CS371 Pong Project Report 11/25/25

Authors: Ben Carey, Christian Brewer, Caleb Burnett

Background: Briefly describe the problem you are solving.

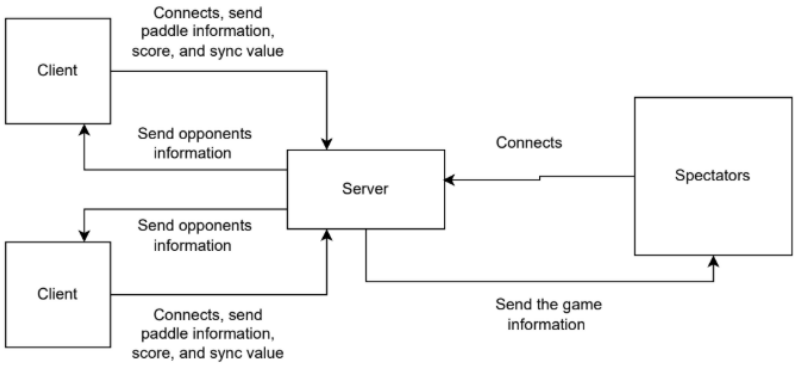
The project is a multiplayer Pong game with a client-server architecture. The problem to be solved is to use socket programming to enable the clients to interact with each other synchronously through the server, allowing the basic rules of Pong to be followed. Therefore, the main problem is creating client and server code using sockets to ensure this is actualized.

Design: Describe how you designed your implementation (design before you start coding).

Because the files provided already implemented client-side code, we decided to save labor by moving forward with that model in an effort to save labor. We next discussed what bonus points we thought would be simple enough to include, and what we would need for the basic functionality to be actualized. We thus used pair-programming to brainstorm what helper functions and logic would be needed for interaction between the server and client. Ultimately, we settled on clients sending their sync, paddle position, ball position, and scores to the server. The server would then pass those along to each client connected, allowing them to update their instance with information about their opponent.

This was implemented by having the server as a "traffic light" for the two clients. Once it receives a packet from a client, it has that client wait until a packet is received from the other client. Once that is received, the server permits both clients to proceed to the next game frame.

Implementation: How was your design implemented (a UML diagram and/or pseudo code may be helpful here)



The host is started first, which waits for a package from the client(s). When a package arrives, the server opens a new thread to handle the client and also chooses to assign the first connected client as the left paddle. The next client becomes the right paddle, and if more join, they are considered spectators. The server then, is the middleman between the clients and helps preserve synchronous functionality of the game logic and broadcasts the game to spectator clients. This branching allows for sending smaller packets which are only relevant to the intended recipient.

The designation of the server allows each client to know its own place in playing the game; thus, each client can conduct itself differently using code branching from if-statements. After a client connects and becomes a player, it receives data packets from the server and also sends its own data packets which include its identification, and the ball/paddle positions, as well as both scores, and its sync number. This allows the server to arbitrate sync and scores by updating values based on the highest common value among that category. Non-player clients branch into only receiving information about the other client's positions.

Challenges: What didn't go as planned and how did you adapt?

We initially wanted to send one packet to the server for every game frame. It would combine game state, like paddle location and score with the sync variable used for synchronizing the clients. However, it became apparent that these packets should be sent separately. This would allow the server to update the opponent with the game state that occurs before the next frame, allowing for more accurate collision detection along the edges of the paddle.

There was an issue with smudging white trails behind the paddle and the ball movement, which was solved by including a pygame refresh display command to

remove previous position drawings. We also reduced our goals from all bonus points to those easiest to implement when we realized how involved some of those goals were.

Lessons Learned: What did you not know before starting this project that you now do?

We learned that LAN hosting is possible on the eduroam network. The intricacies and pitfalls of working with the socket library.
We gained a better understanding of what desync and "lagging" are in the environment of online gaming.

Although we had previous experience with git, some members were able to impart knowledge of its use to others in a helpful way, as well as personal coding practices. Therefore, we learned from each other as well as from the project.

(Optional) Known Bugs: Document the bugs you were unable to fix here and how you would solve them given more time. You'll still lose points but not as many.

• It crashes when a client disconnects from the server.
• The clients that are spectators have sync issues.

Conclusions: Summary of the work and final considerations.

- Sockets
    - Sockets are an amazing tool we were pleasantly surprised to learn, and we will likely use it in future private projects to create applications with network connectivity.
- Protocol
    - We were able to form a packet and use string parsing to extract values, rather than a JSON.
    - We used conditional packet forming to make use of our user-to-user code, allowing spectators to have a connectionless oriented interaction with the host. The first two users connect, being the players, and therefore requiring the handshake of a connection-oriented protocol. We found this to be an interesting and more efficient method, and reflective of real life.
- Syncing
    - To ensure valid game state across the clients, we used a syncing mechanism that has clients' game state wait on an ACK from the server

before progressing. We implemented it in a multithreaded way so that the window remains responsive while the game awaits opponent client data from the server.

- Group dynamic
    - We had a good group with a good work ethic, whose strengths made up for the weaknesses of other members. We enjoyed working together, though we found remote sessions more productive. All members were actively participating and contributing their best to the project.