CS 371 Pong Project

Caroline Waters & AJ Wyatt

# Background

The problem at hand is the design of a multiplayer Pong game utilizing client-server architecture. Players (the clients) play the game over a network, facilitated by a single server managing the client connection. The implementation of this architecture is done through Python socket programming.

# Design

We chose to design this project using a single server handling two client threads. Each client has its own information about game state and synchronization, which is sent to the server. The server then determines which client has the most up-to-date game state and corrects the state and synchronization accordingly. Only the clients are responsible for playing the game. The server manages both client threads (and client connections) simultaneously.

# Implementation

# Challenges

The primary challenges for implementation were our newness to Python and the Python socket library. Small differences between functions like send() and sendall() often had large impacts on functionality, and documentation was sometimes lacking in specifics. Another hurdle was figuring out how to send and receive the game state values, since the code examples we had been given in class just involved small strings.

General debugging was made more difficult due to the nature of the project. Trying to run the Visual Studio debugger on two separate clients simultaneously often yielded misleading results, since the synchronization could change based on who stepped through the statements faster. The threading also made errors more confusing since it was hard to tell if/where an error occurred in a thread. When running two clients and a server program, performance tended to vary based on if each program had its own device versus two clients being run on the same computer. Similarly, some errors would only occur on one device. The synchronization might work on the client that joined first, while the second client would not work at all.

# Lessons Learned

We both have gained a significant understanding of the client-server architecture on a lower level. Being able to tinker with the socket library revealed a lot about what goes on “under the hood” of a client-server connection, like the actual sending and receiving of data through buffers, or how connections are established and handled. The task of implementing a synchronization strategy for the two clients showed us more about how real-time connections between multiple clients are maintained, and we were able to apply our understanding of connection-oriented reliability (or lack thereof) to a concrete example. Additionally, the use of threads in a more practical context than previous class examples helped us draw more conclusions about how threading is utilized in real life applications.

# Known Bugs

# Conclusions