**CS371**

**Final Project Report**

**Jacob Blankenship, Daniel Krutsick**

**Background: Briefly describe the problem you are solving**

In general terms, our task was to take a client code shell with some Pygame assets and functions and convert it into a fully functioning Pong game. This game connects 2 users (clients) to a host (server) by having the clients pick a predetermined IP address and Host Number (from the server) and input into a prompt. After 2 clients are connected a game ensues between the two users. Users interact with each other through a ball and paddle that is received and sent through data to and from the clients and server.

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

We set up our gaps and requirements into stages and put them into baby steps. These general implementational steps were changed as the project went on. They go as follows:

1. Get both members to both access the code on Visual Studio Code, install Pygame, and add the coding space to a GitHub repository.
2. Have one client connect to the server and have both of them able to send/receive data.
3. Have one client connect to the server and have it able to send parsed game state data to the server, and have the server send it back.
4. Have two clients connect to the server and have game state data sent to the server and have both clients' data broadcasted back.
5. Allow for processing of the game state data received by clients, allowing clients to discern received data from themselves and from the other player (Left vs Right).
6. Fix technical issues, such as synced data being received incorrectly/at wrong times, or being able to remove clients from server correctly so game could be efficiently restarted.

**Implementation:** **Diagram is in the files that are zipped together and submitted on Canvas**

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

One of our biggest issues was with our send and receive and how to handle tick desyncs. We had multiple different implementation, such as adding a queue and buffer to handle multiple inputs, sending both data sections at the same time instead of one at a time, a client lock to ensure no race conditions happen when modifying global variables accessed by multiple threads, and a threading event flag created to ensure that two clients have been connected to the server, which then allows the server to send a message out to allow the game to start on both ends at the same time.

Another thing that didn't go to plan was ease of testing, running code usually for our team is just running the file alone. However now, a minimum of 3 terminals need to be running code, inputs take more time as well. Another thing to add is that initially we had no error checking, so if our code didn't work as intended, we would have to close all terminals and start again. We also had an issue with freeing ports which caused errors when running with the same IP+Port (this took a while to realize).

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

- We had both used threads in the past, however, not nearly to the practical extent in this assignment. I think we now have a better understanding of thread utilization.
- Another thing that was new was utilizing a code shell that was already there, which was something that we had done in the past but not for a fairly long time.
- We also didn't know anything about actual Pygame implementation. Neither of us had actually used the package, and while we didn't have to create Pygame elements at all, we did have to understand what the code did beforehand (drawing new rectangles, x + y movement).
- We also learned that through the implementation of our server code, it is not easy to figure out which threads should be done at which times. The complexity of this issue

took a while to get the server working, given we were trying to make it as solid as possible.

- We had learned about using threading events, rather than global Boolean variables, given that we wouldn't want multiple threads to accidentally write to the same variable at the same time, and we also had learned about using client locks to help prevent race conditions and other issues including the clients and server sending and being sent data at weird times.

**Known Bugs:**

- There is a possible slight desync issue that resolves quickly where the ball will move around in a weird pattern. The user that is behind will synchronize itself with the user with the higher time, as it is the user who is more ahead. This will cause the ball to sometimes weirdly move its path as it is updating on the user's end and causes slight issues, but in general it is not game breaking. We don't know an exact fix for this "bug", but if given the time would look into debugging routes to find exact sections of code that are causing the issues.
- Another "bug" is that the server will not actually account for spectators but given this was a bonus and we didn't have enough time to implement it properly, it is not yet completed. Although parts of the code account for any spectators, if a spectator were to join the mid-game match, it would not properly be sent the game states. The client side would also error out because there is no ability to run PlayGame() as a client. The server/clients will also not wait for spectators if there's 2 clients already connected. To fix, we would just add spectator handling for the Client and the ability to wait for a spectator to join before the game starts.
- Another bug within the code is that given the client wants to close its own connection, it will not be allowed to, as there is no code in client sending a message to close its connection on the server and the server has no exception to detect any clients wanting to close down early. This does not account for the fact that the server does however close down connections properly without issue, just the client cannot initiate a

shutdown of its own connection unless the game is complete or the server is forcefully closed down with a Keyboard Exception(ctrl+c). This would never need to be run naturally, and if you want the Client code to terminate, the keyboard exception for the server will terminate it as well. To fix we would just add in the keyboard exception. Overall, none of these bugs should impact performance in any way but are things that could be fixed if we had more time and wanted to increase the accessibility of the code.

**Conclusions: Summary of the work and final considerations:**

Overall, our group, considering only having 2 members instead of the typical 3 did fantastic. Both of us understand our code to a high degree and felt that implementation was difficult but manageable. We believe that our code in a simple, understandable, and working fashion does what it needs to do. Our reasoning is simple and throughout the project our implementation greatly. While we would love to finish many other aspects and add-ons to the code, we feel this is a good stopping point given the constraints and requirements for the project.