PES UNIVERSITY

EC Campus, Bengaluru

Department of Computer Science & Engineering

COMPUTER NETWORKS - UE21CS252B

4th Semester – E Section

ASSIGNMENT – 1

Title of the Project : Tic Tac Toe Using Socket Programming

Submitted to:

Dr. Geetha D

Associate Professor

Submitted By:

Melvin Jojee Joseph- PES2UG21CS294

L Sai Tejas- PES2UG21CS250

Table of Contents

1. Abstract and Scope of the Project

Abstract:

The Tic Tac Toe game project implemented in Python using Socket Programming is a game that allows two players to connect to each other using sockets and play a game of Tic Tac Toe with each other over the network. One player behaves as the server and the other player behaves as the client. The project involves the use of Python and relevant libraries to develop a functional and efficient game of Tic Tac Toe that can be played right on the terminal.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at **an IP**, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

Scope:

The scope of this project is to develop a Tic Tac Toe game that can be played over a network between two players. The game should be implemented using Python and Socket Programming and should be easy to use for the players. The project should be developed using Python and relevant libraries, and should be able to run on any operating system that supports Python and Socket Programming.

2. Project Description

The Tic Tac Toe game project implemented in Python using Socket Programming is a game that allows two players to play a game of Tic Tac Toe over the network. The game involves one player to be the server and the other player to be client. Both the client and server should be connected to the same network.
We used a server/client network architecture for this project using TCP (Transmission Control Protocol), and we used port 12625. The messages were sent back and forth between the server and the client thanks to the pickle module, which converts any object (in this case, a string list storing the nine squares of a tic-tac-toe grid) into a byte stream to be sent and back into an object to be read.
The server starts the game by first running ***server.py***, waiting for the client to connect by then running ***client.py***. Once they're connected, the game starts by itself.
The server starts as **"X"** and goes first, and the client is **"O."** The players choose the square they would like to use with coordinates; both **"A1" and "1A"** would be accepted, for example. The game proceeds, with the players taking turns until one wins or the game is a draw. Their score is then updated on the scoreboard. The host, then the client, is asked whether they'd like a rematch. If both agree, a new game is started.

3. Software Requirements

(Description about Programming Languages, API's, Methods, etc.,)

1. Python 3.x: The project requires the installation of Python 3.x or above to be installed on the system.
2. Inbuilt python libraries socket and pickle.
3. IDE: An Integrated Development Environment (IDE) is recommended to make the development process more efficient. Popular choices include Visual Studio Code, PyCharm, and Sublime Text.
4. Operating System: The project should be able to run on any operating system that supports Python and Socket Programming.
5. Network Connection: A network connection is required for playing the game over the network. This can be a local network or the internet

4. Source Code

tic_tac_toe.py

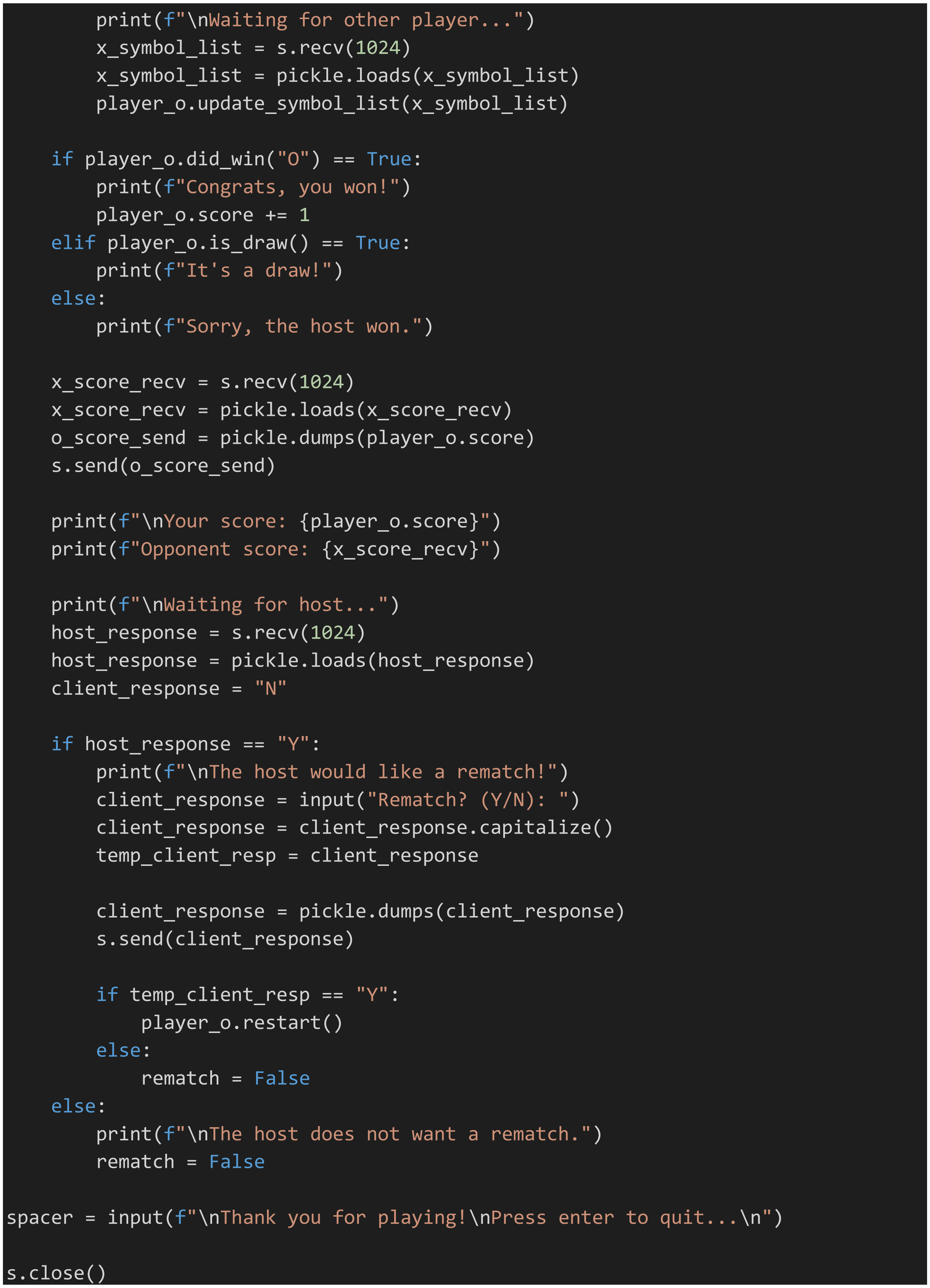
```python
class TicTacToe():

    def __init__(self, player_symbol):
        self.symbol_list = []
        self.score = 0
        for i in range(9):
            self.symbol_list.append(" ")
        self.player_symbol = player_symbol

    def restart(self):
        for i in range(9):
            self.symbol_list[i] = " "


    def draw_grid(self):
        print("\n       A   B   C\n")
        row_one = "   1   " + self.symbol_list[0]
        row_one += " | " + self.symbol_list[1]
        row_one += " | " + self.symbol_list[2]
        print(row_one)

        print("      ---+---+---")

        row_two = "   2   " + self.symbol_list[3]
        row_two += " | " + self.symbol_list[4]
        row_two += " | " + self.symbol_list[5]
        print(row_two)

        print("      ---+---+---")

        row_three = "   3   " + self.symbol_list[6]
        row_three += " | " + self.symbol_list[7]
        row_three += " | " + self.symbol_list[8]
        print(row_three, "\n")


    def edit_square(self, grid_coord):
        if grid_coord[0].isdigit():
            grid_coord = grid_coord[1] + grid_coord[0]
        col = grid_coord[0].capitalize()
        row = grid_coord[1]
        grid_index = 0
        if row == "1":
            if col == "A":
                grid_index = 0
            elif col == "B":
                grid_index = 1
            elif col == "C":
                grid_index = 2
        elif row == "2":
            if col == "A":
                grid_index = 3
            elif col == "B":
                grid_index = 4
            elif col == "C":
                grid_index = 5
        elif row == "3":
            if col == "A":
                grid_index = 6
            elif col == "B":
                grid_index = 7
            elif col == "C":
                grid_index = 8

        if self.symbol_list[grid_index] == " ":
            self.symbol_list[grid_index] = self.player_symbol

    def update_symbol_list(self, new_symbol_list):
        for i in range(9):
            self.symbol_list[i] = new_symbol_list[i]

    def did_win(self, player_symbol):
        g = []
        for i in range(9):
            g.append(self.symbol_list[i])
        sym = player_symbol

        if g[0] == sym and g[1] == sym and g[2] == sym:
            return True

        elif g[3] == sym and g[4] == sym and g[5] == sym:
            return True

        elif g[6] == sym and g[7] == sym and g[8] == sym:
            return True

        elif g[0] == sym and g[3] == sym and g[6] == sym:
            return True
        elif g[1] == sym and g[4] == sym and g[7] == sym:
            return True

        elif g[2] == sym and g[5] == sym and g[8] == sym:
            return True

        elif g[2] == sym and g[4] == sym and g[6] == sym:
            return True

        elif g[0] == sym and g[4] == sym and g[8] == sym:
            return True

        return False

    def is_draw(self):
        num_blanks = 0
        for i in range(9):
                if self.symbol_list[i] == " ":
                    num_blanks += 1
        if self.did_win(self.player_symbol) == False and num_blanks == 0:
            return True
        else:
            return False
```

server.py

```
import socket
import pickle

from tic_tac_toe import TicTacToe

HOST = '127.0.0.1'
PORT = 12625

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(5)

client_socket, client_address = s.accept()
print(f"\nConnnected to {client_address}!")

player_x = TicTacToe("X")

rematch = True

while rematch == True:
    print(f"\n\n TIC-TAC-TOE")
    while player_x.did_win("X") == False and player_x.did_win("O") == False and
player_x.is_draw() == False:
        print(f"\n       Your turn!")
        player_x.draw_grid()
        player_coord = input(f"Enter coordinate: ")
        player_x.edit_square(player_coord)
        player_x.draw_grid()

        x_symbol_list = pickle.dumps(player_x.symbol_list)
        client_socket.send(x_symbol_list)

        if player_x.did_win("X") == True or player_x.is_draw() == True:
            break

        print(f"\nWaiting for other player...")
        o_symbol_list = client_socket.recv(1024)
        o_symbol_list = pickle.loads(o_symbol_list)
        player_x.update_symbol_list(o_symbol_list)

    if player_x.did_win("X") == True:
        print(f"Congrats, you won!")
        player_x.score+=1
    elif player_x.is_draw() == True:
        print(f"It's a draw!")
    else:
        print(f"Sorry, the client won.")

    x_score_send = pickle.dumps(player_x.score)
    client_socket.send(x_score_send)
    o_score_recv = client_socket.recv(1024)
    o_score_recv = pickle.loads(o_score_recv)

    print(f"\nYour score: {player_x.score}")
    print(f"Opponent score: {o_score_recv}")

    host_response = input(f"\nRematch? (Y/N): ")
    host_response = host_response.capitalize()
    temp_host_resp = host_response
    client_response = ""

    host_response = pickle.dumps(host_response)
    client_socket.send(host_response)

    if temp_host_resp == "N":
        rematch = False
    else:
        print(f"Waiting for client response...")
        client_response = client_socket.recv(1024)
        client_response = pickle.loads(client_response)
        if client_response == "N":
            print(f"\nThe client does not want a rematch.")
            rematch = False
        else:
            player_x.restart()

spacer = input(f"\nThank you for playing!\nPress enter to quit...\n")

client_socket.close()
```

client.py

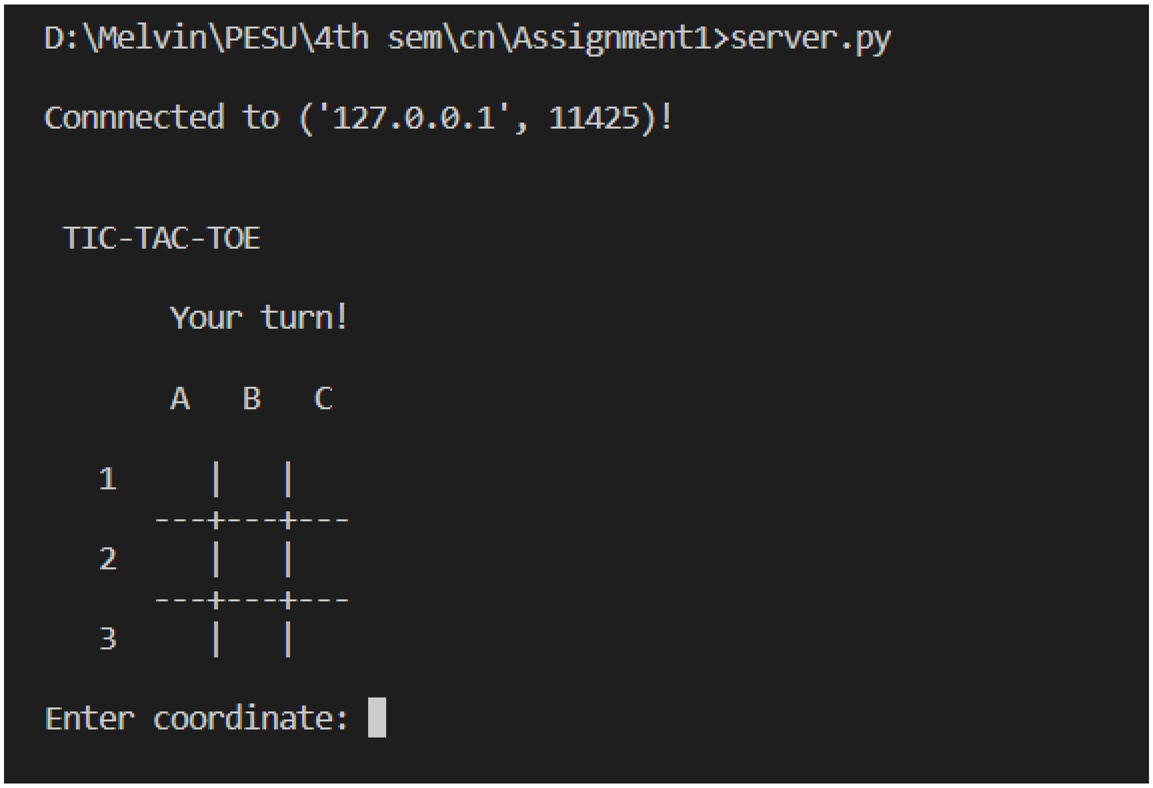
```
import socket
import pickle

from tic_tac_toe import TicTacToe

HOST = '127.0.0.1'
PORT = 12625

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
print(f"\nConnected to {s.getsockname()}!")

player_o = TicTacToe("O")

rematch = True
while rematch == True:
    print(f"\n\n TIC-TAC-TOE")
    player_o.draw_grid()
    print(f"\nWaiting for other player...")
    x_symbol_list = s.recv(1024)
    x_symbol_list = pickle.loads(x_symbol_list)
    player_o.update_symbol_list(x_symbol_list)
    while player_o.did_win("O") == False and player_o.did_win("X") == False and
player_o.is_draw() == False:
        print(f"\n       Your turn!")
        player_o.draw_grid()
        player_coord = input(f"Enter coordinate: ")
        player_o.edit_square(player_coord)
        player_o.draw_grid()
        o_symbol_list = pickle.dumps(player_o.symbol_list)
        s.send(o_symbol_list)

        if player_o.did_win("O") == True or player_o.is_draw() == True:
            break
```



```
        print(f"\nWaiting for other player...")
        x_symbol_list = s.recv(1024)
        x_symbol_list = pickle.loads(x_symbol_list)
        player_o.update_symbol_list(x_symbol_list)

    if player_o.did_win("O") == True:
        print(f"Congrats, you won!")
        player_o.score += 1
    elif player_o.is_draw() == True:
        print(f"It's a draw!")
    else:
        print(f"Sorry, the host won.")

    x_score_recv = s.recv(1024)
    x_score_recv = pickle.loads(x_score_recv)
    o_score_send = pickle.dumps(player_o.score)
    s.send(o_score_send)

    print(f"\nYour score: {player_o.score}")
    print(f"Opponent score: {x_score_recv}")

    print(f"\nWaiting for host...")
    host_response = s.recv(1024)
    host_response = pickle.loads(host_response)
    client_response = "N"

    if host_response == "Y":
        print(f"\nThe host would like a rematch!")
        client_response = input("Rematch? (Y/N): ")
        client_response = client_response.capitalize()
        temp_client_resp = client_response

        client_response = pickle.dumps(client_response)
        s.send(client_response)

        if temp_client_resp == "Y":
            player_o.restart()
        else:
            rematch = False
    else:
        print(f"\nThe host does not want a rematch.")
        rematch = False

spacer = input(f"\nThank you for playing!\nPress enter to quit...\n")

s.close()
```

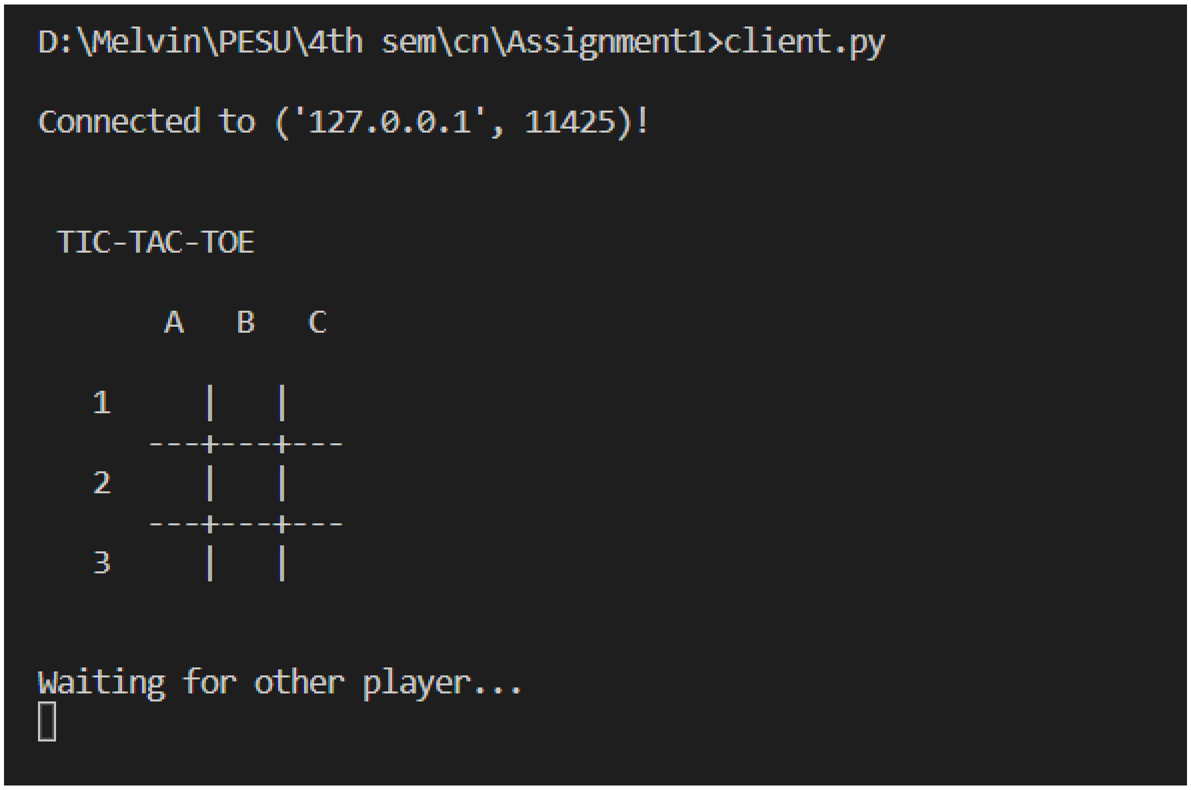
5. Sample Output

(Include necessary output screenshots followed by brief description)
The server.py program and client.py program is started



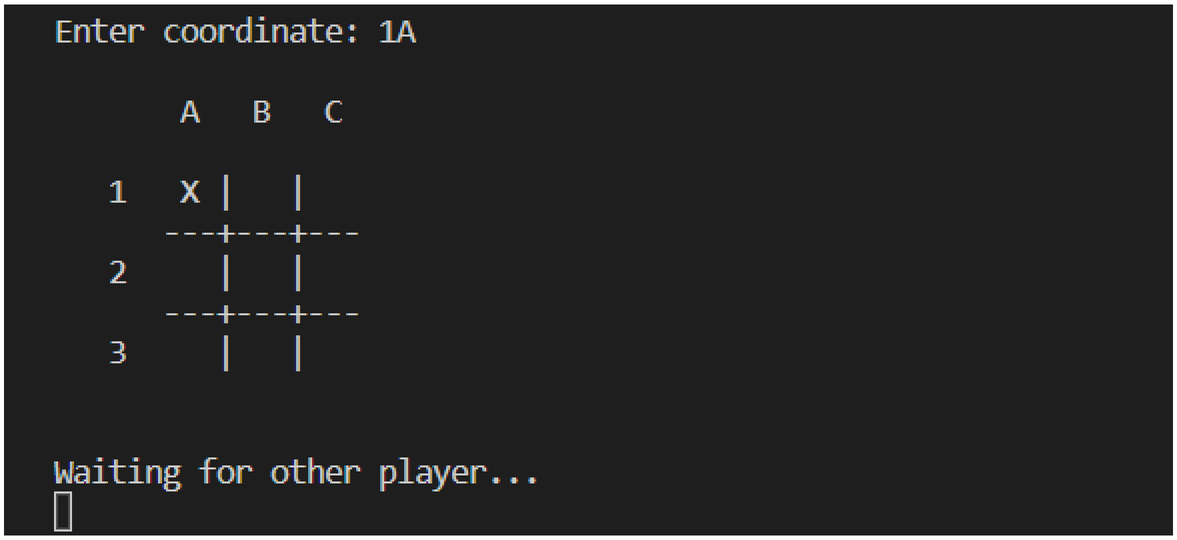
```
D:\Melvin\PESU\4th sem\cn\Assignment1>server.py

Connected to ('127.0.0.1', 11425)!


 TIC-TAC-TOE

       Your turn!

       A   B   C

   1     |   |
      ---+---+---
   2     |   |
      ---+---+---
   3     |   |

Enter coordinate: 
```

The host is being prompted to enter coordinates.



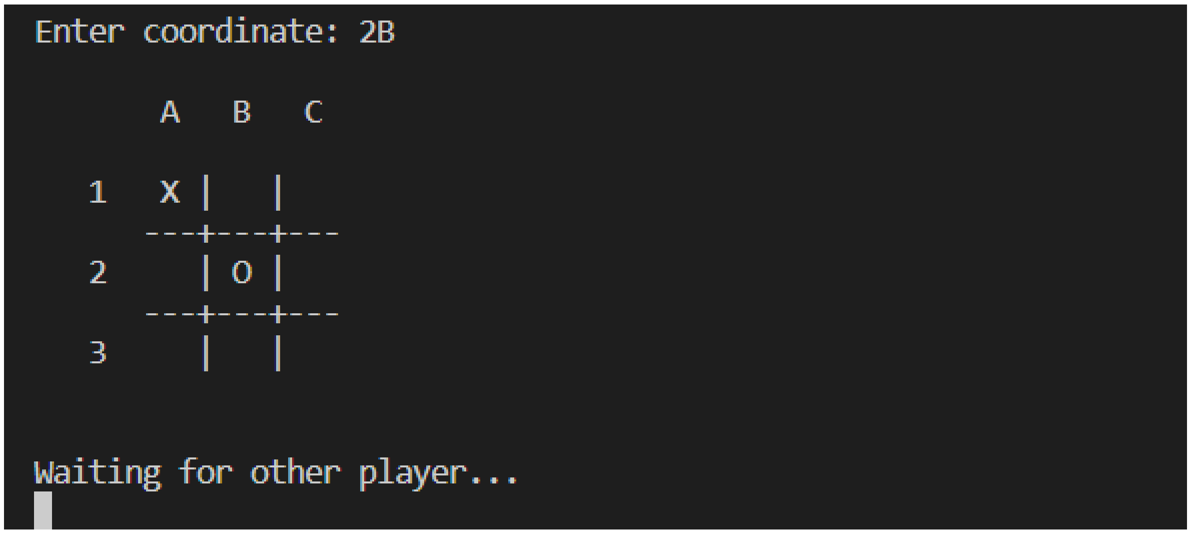
```
D:\Melvin\PESU\4th sem\cn\Assignment1>client.py

Connected to ('127.0.0.1', 11425)!


 TIC-TAC-TOE

      A   B   C

   1     |   |
      ---+---+---
   2     |   |
      ---+---+---
   3     |   |


Waiting for other player...
```

The client is being asked to wait for the server to play.



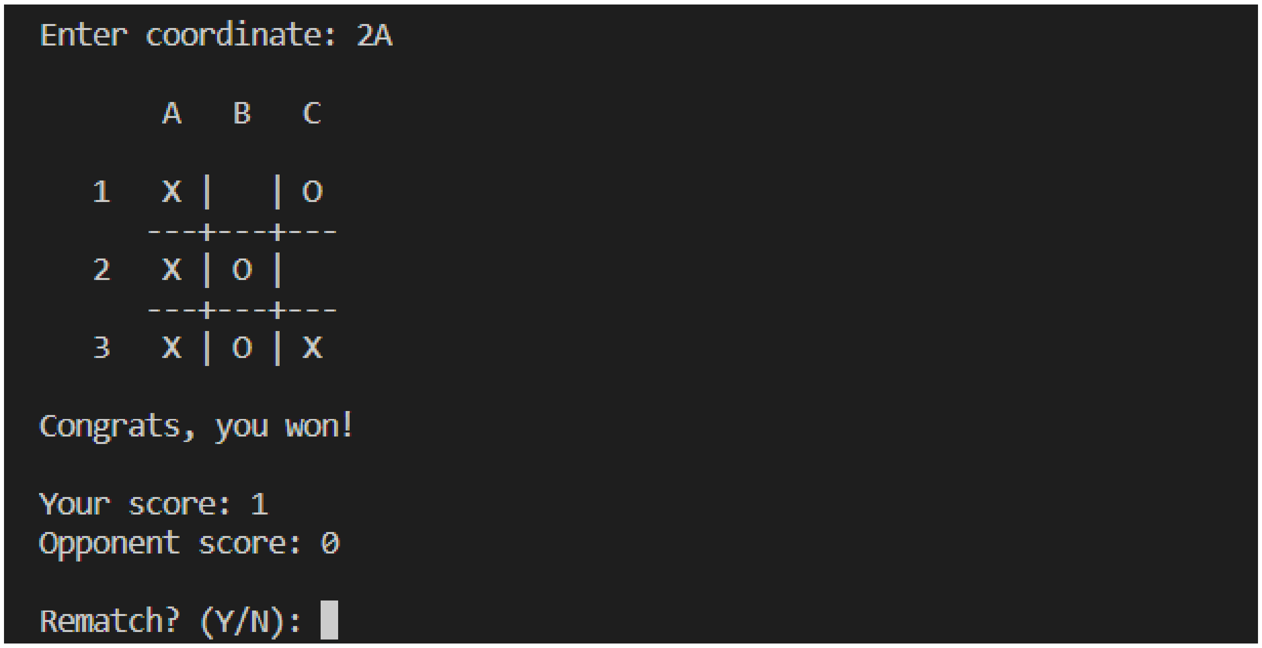
```
 Enter coordinate: 1A

      A   B   C

   1  X  |   |
      ---+---+---
   2     |   |
      ---+---+---
   3     |   |


 Waiting for other player...
```

The server has entered the coordinate 1A and the board has been updated and the host is being asked to wait.



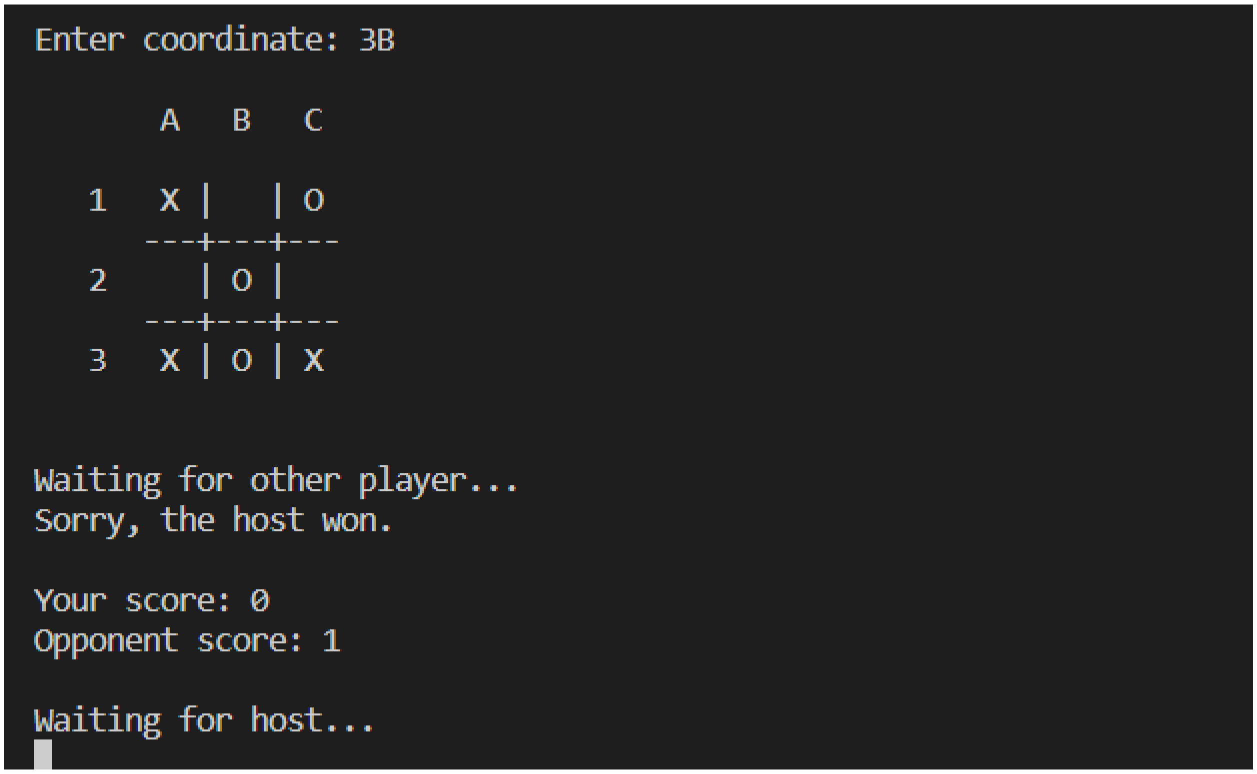
```
Enter coordinate: 2B

      A   B   C

  1   X |   |
     ---+---+---
  2     | O |
     ---+---+---
  3     |   |


Waiting for other player...
```

The client has entered coordinate 2B and the board has been updated. The client is asked to wait for the other player.
The game goes on and in the end one of the players is declared the winner.



```
Enter coordinate: 2A

      A   B   C

  1   X |   | O
     ---+---+---
  2   X | O |
     ---+---+---
  3   X | O | X

Congrats, you won!

Your score: 1
Opponent score: 0

Rematch? (Y/N):
```

Here the server has won the game and the score board has been updated.
The host is also being asked if he would like a rematch.



```
Enter coordinate: 3B

      A   B   C

   1   X |   | O
      ---+---+---
   2     | O |
      ---+---+---
   3   X | O | X


Waiting for other player...
Sorry, the host won.

Your score: 0
Opponent score: 1

Waiting for host...
```

The client has lost and it is also reflected on the scoreboard.

6. Conclusion

In conclusion, the Tic Tac Toe game project implemented in Python using Socket Programming is a functional and efficient game that allows two players to play a game of Tic Tac Toe over the network. The game was tested extensively to ensure that it was functional and efficient and most importantly easy to use for the both players.

7. References

https://www.geeksforgeeks.org/socket-programming-python/

https://www.digitalocean.com/community/tutorials/python-socket-programming-server-client

https://github.com/Suvoo/TicTacToe-Using-Socket-Server