

## **CS 421 – Computer Networks**

### **Programming Assignment 2**

#### **Implementing a Tic-Tac-Toe Game over TCP**

##### **Introduction**

This report details the implementation of a tic-tac-toe game using TCP communication from CS 421 – Computer Network Programming Exercise II. The goal is to develop server client applications in Java or Python. The game logic takes over and the client side interacts with the server to make moves and get game information. The server side acts as the game master, accepting connections, assigning symbols and her ID numbers, and managing game state using threads for concurrent connections. The client side connects to the server, receives game information, and acts based on the server's instructions. Threading is used for non-blocking communication and the implementation follows the TCP protocol using the Java Sockets API or the Python Sockets module. The report also includes submission instructions and optional bonus tasks for adding additional features. Through this task, students gain hands-on experience in implementing network programming, TCP communications, threading, and game logic.

##### **Code Implementation**

###### **Server:**

The code provided is a server-side implementation of the Tic-tac-toe game. It uses TCP socket communication and multithreading to allow two players to connect and play games at the same time. The server listens on the specified port and when two players connect, it assigns 'X' to the first player and 'O' to the second player. The game state is represented by a list called Playground, which initially contains spaces. The server sends instructions and game information to the client, waits for the client's move, validates that move, updates game status, checks win or tie conditions, and sends appropriate messages to the client. In case of a win or tie, the server ends the game and closes the connection.

To elaborate, the code starts by importing the required modules.

It uses sockets for socket operations, threads for multithreading support, and argparse for parsing command line arguments. Define a function called cli\_info to handle communication with each client. This function sends the initial instructions and game information to the client based on the assigned symbol ('X' or 'O') and waits for the client's hand. It validates the hand, updates the playground list, uses the how2win function to see if there is a win or tie condition, and sends the appropriate message to the client. The BoardLook function formats the game board for display. In its main function, the server sets up a socket, binds it to a host and port, and listens for incoming connections. When two players connect, a separate thread is created for each player and her cli\_info function is called with their respective client connections and player ids. A thread is started, connected, and finally the connection and socket are closed. Overall, this code allows two players to play tic-tac-toe against each other through his one server, ensuring proper gameplay and allowing the game to be played when the win or tie conditions are met.

###### **Client:**

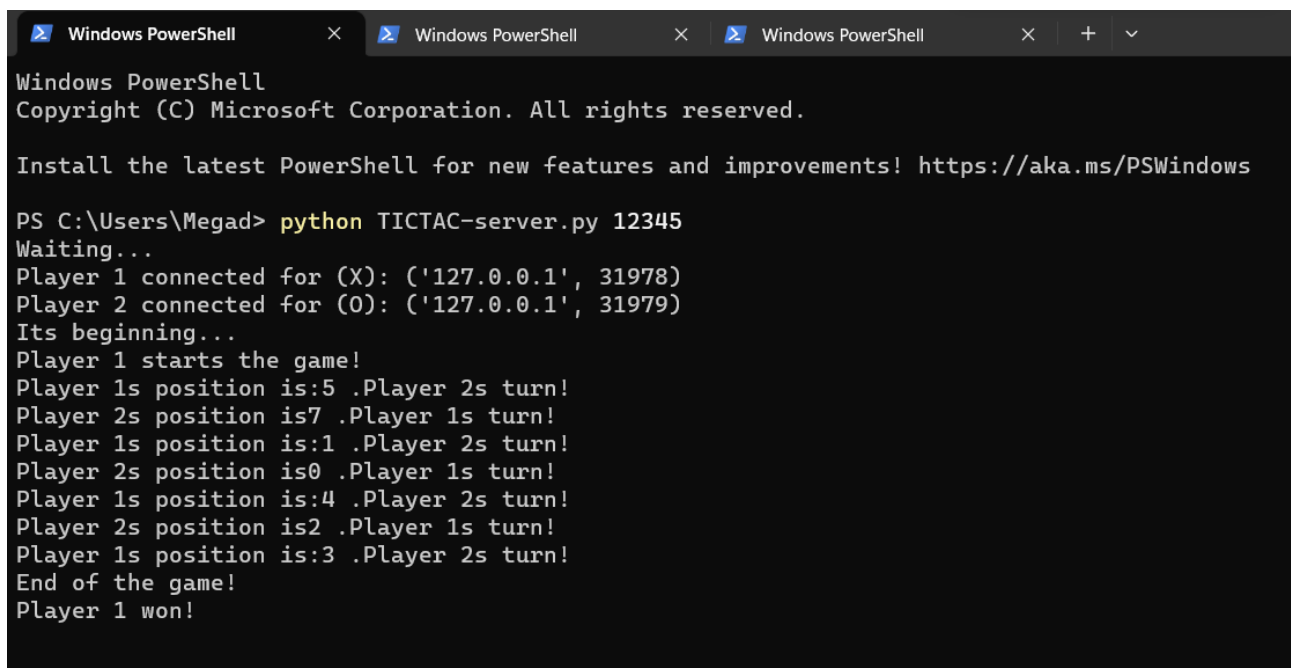
The code provided is a client-side implementation of the proxy server. Connects to the server using the specified port number passed as a command line argument. The client communicates with the server via TCP sockets, allowing bi-directional data transmission. Use multithreading to achieve simultaneous data transmission and reception. This code ensures graceful termination of the thread and closes the client socket when communication is complete.

To elaborate, the code starts by importing the required modules.

It uses sockets for socket operations, threads for multithreading support, and argparse for parsing command line arguments. Defines an argument parser for retrieving port numbers from the command line. After connecting to the server using the client socket, the code enters a loop in the submitData() function, which sends user input to the server unless the exit flag is set. In parallel, the getData() function continuously receives and outputs data from the server. Certain messages trigger the setting of flags that control program flow. The main() function creates two threads, one for receiving data and one for sending data. These threads are started, connected, and then closed. Overall, this code facilitates efficient communication between client and server by using separate threads for sending and receiving data, and ensures graceful program termination.

## Results

**Note:** Since the name of the assignment is different when taking these screenshots, it does not appear in the name you specified in the report. When running the final version of my code, it will run when you type the name you have specified.

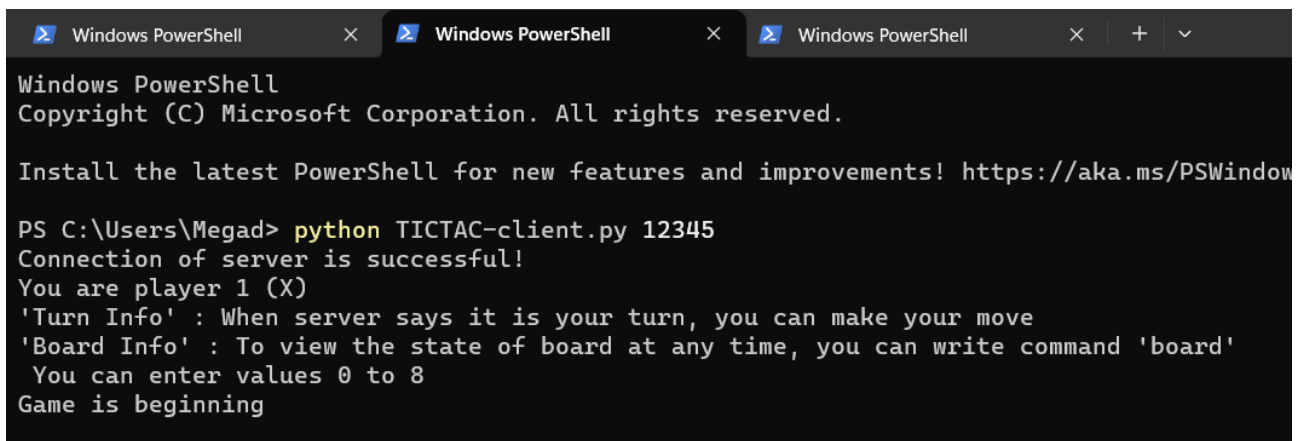


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Megad> python TICTAC-server.py 12345
Waiting...
Player 1 connected for (X): ('127.0.0.1', 31978)
Player 2 connected for (O): ('127.0.0.1', 31979)
Its beginning...
Player 1 starts the game!
Player 1s position is:5 .Player 2s turn!
Player 2s position is:7 .Player 1s turn!
Player 1s position is:1 .Player 2s turn!
Player 2s position is:0 .Player 1s turn!
Player 1s position is:4 .Player 2s turn!
Player 2s position is:2 .Player 1s turn!
Player 1s position is:3 .Player 2s turn!
End of the game!
Player 1 won!
```

**Image 1:** Server side of game 1

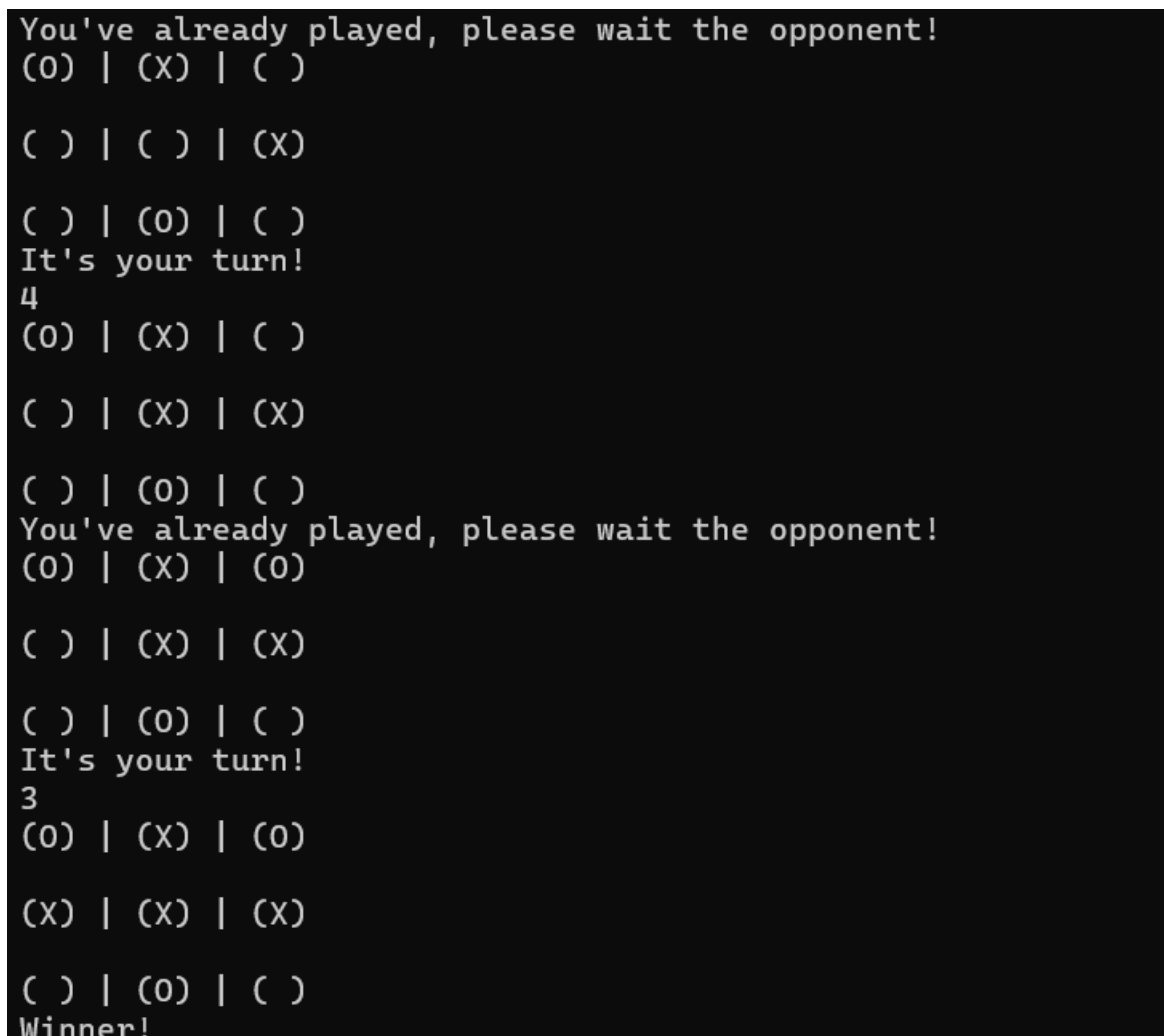


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Megad> python TICTAC-client.py 12345
Connection of server is successful!
You are player 1 (X)
'Turn Info' : When server says it is your turn, you can make your move
'Board Info' : To view the state of board at any time, you can write command 'board'
You can enter values 0 to 8
Game is beginning
```

**Image 2:** Player 1 side for game 1, introduction



```
You've already played, please wait the opponent!
(O) | (X) | ( )

( ) | ( ) | (X)

( ) | (O) | ( )
It's your turn!
4
(O) | (X) | ( )

( ) | (X) | (X)

( ) | (O) | ( )
You've already played, please wait the opponent!
(O) | (X) | (O)

( ) | (X) | (X)

( ) | (O) | ( )
It's your turn!
3
(O) | (X) | (O)

(X) | (X) | (X)

( ) | (O) | ( )
Winner!
```

**Image 3:** Player 1 won game 1

```
It's your turn!
1
Illeagal move. Please do legal move.
0
(O) | (X) | ( )
( ) | ( ) | (X)
( ) | (O) | ( )
You've already played, please wait the opponent!
(O) | (X) | ( )
( ) | (X) | (X)
( ) | (O) | ( )
It's your turn!
2
(O) | (X) | (O)
( ) | (X) | (X)
( ) | (O) | ( )
You've already played, please wait the opponent!
(O) | (X) | (O)
(X) | (X) | (X)
( ) | (O) | ( )
Lose!
```

**Image 4:** Player 2 lost game 2, it makes illegal move

```
You shall wait your opponent due to rule of turns!
(X) | ( ) | ( )

(O) | ( ) | ( )

( ) | ( ) | ( )
It's your turn!
1
(X) | (X) | ( )

(O) | ( ) | ( )

( ) | ( ) | ( )
You've already played, please wait the opponent!
1
You shall wait your opponent due to rule of turns!
(X) | (X) | ( )

(O) | ( ) | (O)

( ) | ( ) | ( )
It's your turn!
2
(X) | (X) | (X)

(O) | ( ) | (O)

( ) | ( ) | ( )
Winner!
```

**Image 5:** Player 1 won game 2, he/she always tried to make move even when it's not her/his turn

```
( ) | ( ) | ( )
( ) | ( ) | ( )
( ) | ( ) | ( )

It is your time to turn!
5
( ) | ( ) | ( )
( ) | ( ) | (X)
( ) | ( ) | ( )
You've already played, please wait the opponent!
turn
It's not your turn!
board
( ) | ( ) | ( )
( ) | ( ) | (X)
( ) | ( ) | ( )
```

**Image 6:** Player 1 writes turn and board command

```
( ) | ( ) | ( )  
( ) | ( ) | ( )  
( ) | ( ) | ( )  
  
Please wait your opponent!  
board  
( ) | ( ) | ( )  
  
( ) | ( ) | ( )  
  
( ) | ( ) | ( )  
turn  
It's not your turn!  
( ) | ( ) | ( )  
  
( ) | ( ) | (X)  
  
( ) | ( ) | ( )  
It's your turn!
```

**Image 7:** Player 2 writes turn and board command

## Conclusion

In summary, the provided code provides a server-side tic-tac-toe game implementation using TCP socket communication and multi-threading. It effectively manages interaction between clients, assigning icons, managing game state, and communicating instructions and updates. This code demonstrates the use of threads for concurrent communication and includes input validation and error handling. It creates a compelling user experience by providing clear instructions, displaying the game board, and accurately judging game outcomes. This code serves as a solid foundation for building a server-based tic-tac-toe game, demonstrating effective use of socket programming and multithreading techniques. Other improvements such as player authentication, game statistics and scalability can be added to extend the gaming experience and accommodate more players.