CSS 432
Team: Yeseul An, Iris Favoreal, Dinh Luong
Professor: Brent Lagesse

Introduction

This is a client-server application that implements a turn-based multi-player Tic-Tac-Toe game. Clients or players connect to a server which coordinates the gameplay with matched pairs serviced in a room, with a total of up to two rooms. The application is text-based and is played on a command-line console.

How to Compile & Run the Game

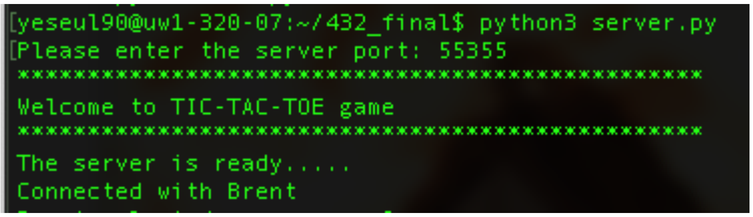
Prerequisite: Our game is built on top of python3. We need to execute the game with command python3.

There are two code files included:

client.py	code file of the client
server.py	code file of the server

To run the server code:

1. Type $python3 server.py on the terminal
2. Type port number (which should be the same as client input) ex: $55355

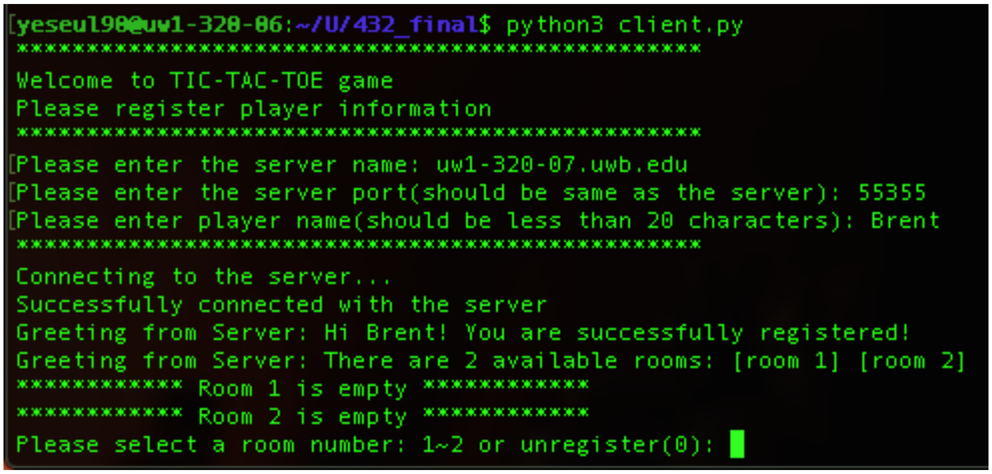


```
[yeseul90@uw1-320-07:~/432_final$ python3 server.py
[Please enter the server port: 55355
 *************************************************
 Welcome to TIC-TAC-TOE game
 *************************************************
 The server is ready.....
 Connected with Brent
```

Figure 1: Starting of the Server Program

To run the client code: type $python3 client.py on the terminal

1. Type $python3 client.py
2. Type server name that you want to connect to. ex: $uw1-320-07.uwb.edu
3. Type port number (which should be the same as server) ex: $55355
4. Type name of the player. ex: $Brent

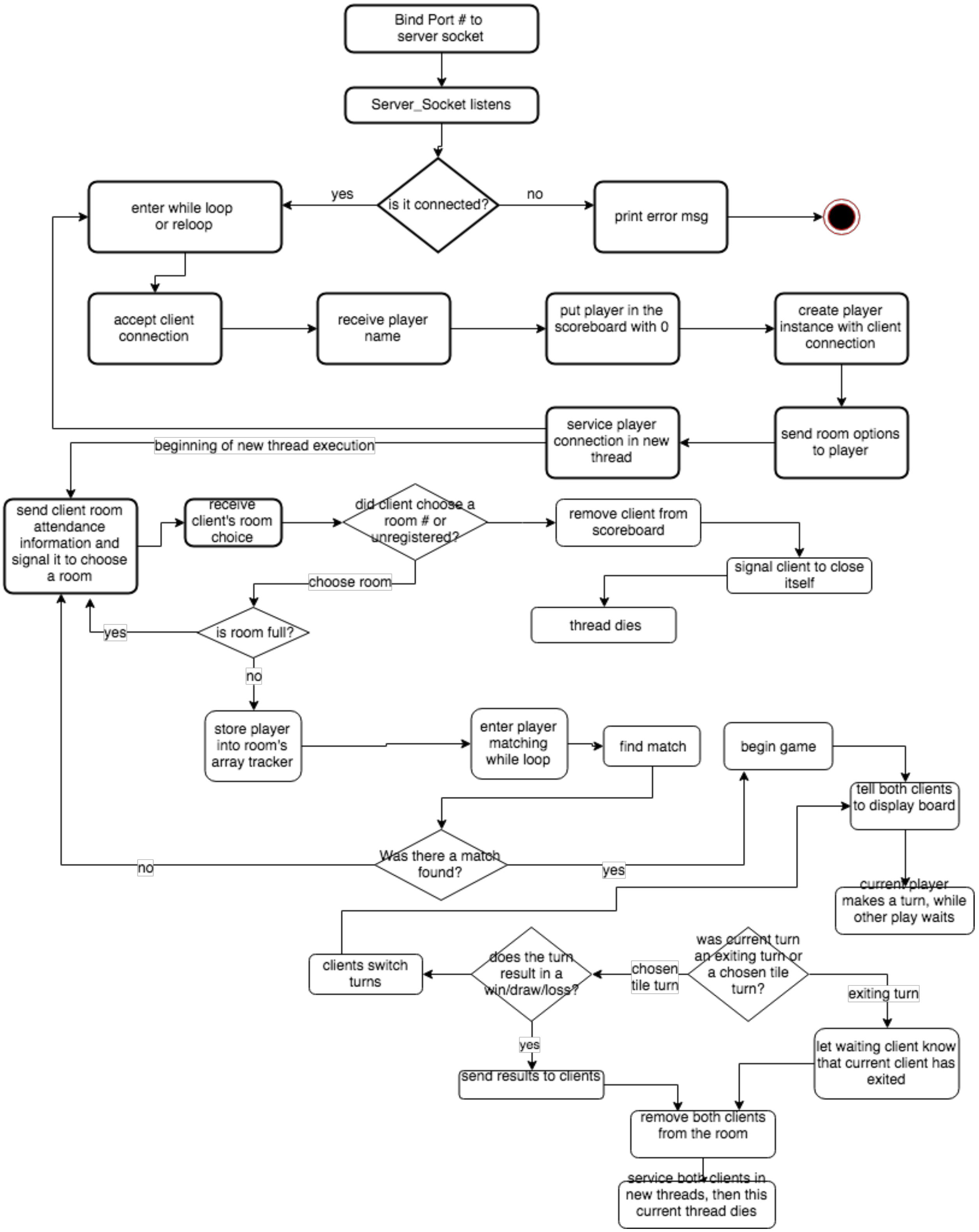


```
[yeseul90@uw1-320-06:~/U/432_final$ python3 client.py
*************************************************
Welcome to TIC-TAC-TOE game
Please register player information
*************************************************
[Please enter the server name: uw1-320-07.uwb.edu
[Please enter the server port(should be same as the server): 55355
[Please enter player name(should be less than 20 characters): Brent
*************************************************
Connecting to the server...
Successfully connected with the server
Greeting from Server: Hi Brent! You are successfully registered!
Greeting from Server: There are 2 available rooms: [room 1] [room 2]
************ Room 1 is empty ************
************ Room 2 is empty ************
Please select a room number: 1~2 or unregister(0):
```

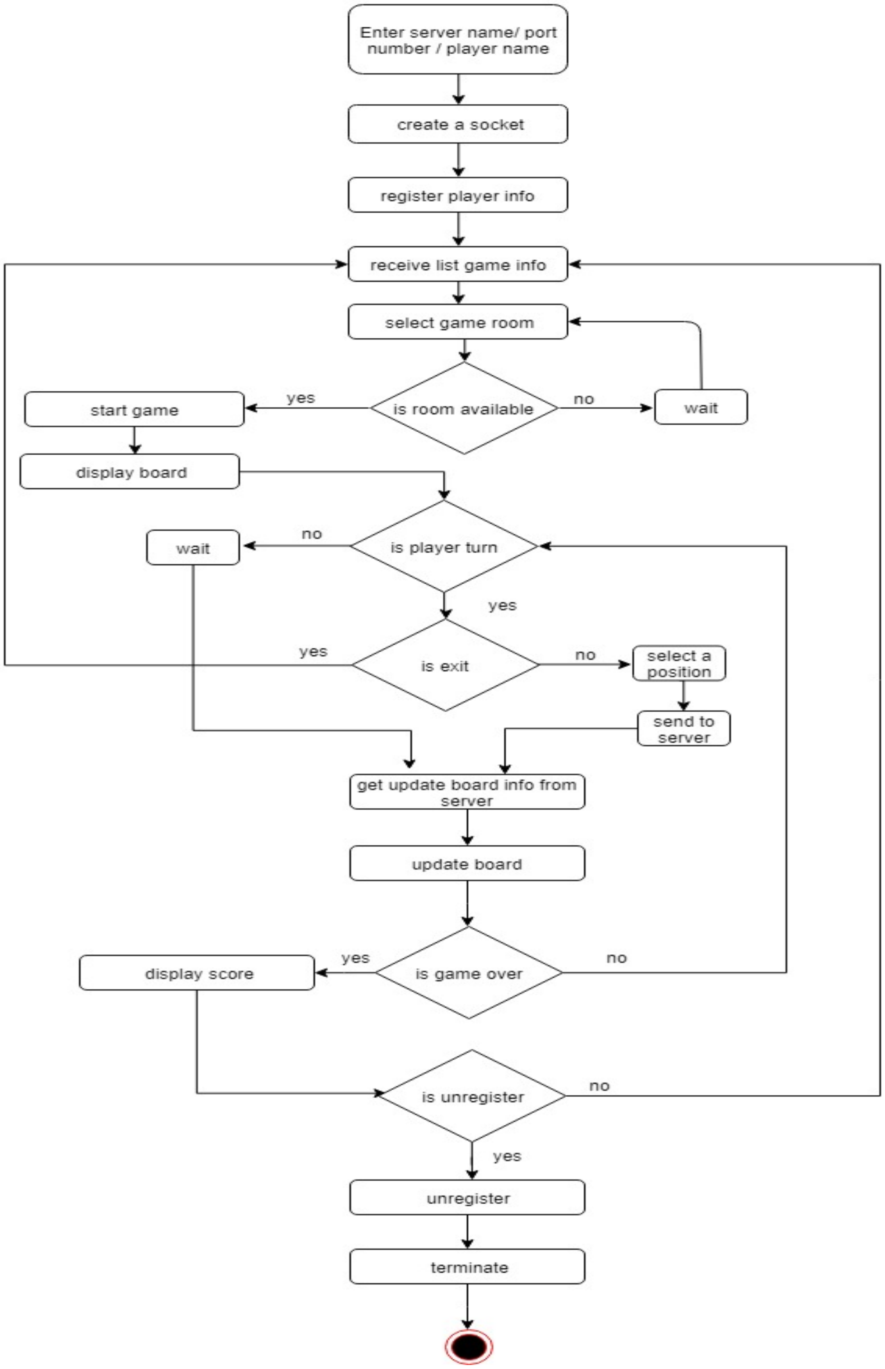
Figure 2: Starting of the Client Program

Flow Chart of the Program

Server


Bind Port # to server socket
Server_Socket listens
is it connected?
yes
no
enter while loop or reloop
print error msg
accept client connection
receive player name
put player in the scoreboard with 0
create player instance with client connection
send room options to player
service player connection in new thread
beginning of new thread execution
send client room attendance information and signal it to choose a room
receive client's room choice
did client choose a room # or unregistered?
remove client from scoreboard
signal client to close itself
thread dies
choose room
is room full?
yes
no
store player into room's array tracker
enter player matching while loop
find match
begin game
tell both clients to display board
Was there a match found?
no
yes
current player makes a turn, while other play waits
clients switch turns
does the turn result in a win/draw/loss?
chosen tile turn
was current turn an exiting turn or a chosen tile turn?
exiting turn
yes
let waiting client know that current client has exited
send results to clients
remove both clients from the room
service both clients in new threads, then this current thread dies

Client


Enter server name/ port number / player name
create a socket
register player info
receive list game info
select game room
is room available
yes
no
start game
wait
display board
is player turn
no
wait
yes
is exit
yes
no
select a position
send to server
get update board info from server
update board
is game over
yes
no
display score
is unregister
no
yes
unregister
terminate

Implementation of Protocols

Register

The server (server.py) has to be running before a client program can connect to it. When a player runs the client.py, the client has to input the running server's corresponding name and port number to connect to the tic-tac-toe game server which should be listening for connections.

The server's listening socket accepts the client's incoming connection and services it in a new socket. If the connection is successful, the player will be asked to input their name as means of registration. When the player sends the name to the server, the server calls its register function and and creates a player instance with the client's name and the client connection. This client player is then serviced in a new thread. Also, it records the player's name in the dictionary for the scoreboard. When the server successfully registers the player, it sends back the message saying that the player is successfully registered.

The register function is continuously runs in a while loop, accepting incoming client connections, registering its information into the server accordingly, and servicing each client connection in a new thread.

List Games

Our game provides two room for players to join. After the player connects to the server and registers his/her name, the server sends room attendance availability information to the client. In this way, the player can see what rooms are available to join into and who's in it so that they can select the room they want.

There are three possible situations in selecting room:

- If the room is empty, there will be no one waiting in that room. The player has to wait for another client to connect and join that room.
- If there is already a waiting player in the room, the game starts automatically, and the latter player will be the first to take its turn.
- If a player selects a room and if the room is full, the server will send message to tell the player that the room is full. the player has to wait or select the different room option.

The protocol is mostly implemented with sending and receiving messages between client and server. To have the socket deliver messages without error, we implement the following algorithm:

- The sending side measures the size of the message and sends the length to the receiver
- The receiver receives the size of the message
- The sending side sends the actual message to the receiver
- The receiver receives the message from the socket only up to the size that it received earlier

By measuring and relaying the size to the receiver before sending the message, this will ensure that the receiver reads the message up to its exact length from the sender in their communication.

Additionally, we generate a list of message types to the signal what type of message the sender is going to send out. (e.g. type “q” indicates a player has quit the game or lost connection; type “n” indicates that a message is a number; type “s” indicates the message is long sentence)

Create Game

When a player selects the room and when the room has 2 players matched, the server will automatically create the game. There is no restriction for a player to join the room unless the room is full. After two players are matched, each player will receive a message from the server about who they are matched with and the role they have been assigned to. For example, the player will receive a message, “We found you a match player: Brent | Your role in the game is O”.

Join Game

A player joins the game by typing the corresponding room number. They will receive information from the server about who is in the room and whether the room they want to join is full or not. Again, if the room the player selects is already full, the player will receive the message from the server that he/she has to wait and select the other room. If there is no players in the room, a player has to be waited in the selected room until the other player joins the room. If there is already a player waiting in the room, when the other player joins the room the server automatically creates and starts the game.

Exit Game

During the game, every time a player has a turn, the player has an option to exit the game by selecting 0. When the player chooses to exit the game, the player is sent back to the main menu where they are given the option to choose a room to join again or unregister. The waiting player is notified that their opponent has left the game, so this player is also sent back to the main menu where they can choose a room or unregister. Unlike unregister, when a player selects the option exit, this means that the client program of each player is still connected with the server, and server still has the records of the players in the scoreboard.

Unregister

In our protocol, the players are maintained in the lists: room1_pair, room2_pair, and the scoreboard. When the player either exits the game or finishes the game (the game has been won or lost or the game resulted in a draw), the player records still exists in the scoreboard and their connections still remains with the server. The players can select a new room and keep starting new games. However, when a player decides to unregister, the player will be removed from the scoreboard and the connection is closed in the client program. Unlike exit functionality, unregister basically removes the record of a player permanently from the server and the client program no longer connected with the server.

Match Players

The server calls the matching player protocol every time a new player joins the game. This method uses a lock to avoid concurrent accesses from multiple threads. The method will return none if there was no match found with the calling thread or it will return a second player to which the calling thread is matched to. Depending on the room chosen by the calling thread, it will parse through that room's array of players and check if there is another player that is not the calling thread's player. If there is, the calling thread player is matched with this player. The game then began.

Scoreboard

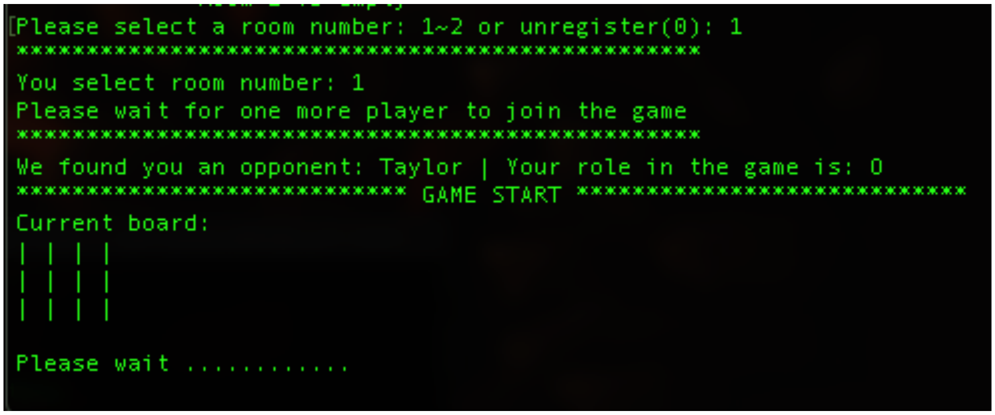
We implemented scoreboard option for each player that are not unregistered. The scores are announced to the players from the server every time they finish the game. The score is sorted in reverse so that the highest score comes first. If the scores are the same, the players are listed alphabetically.

How to Play the Tic-Tac-Toe

Following are the steps to play our game:

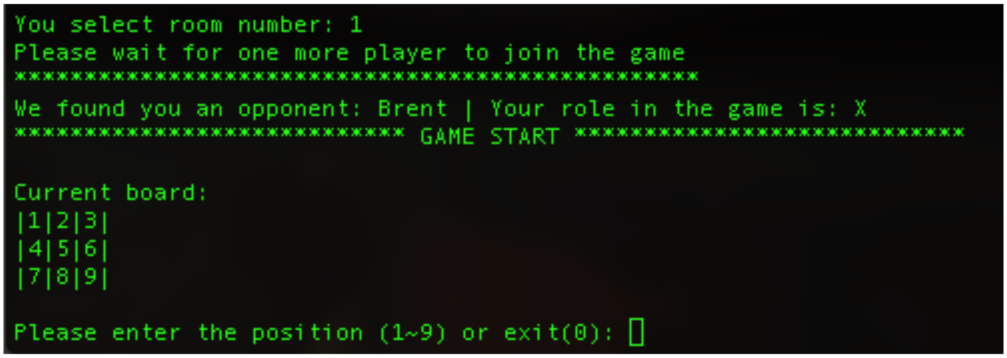
1. Run the server.py which implements the server that receives client connections as the players
2. Each player has to run the client.py, and repeat these steps to play the game
   a. Input server name and port corresponding to running server's info
   b. Input player name
   c. Select a game room
      i. If the game room is available, start the game. In the "Please enter the position" field, type in a number (from 1 to 9) to play the game until it finishes, or type in 0 to quit the current game and select a new game room to play a new game
      ii. If the game room is not available, wait and try to select a game room again after 1 minute to see if the room is available
   d. To exit the current game and restart a new game, type in 0 in the "Please enter the position" field when playing the game
   e. To unregister and terminate the game, type in 0 in the "Please select a room number" field when select a game room.

Below are the screenshots of playing the game between two players:



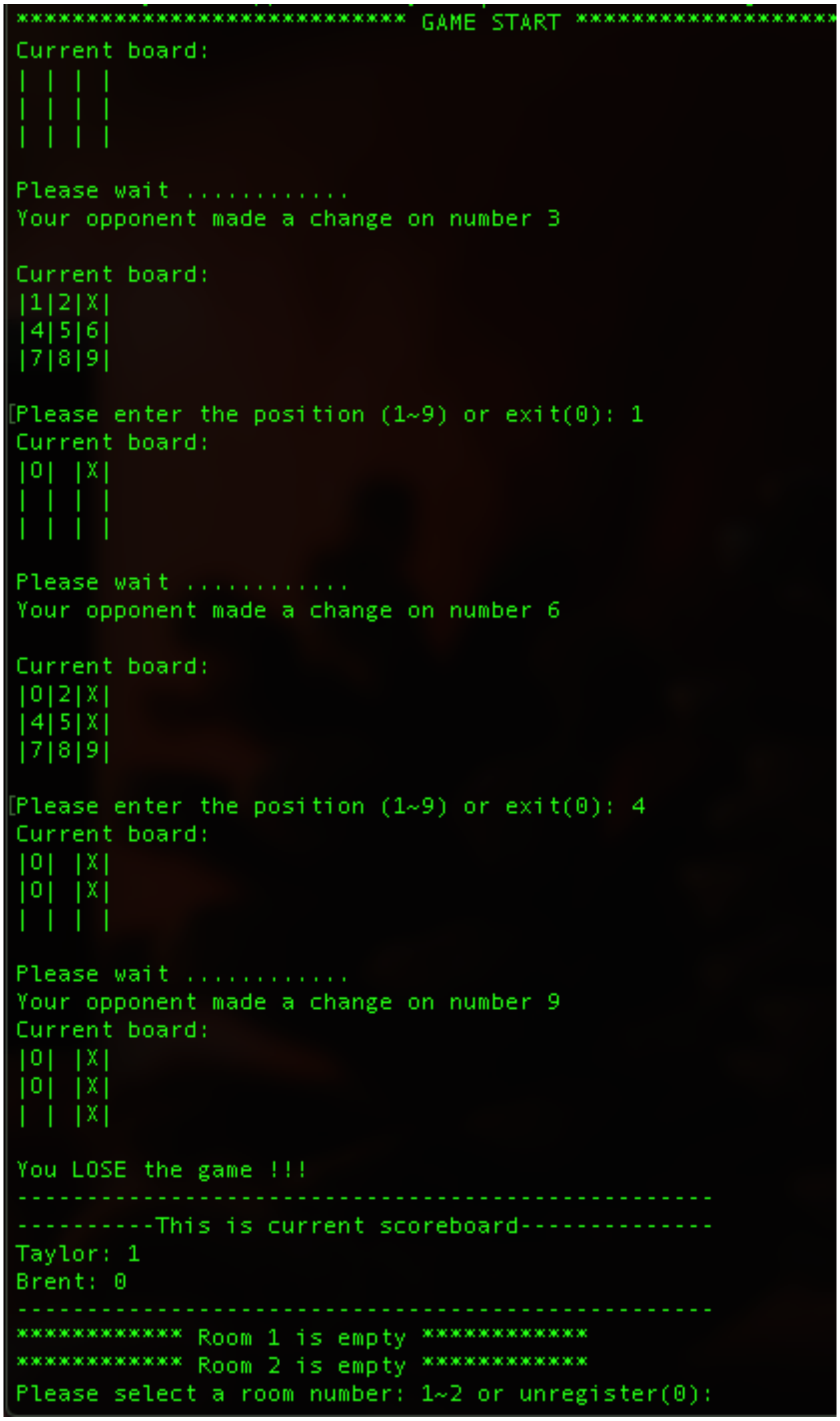
```
[Please select a room number: 1~2 or unregister(0): 1
*************************************************
You select room number: 1
Please wait for one more player to join the game
*************************************************
We found you an opponent: Taylor | Your role in the game is: 0
**************************** GAME START ****************************
Current board:
| | | |
| | | |
| | | |

Please wait ............
```

Figure 3: Game Starting in Player 1



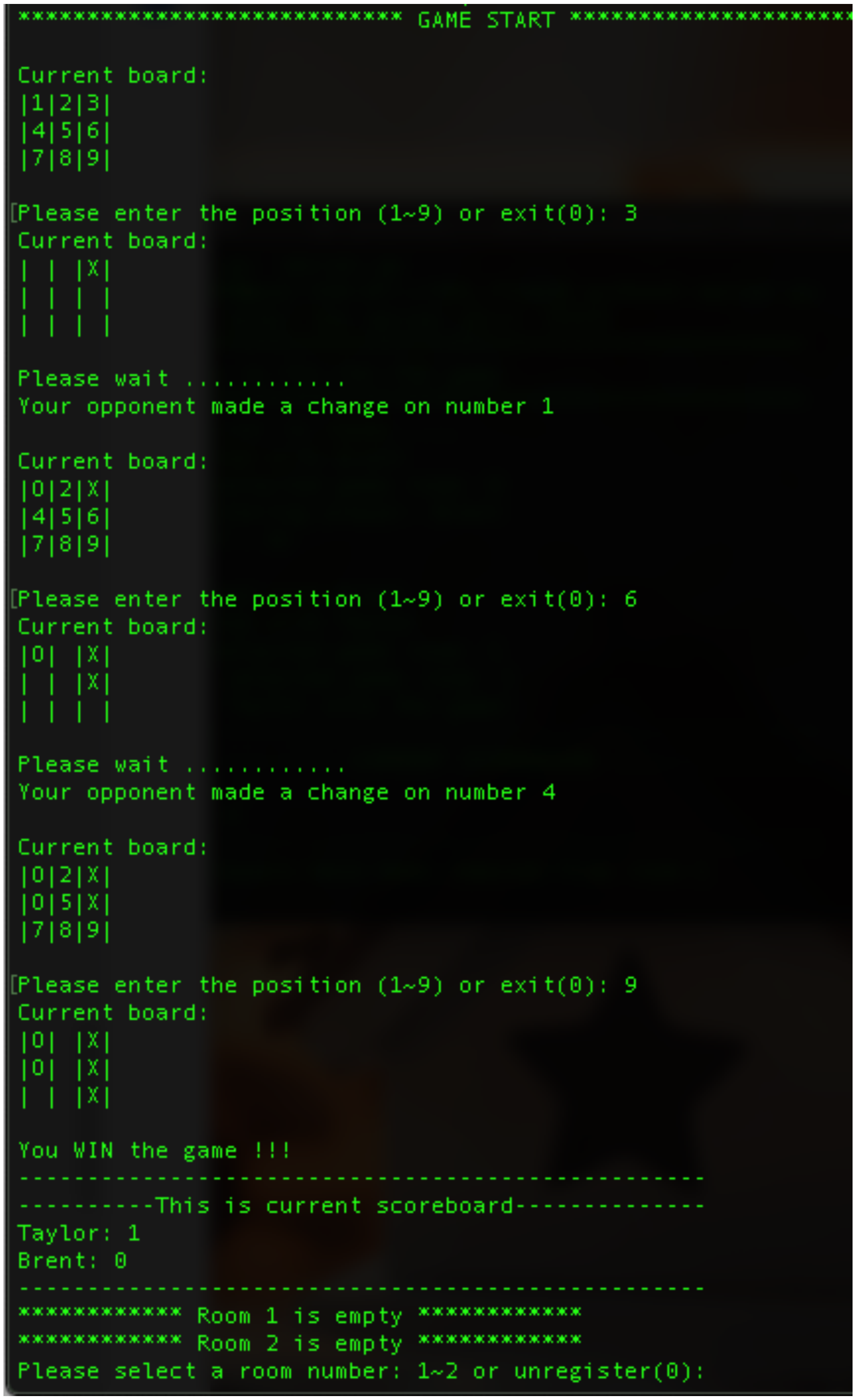
```
You select room number: 1
Please wait for one more player to join the game
*************************************************
We found you an opponent: Brent | Your role in the game is: X
**************************** GAME START ****************************

Current board:
|1|2|3|
|4|5|6|
|7|8|9|

Please enter the position (1~9) or exit(0):
```

Figure 4: Game Starting in Player 2



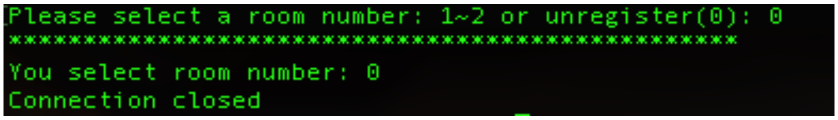
```
**************************** GAME START *******************
Current board:
| | | |
| | | |
| | | |

Please wait ............
Your opponent made a change on number 3

Current board:
|1|2|X|
|4|5|6|
|7|8|9|

[Please enter the position (1~9) or exit(0): 1
Current board:
|O| |X|
| | | |
| | | |

Please wait ............
Your opponent made a change on number 6

Current board:
|O|2|X|
|4|5|X|
|7|8|9|

[Please enter the position (1~9) or exit(0): 4
Current board:
|O| |X|
|O| |X|
| | | |

Please wait ............
Your opponent made a change on number 9
Current board:
|O| |X|
|O| |X|
| | |X|

You LOSE the game !!!
--------------------------------------------------
----------This is current scoreboard--------------
Taylor: 1
Brent: 0
--------------------------------------------------
************ Room 1 is empty ************
************ Room 2 is empty ************
Please select a room number: 1~2 or unregister(0):
```

Figure 5: Game Playing in Player 1



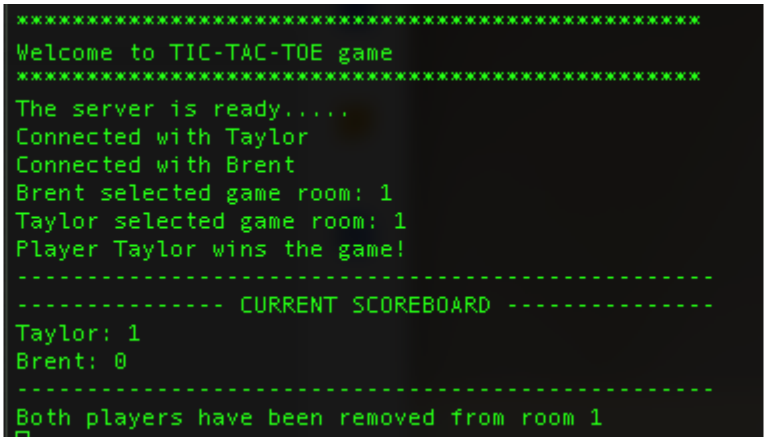
```
**************************** GAME START *********************

Current board:
|1|2|3|
|4|5|6|
|7|8|9|

[Please enter the position (1~9) or exit(0): 3
Current board:
| | |X|
| | | |
| | | |

Please wait ............
Your opponent made a change on number 1

Current board:
|O|2|X|
|4|5|6|
|7|8|9|

[Please enter the position (1~9) or exit(0): 6
Current board:
|O| |X|
| | |X|
| | | |

Please wait ............
Your opponent made a change on number 4

Current board:
|O|2|X|
|O|5|X|
|7|8|9|

[Please enter the position (1~9) or exit(0): 9
Current board:
|O| |X|
|O| |X|
| | |X|

You WIN the game !!!
--------------------------------------------------
----------This is current scoreboard--------------
Taylor: 1
Brent: 0
--------------------------------------------------
************ Room 1 is empty ************
************ Room 2 is empty ************
Please select a room number: 1~2 or unregister(0):
```

Figure 6: Game Playing in Player 2



```
Please select a room number: 1~2 or unregister(0): 0
*************************************************
You select room number: 0
Connection closed
```

Figure 7: Client Unregister the Game



```
*************************************************
Welcome to TIC-TAC-TOE game
*************************************************
The server is ready.....
Connected with Taylor
Connected with Brent
Brent selected game room: 1
Taylor selected game room: 1
Player Taylor wins the game!
--------------------------------------------------
--------------- CURRENT SCOREBOARD ---------------
Taylor: 1
Brent: 0
--------------------------------------------------
Both players have been removed from room 1
```

Figure 8: Server Side of Display

Implementation

Development Tools

Language: Python

Source control: git, bitbucket

Language IDE: Pycharm, IntelliJ

Test Cases

We have tested our game based on the test cases below. These test cases are executed successfully without any errors.

1. Play games between two players
2. Players with all cases: win, lost, draw
3. Play games between four players in different rooms
4. A player tries to enter the room that is already full
5. A player waits until current players playing a game finish up their a game, and starts a new game with one of them.
6. One player exits the game in the middle of the game -> go back to the main menu
7. A player unregisters from the game

How We Distribute the Work

We have distributed the work based on our capabilities.

Iris Favoreal: Handled the client multi-thread operations such as implementing the exit functionality so that whenever a player exits the game, it will go back to the main loop. Also, she guided the group members on how to lock certain functions from concurrent execution.

Yeseul An: Implemented TCP sockets for the client and server programs of the game. Implemented winning and losing logic of the game and the scoreboard using dictionary so that the players can see their scores after they finish the game. Tested the game with the possible cases and find/fix the bugs.

Dinh Luong: Implemented functions that allow different types of messages sent and received between client and server programs through sockets correctly. Implemented formatting and displaying board functionalities in the game so that the players can see their updated board while playing the game. Merged the codes among the group members so that each member can work on the updated merged code for each development period

Common problem: the most common problems that we faced while doing this network programming was sending and receiving messages correctly between the clients and the server. Whenever we added new codes in the program, we needed to discuss with each other and coordinate well so that sending and receiving messages are aligned together. Any lapse in the alignment of sending and receiving messages caused errors in our programs.