simultaneous connections but guarantees the autonomy of each thread, enabling efficient and independent handling of their respective game states.

We designed the client to be able to seamlessly connect to a given server. It receives its paddle assignment from the server and sends its game state information via update packets. These packets contain game information in the form of JSON dictionaries encoded as strings, facilitating efficient transmission to the server. The server, in turn, broadcasts this information to all connected clients. The client receives game state information from the other client (player) and unpacks it to determine the most up-to-date game state information. In tandem with this, a synchronization variable is utilized to ensure that both clients are receiving the current state information of the game, including elements such as the score, the ball location and velocity, and the location of the opponent's paddle.

Implementation

When each client connects, the server creates a thread for the client to handle game information transfer. Depending on the order in which the client connects, it will be assigned a paddle, either "left" or "right". If it is first to connect it is assigned the left paddle; if it is second to connect it is assigned the right paddle. This paddle assignment is sent as an encoded message back to the client, which is then passed into the playGame() function, so it knows what paddle it is using.

The server receives game information from one client and sends it to the other client. The data contains information such as paddle position, ball position and speed, scores, and the synchronization value. Game state information is prepared and stored as a JSON dictionary, but is converted to a string format to be encoded and sent to either the server or the client, depending on where the information is coming from. When each client receives the other's game state information, it updates its own local data for the opponent's paddle position, the sync value, and

the scores. It compares the received synchronization value with its own, updating it to the client's sync value which is higher to remain in sync. This synchronization value is stored as a variable "sync". Upon receiving the game state packet from the other client, the current client checks to ensure all game state fields are present in the JSON dictionary, and updates its game state values accordingly.

To handle the ball position updates, one client is designated as the "host", meaning that client's ball information takes precedence over the other. When the non-host client gets the opponent's game information, it updates its own ball information to match that of the host. We decided to implement it this way because we know that the ball information should always be the same in real-time for both clients.

Challenges

One of our initial challenges was in synchronizing the paddles so that their movements could be viewable on each client's screen. This was the first field of the game state information we attempted to synchronize, so finding an effective solution felt rather complicated and required a lot of brainstorming. We utilized a drawing board (MS Paint) to visualize how the packets were being sent across the network and decoded at the server, and subsequently encoded to the other client. From this, we were able to devise a solution involving a JSON dictionary to unpack the paddle data, as discussed in our implementation. Our last main challenge was synchronizing the ball position for both clients. The game was functioning almost perfectly, but we found the ball positions and velocities would diverge if a player hit the ball while their paddle was in motion. We solved this issue by designating one of the clients as the "host" and forcing the other player to use the host's ball information at all times.

Most of our challenges revolved around synchronizing various game state fields, but once we had a framework on how to handle the paddle information, it became more apparent and intuitive on how to design implementations to handle the other data. To these examples, we think visualizing how packets and their contents were being transferred and unpacked across the network helped us in creating solutions for our problems.

Lessons Learned

1. How to send information across a network using sockets
2. How to establish a connection to a server via a python script
3. How to utilize custom python libraries and modules

Conclusions

In conclusion, our implementation of a multiplayer Pong game through a client-server architecture demonstrates successful socket programming and effective management of concurrent connections. Overcoming challenges in synchronizing paddle movements and ball positions, our design, utilizing separate threads for each client and designating a "host," allowed us to create an interactive multiplayer experience for the game. This project has allowed our team to gain insights into network communication and associated python scripting, and we have developed a strong interest and beginning foundation in game development and network applications.