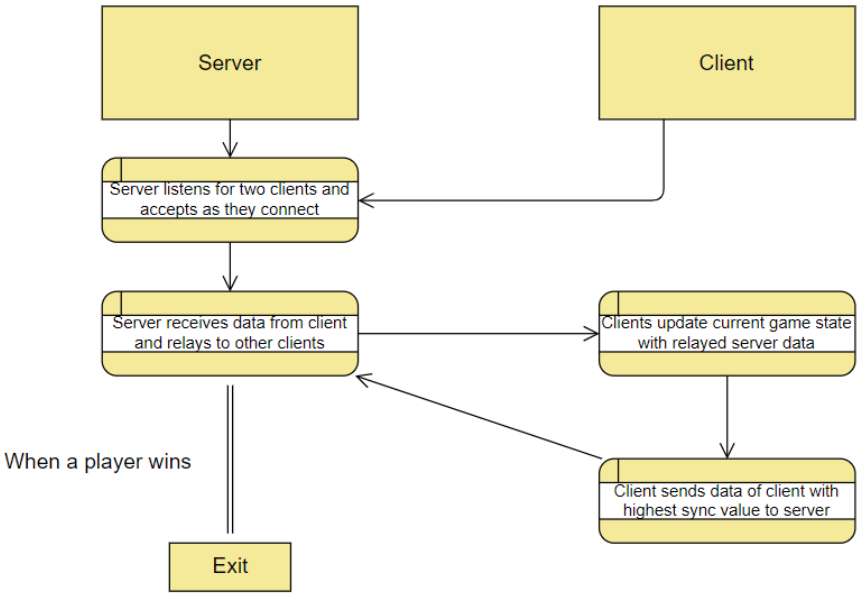
Pong Project

Background: Given a Pong client we have been tasked to create a server that would receive, relay, and synchronize the game state between (at least) 2 Pong clients using sockets and threading. Allowing the same game to be played on two different clients, each with their own paddle, against each other.

Design: The plan to implement prior to coding has been to create a server using sockets that can handle two clients using threads. The server will send the initial game data to the clients and from there the clients and server send and relay information back and forth maintaining a synchronized state throughout connected clients.

Implementation:



Challenges: Sending the relevant game data from the clients to the server back to the clients was quite a challenge. It seemed as if every time we troubleshooted the code to fix this, other parts of the client code would break. Once we eventually overcame this challenge, we noticed that the clients were not synchronizing correctly and needed to do further research on threading. Finally, once everything was working as expected; the output seemed to move slower than what was desired.

Lessons Learned: We have gained a better understanding of networking as a whole and how a server may communicate with clients to relay information. In addition, we got more familiar with the socket networking interface and threading. I believe this project also taught us to use good convention and best practices by requiring type hinting and encouraging version control giving more experience and familiarity with the industry's current best practices. Working on this project felt like working on a real-world problem.

Known Bugs: The Server does not update slow enough which causes the score to be nullified whenever the ball goes to the right or left side. After implementing many different thread locks to prevent this we

where unable to find out the cause. Other attempts were made to solve this problem, we input a sequence number into the Client messages so that we could keep track of the order in which they arrive. At one point we used “time.sleep(0.1)” to see if a small buffer would allow more efficient processing of data but all that did was make it slower. That is the main bug that we where unable to solve. Before we had a bug that would overload the server with data, causing huge strings to be sent back and forth. To solve this problem we introduced input_locks in the server that prevented it from intaking more than one packet at a time.

Conclusions: To implement the pong game we utilized the provided sync mechanism along with the TCP protocol. With the server being a buffer between the two clients controlling what data is sent back and forth. The main problem we had, like all networks, was handling the packets and finding ways to handle lost or invalid data. When packets where sent to back to the client a lot of times there would be two game states in one string and adding a “\n” to the end of the packets and discarding everything after the “\n” really helped with this problem. Although we where not restricted to the paradigm provided we figured sticking with it would give us good insight on how problems are solved in real-world networking cases. If we would have abandoned the sync protocol it would most likely been easier but we would not have learned about what works and what doesn’t as much. This project was great practice for getting to know the basics of a small network.