Pong Project

**Background:** In this project, we are creating a multiplayer game of the Atari classic video game “Pong”. This project is done using client-server architecture and socket programming. All programming for this project was done in Python.

**Design:** We decided to make a list of the criteria necessary for the two parts of the project (client and server). The client is responsible for communicating with the server to relay and receive information about the current game state. The client needs to constantly send the user’s paddle location to the server. The server is responsible for communicating with two clients simultaneously by using sockets. The server implementation needed to use threads for the clients. The server is responsible for relaying the location of the opposing player’s paddle, ball, and the current score. Once we itemized all required features for the assignment, we decided to each start working on a part of the project to make faster progress and finish the assignment sooner.

**Implementation:** Ultimately we decided to split the game logic into its own file named gameInstance.py. This file does the heavy lifting for this assignment and contains methods for tracking the ball’s position, velocity, player position, score, and sync value. In addition, this class also contains methods for the server to send to the clients so that the game remains in sync. The pong client file contains a class responsible for joining the server, running the main game loop and executing the start screen for the game. Below is a UML class diagram to give a better understanding of the files and the methods that they contain.

**Challenges:** We realized that our initial approach of tackling two different aspects of the project simultaneously was more difficult than we expected. In the beginning of the development phase, we found ourselves committing to the GitHub repository at different points which would cause issues with our local repositories being several commits either ahead or behind the main branch. To resolve this, we agreed to notify one another whenever a commit to the remote repository was pushed so the other person could pull the latest changes and continue working. The next challenge that we faced was that our clients would freeze once they connected to the server. In order to resolve this we started commenting out portions of the client code that we wrote to narrow down where the crash was occurring. Most of our challenges for this project stemmed from issues relating to implementing the various aspects that involve synchronizing the game. The easiest synchronization feature was the score since its mostly static throughout large parts of the game. Paddle and ball position were by far the most challenging area of the entire assignment.

**Lessons Learned:** Our team learned the importance of planning a project rather than jumping straight into coding without direction. We also realized that constant communication is crucial when collaborating with a team to reduce duplicate code, or a complete rewrite of code based on different implementations.

**Conclusions:** Upon completion of the project, we realized that even a simple multiplayer game hosted locally for multiple clients can be a tricky task. In the end we had to refactor our whole codebase, and created utility functions to improve the code for sending/receiving messages and reduce errors. We also used dynamic message lengths by appending the length to the first bytes of the encoded message and reading it back to determine how much to read. we also broke out each instance of a game into a GameInstance so that multiple games can be running concurrently. This way of restructuring the code also made it far easier to implement the bonus features