Pong Project Report – Maggie Bacon and Emily Behrendsen

Location: https://github.com/emilyanne938/Pongproject

Background:

The purpose of this project is to implement the game pong with simple client server architecture. The project gave us the pong game logic that can play on one computer only, on one side. Using a server, two clients on different computers can play the game together while keeping them synchronized.

The task of the server is to handle simultaneous communication between the two connected clients. The two clients require the server to send the others ball location, paddle locations, and score to ensure their games run smoothly and in sync.

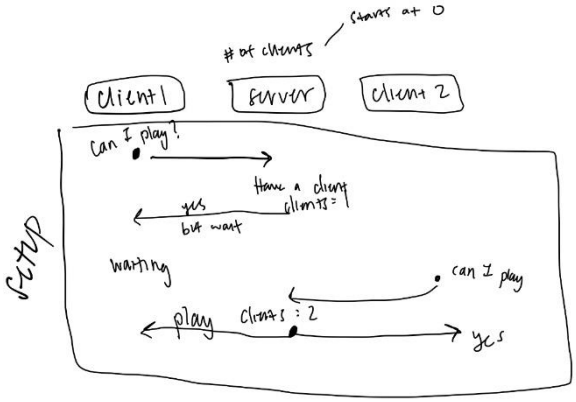
The client's main job is to send their individual game states to the server. It is responsible for the location of their own paddle, the ball, and the current score. It will receive the location of the player's paddle, and the most up to date game state.

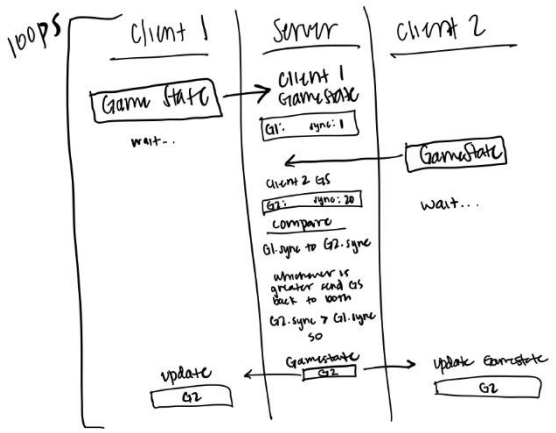
Design:

For the design of our implementation, we drew out a couple diagrams that we could follow as we wrote the syntax. This helped us not overcomplicate the goal we were trying to accomplish. Also, we were able to stay focused and understand visually where we were at in the process.

Figure 1

This image to the right shows the simple client server connection process. It shows us what messages are needed to be sent during the setup. It also helps us keep track of when the variables should be updated.





This image to the left shows the client server connection process as a whole and how the sync variable with updating the game state is going to fit in to that connection.

Figure 2

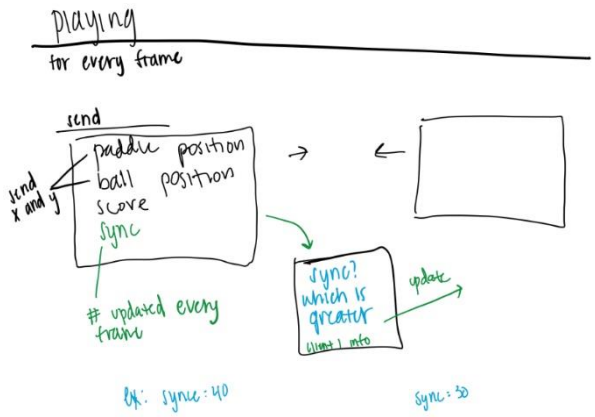


Figure 3

This image to the left shows specifically how the sync variable was going to work. It shows us what needs to be sent every frame and how the sync variable would be used. We know that whichever client has the greater number for sync will be the one that is updated.

Before starting the project, we knew the main goals were to be able to have two clients have the ability to connect to the server and play a ping pong game that was synchronized to each other.

Implementation:

The diagrams shown above helped us write the function down below. Implementation was challenging to figure out how exactly we wanted to break down the steps we knew had to be done. I've broken this down into the server implementation and the client.

For the server, we wrote createserver(), connection(), handleclient(), and main() to create and maintain the connection between the server and to two clients and update the game state. What each function does is shown below:

```
Function createServer()
   Create a TCP server socket
   Set socket options for reusing address
   Bind socket to IP and port
   Return server socket
End Function
```

```
Function connection(clientSocket: Socket, clientList: List of Socket)
   Initialize screenWidth to 640
   Initialize screenHeight to 480

   If length of clientList is odd
      Set playerPaddle to "left"
   Else
      Set playerPaddle to "right"

   Send screenHeight to clientSocket
   Receive acknowledgment for screenHeight from clientSocket

   If acknowledgment is "height_ack"
      Send screenWidth to clientSocket
   Receive acknowledgment for screenWidth from clientSocket

   If acknowledgment is "width_ack"
      Send playerPaddle to clientSocket

   If playerPaddle is "right"
      Send "go" signal to client in clientList[1]
End Function
```

```
Function handleClient(clientList: List of Socket)
   Declare global variables client1gameState, client2gameState

   Loop indefinitely
      Receive gameState from first client in clientList and split it
      Receive gameState from second client in clientList and split it

      Compare synchronization variables in both gameStates

      If first state is ahead
         Update paddle positions in first state
         Send updated first state to both clients
      Else
         Update paddle positions in second state
```

```
        Send updated second state to both clients
    End Loop
End Function
```

```
Function main()
    serverSocket = createServer()
    Call serverSocket.listen()

    Initialize maxplayers to 2
    Initialize currentNumClients to 0
    Initialize clientList as empty list

    Loop until quit condition
        Accept new client connection in serverSocket
        Add new client to clientList
        Receive message from new client

        If message is "Can I play?" and currentNumClients is less than or equal to maxplayers
            Increment currentNumClients
            Call connection(serverSocket, new client, clientList)
            Start new thread handleClient with new client and clientList
    End Loop

    Close serverSocket
End Function
```

For the client, we added the following code to playGame() to send the server their game state so the server can keep the two clients synced together, and joinserver() request access to the game and receive the screen size and left or right paddle assignment.

```
def playGame(screenWdith, screenHeight, playerPaddle, client):
        *game logic inserted here

        sync += 1
        gameState = "ballX, ballY, playerPaddleX, playerPaddleY, opponentPaddleX,
                opponentPaddleY, leftScore, rightScore, sync, playerPaddle"
         send gameState to client
         recieve gameState from server

        if playerPaddle == left
```

```
                update all variables in GameState to player 1
        else
                update all variables in GameState to player 2
End Function
```

```
def joinServer(ip, port, errorLabel, app):

        create socket
        connect to server

        player1 requests to play - client sends message

        recieve screenHeight from server
        send height ack to server
        recieve screenWidth from server
        send width ack to server
        recieve paddle from server

        if paddle == left
                errorLabel = Connected and waiting on player 2.

        app.withdraw()
        playGame(screenWidth, screenHeight, paddle, client)
        app.quit()
End Function
```

Challenges:

Many challenges were presented throughout the course of working on this project. At first, we had to overcome the challenge of simply understanding what our goal was and how we needed to act. To overcome this, we drew out a picture of what our implementation could look like. This helped us stay focused and organized when writing our code. The images above, under the design explanation, show the diagrams that we followed.

The typical challenges of working with a group did come up in a few areas. With very busy schedules, we had trouble finding time to work together and making sure we were both on the same page at all times. To overcome this, we made a schedule of times to meet and made sure that if one of us worked on the project separately, we communicated the changes and progress that were made. Additionally, we used GitHub to push and pull the changes that were made within the code. GitHub was a great resource for our group to utilize.

The biggest challenge that we ran in to was feeling that we had a good understanding of what was supposed to happen, but not understanding what was wrong with our syntax in the code. This specifically happened when working with the sync variable to try and sync the games of the two clients. Before working on this process, both our clients were able to successfully connect to the server and play. After our first major breakthrough with the sync variable, we thought that our syntax was correct. However, the clients were able to connect but the Window would not respond, and the game was not able to be played. After attempting to debug, we decided that it would be best to restructure the threading and the way we were updating the status. In doing so, we were able to get one player's game fully working, but the other client still had an unresponsive window. Finally, we realized ******** and made some more changes to get both clients synced up and able to play.

Lessons Learned:

Before starting this project, we had a very basic understanding of Python. We have learned a lot more about Python than we knew before and have become more familiar with its concepts and syntax. Additionally, we have a deeper understanding of the client server program. We understand the importance of implementing a way to sync the two games, so that one player does not get behind the other.

Something that we could only learn after this project was completed, was to not overcomplicate this process as a whole. Although parts of this assignment were very challenging, there were many objectives that we were trying to make more complicated when we started.

Conclusions:

This project not only challenged us, but also taught us how to establish a client-server connection. We overcame obstacles and had to adapt. For example, when we were drawing out and brainstorming, we thought the sync variable and game state would be the easiest part to implement, but in our case, it was not. By meeting two times a week for much of the semester and utilizing GitHub to share our progress, we implemented a synchronized pong game that runs over a network server.