External Project Report on
Computer Networking (CSE3034)

Tic-Tac-Toe Game Using Client-Server Architecture

Submitted By :

Ishita Jena	Regd. No. : 2141010031
Gayatri Sahu	Regd. No. : 2141019146
Soumya Shree Dash	Regd. No. : 2141004097
Pratik Swarup Mishra	Regd. No. : 2141013122
N Nirupsha	Regd. No. : 2141014127

B.Tech. CSE 5th Semester(Section M)

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)
SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA

Declaration

We, the undersigned students of B. Tech. of **CSE** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled "**Tic-Tac-Toe Using Client-Server Architecture**" submitted to **Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking (CSE 3034)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Ishita Jena
Regd. No. : 2141010031

Gayatri Sahu
Regd. No. : 2141019146

Soumya Shree Dash
Regd. No. : 2141004097

Pratik Swarup Mishra
Regd. No. : 2141013122

N Nirupsha
Regd. No. : 2141014127

DATE : 12-01-24
PLACE : BHUBANESWAR

Abstract

This project introduces a robust implementation of the classic Tic Tac Toe game utilizing a Client-Server architecture facilitated by Java's socket programming. The primary objective is to establish a reliable and interactive networked gaming platform where multiple players can engage in the game remotely. The architecture comprises two core elements: a server responsible for orchestrating game sessions and multiple clients connecting to the server for gameplay.

The server component operates as the central point of coordination, managing incoming connections from clients, allocating game rooms, and synchronizing gameplay actions among participants. Employing multithreading, the server ensures concurrent handling of client requests, enabling a smooth and responsive gaming experience. Furthermore, the server enforces game rules and validates player moves, maintaining fair and consistent gameplay across all connected clients.

On the client side, users interact with the server through a simple and intuitive interface. Upon establishing a connection, clients can either join existing game rooms or create new ones, inviting other players to participate. The communication protocol between clients and the server is defined meticulously, ensuring seamless transmission of game states, player moves, and status updates. Additionally, error handling mechanisms are implemented to address network interruptions or unexpected events, preserving the stability and continuity of ongoing games.

Client-Server model is a computing architecture where programs or devices (clients) request services or resources from a central server, which fulfills these requests, manages resources, and provides services. Socket programming is a mechanism that enables communication between processes over a network by using sockets, endpoints for sending and receiving data, and encompasses protocols for data transmission. A socket is an endpoint for communication between two machines over a network. It allows bidirectional data flow and consists of an IP address and a port number. Multithreading is a programming technique that allows multiple threads within a process to execute independently.

The project provides a practical demonstration of fundamental concepts in networking, such as the Client-Server model, where the server acts as a centralized authority managing multiple client interactions. Socket programming in Java plays a pivotal role in establishing communication channels between these entities, facilitating the exchange of data packets over a network. Sockets, as communication endpoints, enable bidirectional data flow, allowing clients and servers to send and receive information. This project's implementation encompasses these networking fundamentals, showcasing their application in creating an engaging and scalable multiplayer game environment.

Contents

1 Introduction

In an age where digital interconnectedness transcends physical boundaries, the fusion of classic games with cutting-edge technology presents an intriguing intersection of tradition and innovation. The project at hand ventures into the realm of networked gaming, marrying the simplicity of the timeless Tic Tac Toe with the sophisticated architecture of Client-Server communication, utilizing Java's versatile socket programming capabilities. This ambitious endeavor seeks not only to recreate a beloved game but to pioneer a platform where individuals across the globe converge in real-time, engaging in spirited battles of wit and strategy.

At its core, this project stands as a testament to the transformative potential of networked applications, embodying the ethos of collaborative gaming. The Client-Server architecture serves as the backbone, orchestrating a harmonious interaction between multiple participants. The server, akin to a conductor, orchestrates the symphony of gameplay, overseeing connections, arbitrating moves, and broadcasting the evolving game state to all involved clients. On the client side, players traverse the digital realm, interacting with the server to partake in the strategic dance of Xs and Os, fostering an immersive and dynamic gaming experience.

However, this initiative extends beyond mere recreation. It assumes the mantle of an educational platform, offering a portal into the intricate workings of networked systems and the nuances of socket programming in Java. The project peels back the layers of abstraction, unraveling the complexities of establishing connections, transmitting data, and synchronizing actions across disparate entities in a networked environment. It serves as a repository of knowledge, fostering an understanding of error handling, concurrency, and the orchestration of multiple client-server interactions.

Furthermore, this project encapsulates the spirit of innovation, serving as a catalyst for a paradigm shift in how traditional games integrate with modern technology. It unveils a world where boundaries blur, where individuals from diverse corners of the globe converge in a shared digital arena. Beyond entertainment, it showcases the transformative power of collaborative technology, transcending language barriers and geographical divides to foster a global community united in play and camaraderie.

As we embark on this journey of exploration, innovation, and collaboration, this project stands as a testament to the harmonious synergy between tradition and technology. It underscores the infinite possibilities and the unifying potential embedded within the realm of networked applications, promising an immersive and transformative experience for enthusiasts, learners, and gaming aficionados alike.

2 Problem Statement

Explanation of Problem: The objective of this project is to design and implement a Client-Server Tic Tac Toe game utilizing console interaction in Java. The system aims to allow multiple players to engage in the game remotely, entering their moves through the console and observing the game's progression and outcomes via the compiler. The project necessitates the development of both server-side and client-side applications, with the server orchestrating game sessions and managing player interactions, while clients connect to the server and participate in the game through console inputs.

Identification of Elements/Object Entered Through Console:

- Server Application: Upon execution, the server application prompts for initialization, including port number and configurations. Once active, the server listens for incoming client connections and coordinates game sessions. Messages regarding game states, moves, and notifications are logged and displayed on the server console.
- Client Application: Each client application prompts for server connection details, such as IP address and port number. Once connected, clients await game initiation signals and their assigned symbols (X or O). Players enter their moves via the console, providing row and column numbers to indicate their placement on the Tic Tac Toe grid. The compiler displays updated game states, opponent moves, and end-of-game results.

Result Reflection in the Compiler: The console interaction reflects the game's progress, displaying the game grid, moves made by both players, and the final outcome (win, loss, or draw). The compiler output showcases a textual representation of the Tic Tac Toe grid after each move, enabling players to observe the game's state and make informed decisions for their subsequent moves. Additionally, end-of-game messages indicating the winner or a draw are displayed in the compiler upon game completion.

Constraints:

- Input Validation: The system rigorously validates player inputs to ensure moves are within the grid boundaries and in unoccupied cells. This prevents disruptive erroneous inputs during gameplay.
- Network Connectivity: The game's reliability hinges on stable network connections between the server and clients. The project implements robust error-handling mechanisms to maintain game integrity in the face of potential network disruptions.
- Concurrency and Synchronization: To ensure fair gameplay among multiple clients, the project incorporates multithreading and synchronization techniques.

3 Methodology

Algorithm 1 **Server-side game handling**

```
Create empty list *clientSockets*
Create a game board representation (e.g., a 3x3 matrix)
Initialize the game board
while true do
    Accept incoming connections from clients
    Add new client socket to the list
    if number of connected clients is 2 then
        Notify clients that the game can start
        Assign symbols (X and O) to the clients
        Send symbols to respective clients
        Send initial game state to clients
    while game not over do
        Receive move from a client
        Validate move
        Update game board with the move
        Check for winning condition or a tie
        Send updated game state to clients
        if game over then
            Notify clients about the game result
            Close connections
            break out of the loop
```

- Initialization: The server initializes its socket, creates a list to hold client sockets, and sets up the game board.
- Connection Handling: Continuously accepts incoming connections from clients and adds them to the list. When the required number of clients (two in this case) is reached, it notifies them that the game can start.
- Game Management Loop: It enters a loop where it manages the game until it's over. It listens for moves from clients, validates these moves, updates the game board, and checks for win/tie conditions.
- End Game Handling: When the game is over, it notifies clients of the result, closes connections, and exits the loop, ending the server's game handling process.
- Continuous Operation: This algorithm perpetually runs, allowing new connections to join and managing the game for connected clients until the server is stopped.

Algorithm 2 Client-side game handling

```
Initialize client socket
Connect to the server
while true do
    Receive game initialization message from server
    Receive assigned symbol (X or O) from server
    Display initial game state
    while game not over do
        if it's this client's turn then
            Prompt user for move input
            Send move to server
            Receive updated game state from server
            Display updated game state
        else
            Wait for opponent's move
            Receive opponent's move and updated game state from server
            Display updated game state
        if game over then
            Display game result
            Close connection
            break out of the loop
```

- Initialization and Connection: The client initializes its socket and establishes a connection to the server.
- Receiving Initial Information: It waits to receive game initialization messages and its assigned symbol (X or O) from the server. Upon receiving this data, it displays the initial game state.
- Game Playing Loop: It enters a loop to manage the game until it's over. If it's the client's turn, it prompts the user for input, sends the move to the server, receives the updated game state, and displays it. Otherwise, it waits for the opponent's move and updates its game state accordingly.
- End Game Handling: When the game is over, it displays the result and closes the connection to the server, exiting the loop and ending the client's game interaction.
- Continual Interaction: Similar to the server, the client algorithm continuously operates to handle game interactions until manually terminated or disconnected from the server.

4 Implementation

Server.java

```java
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {
    public static void main(String[] args) throws
       IOException {
        ServerSocket serverSocket = new
           ServerSocket(2000);
        Socket socket = null;
        while (true) {
            System.out.println("waiting client
               1..........");
            socket = serverSocket.accept();
            Client c1 = new Client(socket);
            System.out.println("client 1
               joined.........");
            System.out.println("waiting client
               2..........");
            socket = serverSocket.accept();
            Client c2 = new Client(socket);
            System.out.println("client 2
               joined.........");
            GameEngine ge = new GameEngine(c1, c2);
            ge.run();
        }
    }
}
```

- The Server continuously waits for and accepts two client connections sequentially (c1 and c2) using `serverSocket.accept()`, establishing communication with the clients.
- Once both clients are successfully connected, the Server instantiates a `GameEngine` with these clients (c1 and c2) to manage and run the game between them.

Client.java

```java
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
public class Client {
    Socket socket;
    InputStream is;
    OutputStream os;
    String name;
    Client(Socket socket) throws IOException {
        this.socket = socket;
        is = socket.getInputStream();
        os = socket.getOutputStream();
        initialSetup();
    }
    private void initialSetup() {
        System.out.println("Client Name");
        write("Type your name: ");
        name = read();
        System.out.println(name);
    }
    public String read(){
        String msg = "";
        boolean exit = false;
        while (!exit){
            try {
                if (is.available() > 0) {
                    int d;
                    while ((d = is.read()) != 38) {
                        msg = msg + (char) d;
                    }
                    exit = true;
                }
            } catch (IOException e) {
                System.out.println("Error reading
                  msg..........");
```

Client.java (Continued...)

```
                }
            }
            return msg;
        }
        public void write(String msg){
            try {
                os.write((msg+"&").getBytes());
                os.flush();
            } catch (IOException e) {
                System.out.println("Error sending
                   msg...........");
            }
        }
        public void close() {
            try {
                socket.close();
                os.close();
                is.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

- Initialization via Constructor: The Client class initializes a client instance by taking a Socket which facilitates the communication between the client and the server.
- Stream Handling: It sets up input and output streams (`InputStream` and `OutputStream`) to read and write data through the established socket connection.
- Initial Setup: The `initialSetup()` method prompts the user to enter their name and then reads the input using the `read()` method
- Read and Write Methods: The `read()` method reads input data from the input stream and accumulates it into a String until encountering the delimiter. The `write()` method sends a message to the server by writing to the output stream.
- Closing the Client: The `close()` method closes the socket connection along with the associated input and output streams when called.

TicTacToe.java

```
public class TicTacToe {
    int[] array;
    int count;
    TicTacToe() {
        array = new int[9];
        count = 0;
    }

    public void cardinal(String msg, String mark)
       throws PositionAlreadyMarkedException,
       GameDrawException, PlayerWonException {
        int position = Integer.parseInt(msg);
        int m = mark.equals("x") ? 1 : -1;
        markZeroesCrosses(position, m);
        continueGameOrNot(m);
    }
    private void markZeroesCrosses(int position, int
       mark) throws PositionAlreadyMarkedException {
        if (array[position - 1] == 0) {
            array[position - 1] = mark;
            count++;
        } else
            throw new
               PositionAlreadyMarkedException();
    }
    private boolean continueGameOrNot(int mark)
       throws GameDrawException, PlayerWonException {
        if (count == 9)
            throw new GameDrawException();
        else {
            threes(mark == 1 ? 3 : -3);
        }
        return true;
    }
    private void threes(int sum) throws
       PlayerWonException {
        if (array[0] + array[1] + array[2] == sum)
```

TicTacToe.java (Continued...)

```
                throw new PlayerWonException("1,2,3");
            else if (array[0] + array[3] + array[6] ==
               sum)
                throw new PlayerWonException("1,4,7");
            else if (array[2] + array[5] + array[8] ==
               sum)
                throw new PlayerWonException("3,6,9");
            else if (array[6] + array[7] + array[8] ==
               sum)
                throw new PlayerWonException("7,8,9");
            else if (array[0] + array[4] + array[8] ==
               sum)
                throw new PlayerWonException("1,5,9");
            else if (array[1] + array[4] + array[7] ==
               sum)
                throw new PlayerWonException("2,5,8");
            else if (array[2] + array[4] + array[6] ==
               sum)
                throw new PlayerWonException("3,5,7");
            else if (array[3] + array[4] + array[5] ==
               sum)
                throw new PlayerWonException("4,5,6");
        }
        public String arrayToString() {
            String msg = "";
            for (int i = 1; i < 10; i++) {
                String mark = array[i - 1] == 0 ? " " :
                   array[i - 1] == -1 ? "O" : "X";
                msg += String.format("%2s",mark);
                if (i % 3 == 0) {
                    msg += "\n---------\n";
                }
                else {
                    msg += "|";
                }
            }
```

TicTacToe.java (Continued...)

```
        msg += "\n";
        return msg;
    }
}
```

- All game logic for the Tic-Tac-Toe game are implemented in `TicTacToe.java`.

GameEngine.java

```
import java.io.IOException;
public class GameEngine implements Runnable {
    Client client1;
    Client client2;
    TicTacToe game;
    public GameEngine(Client client1, Client client2)
       throws IOException {
        this.client1 = client1;
        this.client2 = client2;
        game = new TicTacToe();
    }
    @Override
    public void run() {
        client1.write("initial state\n" +
           game.arrayToString());
        client2.write("initial state\n" +
           game.arrayToString());
        boolean exit = false;
        boolean chance = true;
        String c1Mark = "x";
        String c2Mark = "o";
        while (!exit) {
            if (chance) {
                exit = play(client1, client2, c1Mark);
                chance = false;
            }
```

GameEngine.java (Continued...)

```
            else {
                exit = play(client2, client1, c2Mark);
                chance = true;
            }
        }
        client1.close();
        client2.close();
    }
    private boolean play(Client c1, Client c2, String
       mark) {
        String msg;
        try {
            boolean marked = false;
            while (!marked) {
                try {
                    c1.write("Please enter from 1-9:
                       ");
                    msg = c1.read();
                    game.cardinal(msg, mark);
                    marked = true;
                } catch
                   (PositionAlreadyMarkedException ge)
                   {
                    c1.write("Position already
                       occupied\nSelect another");
                }
            }
            c1.write(game.arrayToString());
            c2.write(game.arrayToString());
            c1.write("Wait for opponents
               move.........");
            c2.write("Your move!!!...........");
        } catch (PlayerWonException pwe) {
            c1.write("\n....Congrats!!!.....\n.....You
               won " + pwe);
            c2.write("\n.........You Lost " + pwe);
```

GameEngine.java (Continued...)

```
                c1.write(game.arrayToString());
                c2.write(game.arrayToString());
                return true;
            } catch (GameDrawException gde) {
                c1.write("..........Game
                   Draw!!!..........");
                c2.write("..........Game
                   Draw!!!..........");
                c1.write(game.arrayToString());
                c2.write(game.arrayToString());
                return true;
            }
            return false;
        }
    }
```

- Client Communication and Interaction: Manages communication between the clients (client1 and client2) by facilitating turn-based moves, handling exceptions for invalid moves or game-ending conditions, and updating clients with game progress and outcomes.
- Initialization and Game Setup: Upon creation, GameEngine initializes with references to client1 and client2, creating an instance of the `TicTacToe` game.
- Initial State Transmission: Sends the initial state of the game board to both clients via `client1.write()` and `client2.write()`, allowing them to visualize the initial layout.
- Turn-Based Moves: Alternates between the clients (client1 and client2), prompting them to make moves through `c1.write("Please enter from 1-9:  ")`. It reads the input using c1.read() and processes the moves through the `game.cardinal(msg, mark)` method.
- Game Progress Updates: Updates both clients with the current state of the game board after every move.
- End Game Handling: Handles game termination conditions, such as a player winning (`PlayerWonException`) or a draw (`GameDrawException`), notifying clients of the outcome and sending the final game board state before closing the client connections.

GameException.java

```
public class GameException extends Exception {}
class PositionAlreadyMarkedException extends
  GameException {}
class GameDrawException extends GameException {}
class PlayerWonException extends GameException {
    String pos;
    PlayerWonException(String pos) {
        this.pos = pos;
    }
    @Override
    public String toString() {
        return "pos='" + pos;
    }
}
```

- All exceptions are handled in this code.

ClientApp.java

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.util.Scanner;
public class ClientApp {
    public static void main(String[] args) throws
       IOException {
        Socket socket = new Socket("localhost", 2000);
        OutputStream os = socket.getOutputStream();
        InputStream is = socket.getInputStream();
        Thread read = new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    try {
```

}

ClientApp.java (Continued...)

```
                    if (is.available() > 0) {
                        int d = 0;
                        String msg = "";
                        while ((d = is.read()) !=
                           38) {
                            msg = msg + (char) d;
                        }
                        System.out.println(msg);
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    });
    read.start();
    Thread write = new Thread(new Runnable() {
        @Override
        public void run() {
            Scanner sc = new Scanner(System.in);
            while (true) {
                String msg = sc.nextLine();
                try {
                    os.write((msg +
                       "&").getBytes());
                    os.flush();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    });
    write.start();
  }
}
```

- Socket Initialization: The main() method establishes a socket connection to the server running on localhost (127.0.0.1) at port 2000 using `Socket socket = new Socket("localhost", 2000)`.

- Input and Output Streams: Creates input (`InputStream`) and output (`OutputStream`) streams from the socket to receive and send data to the server(`socket.getInputStream()` and `socket.getOutputStream()`).
- Read Thread Setup: Starts a separate thread (read) that continuously listens for incoming messages from the server through the input stream. When a message arrives, it reads the data and prints it to the console.
- Write Thread Setup: Starts another thread (write) that reads user input from the console (`Scanner sc = new Scanner(System.in)`) and sends it to the server through the output stream. It continuously listens for user input and transmits it to the server.

ClientApp2.java

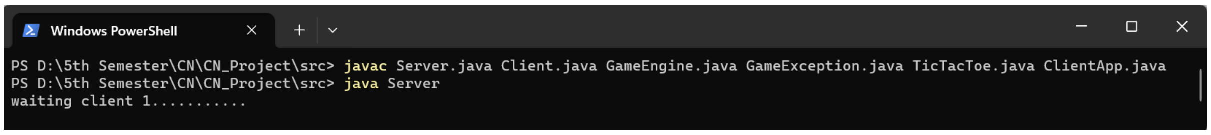
```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
import java.util.Scanner;
public class ClientApp2 {
    public static void main(String[] args) throws
       IOException {
        Socket socket = new Socket("localhost", 2000);
        OutputStream os = socket.getOutputStream();
        InputStream is = socket.getInputStream();
        Thread read = new Thread(new Runnable() {
            @Override
            public void run() {
                while (true) {
                    try {
                        if (is.available() > 0) {
                            int d = 0;
                            String msg = "";
                            while ((d = is.read()) !=
                               38) {
                                msg = msg + (char) d;
```

ClientApp2.java (Continued...)

```
                              }
                              System.out.println(msg);
                          }
                      } catch (IOException e) {
                          e.printStackTrace();
                      }
                  }
              }
          });
          read.start();
          Thread write = new Thread(new Runnable() {
              @Override
              public void run() {
                  Scanner sc = new Scanner(System.in);
                  while (true) {
                      String msg = sc.nextLine();
                      try {
                          os.write((msg +
                             "&").getBytes());
                          os.flush();
                      } catch (IOException e) {
                          e.printStackTrace();
                      }
                  }
              }
          });
          write.start();
      }
  }
```

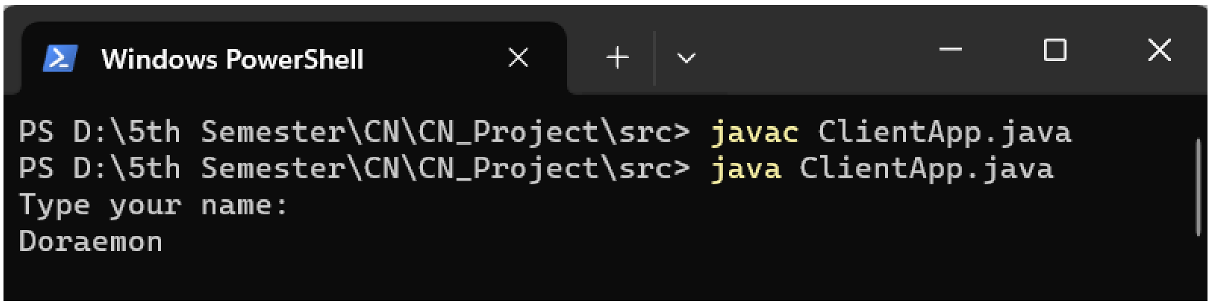
- Since, atleast two clients are required to start the game, another client is created via the `ClientApp2.java`.
- If both server and client are run in the same system, then `localhost` is used in place of IP address.
- If server and client are run in different systems, then `localhost` is replaced with IP address of corresponding system.

5 Results & Interpretation



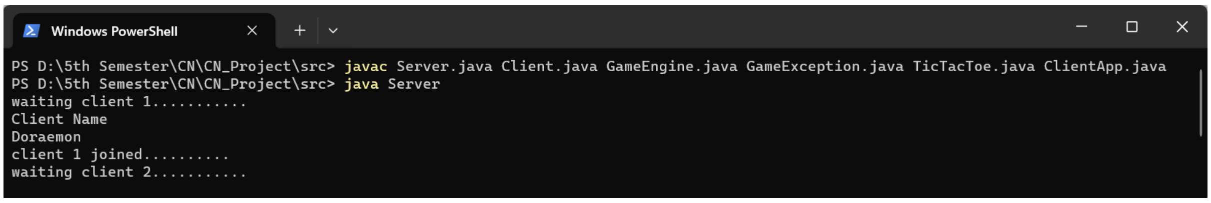
```
PS D:\5th Semester\CN\CN_Project\src> javac Server.java Client.java GameEngine.java GameException.java TicTacToe.java ClientApp.java
PS D:\5th Semester\CN\CN_Project\src> java Server
waiting client 1..........
```

- All programs are compiled and `Server.java` is run.
- The server is waiting for the first client to connect.



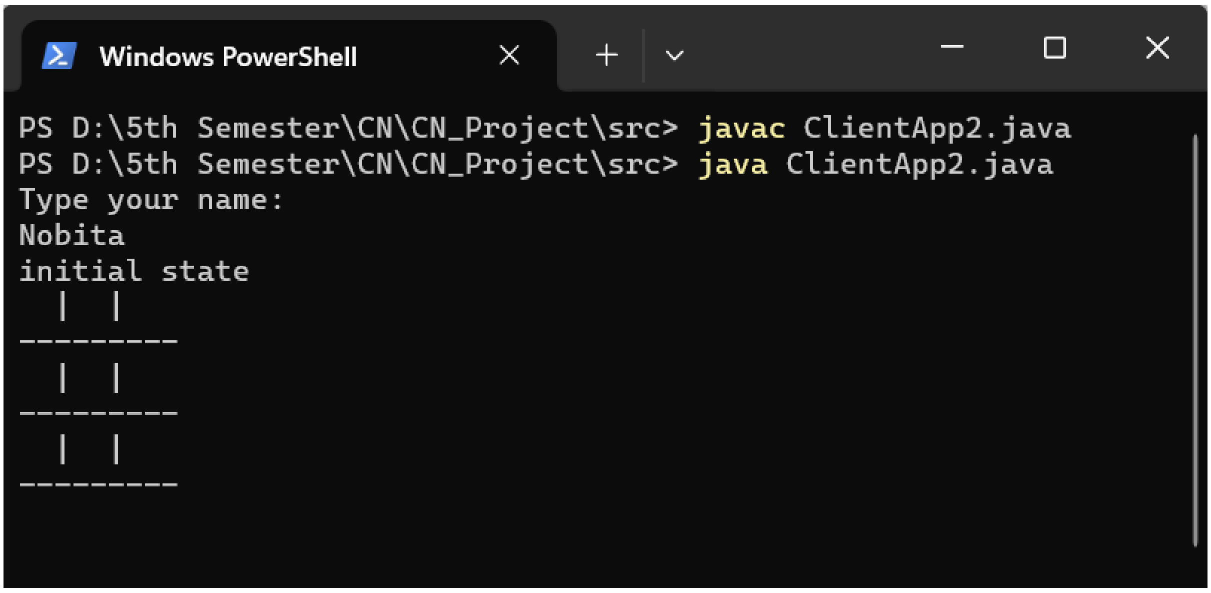
```
PS D:\5th Semester\CN\CN_Project\src> javac ClientApp.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp.java
Type your name:
Doraemon
```

- The first client connects to the server via the `ClientApp.java` and enters its username.



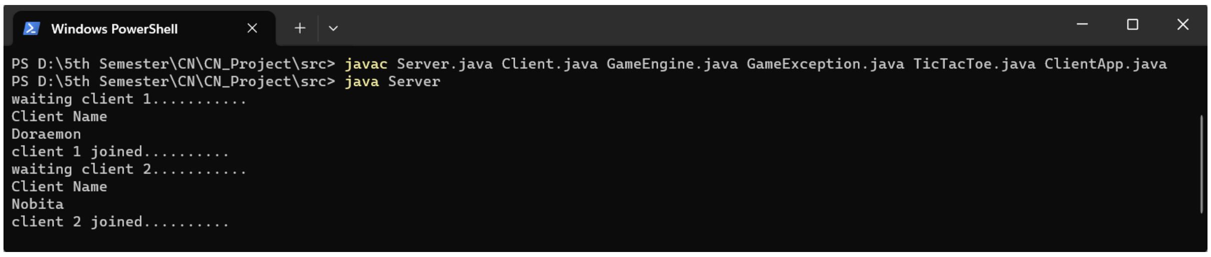
```
PS D:\5th Semester\CN\CN_Project\src> javac Server.java Client.java GameEngine.java GameException.java TicTacToe.java ClientApp.java
PS D:\5th Semester\CN\CN_Project\src> java Server
waiting client 1..........
Client Name
Doraemon
client 1 joined.........
waiting client 2..........
```

- The first client is successfully connected with the server and the server is now waiting for the second client to connect.



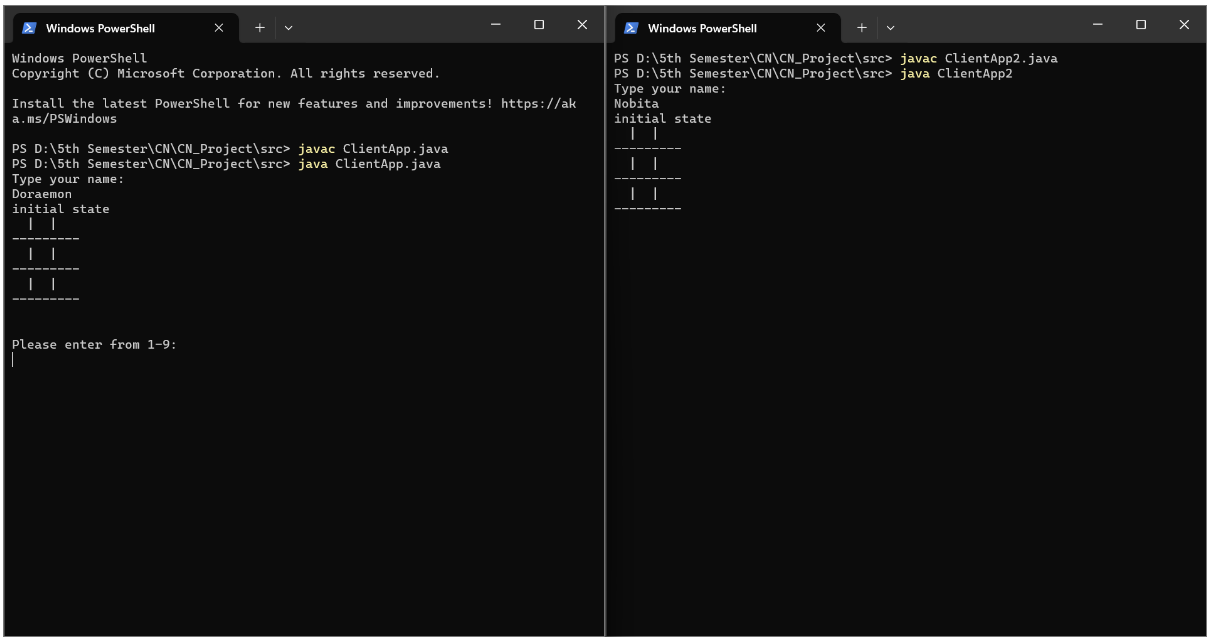
```
PS D:\5th Semester\CN\CN_Project\src> javac ClientApp2.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp2.java
Type your name:
Nobita
initial state
  |  |
---------
  |  |
---------
  |  |
---------
```

- The second client successfully connects to the server via the `ClientApp2.java` and enters its username.



```
PS D:\5th Semester\CN\CN_Project\src> javac Server.java Client.java GameEngine.java GameException.java TicTacToe.java ClientApp.java
PS D:\5th Semester\CN\CN_Project\src> java Server
waiting client 1..........
Client Name
Doraemon
client 1 joined.........
waiting client 2..........
Client Name
Nobita
client 2 joined.........
```

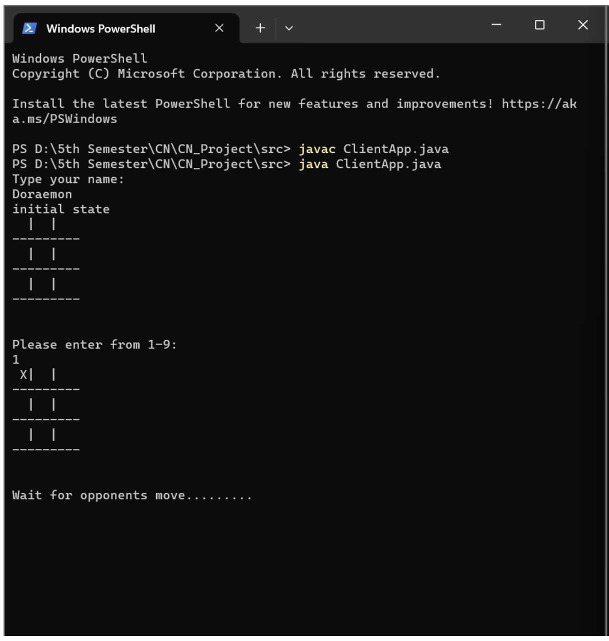
- Both clients are successfully connected to the server.



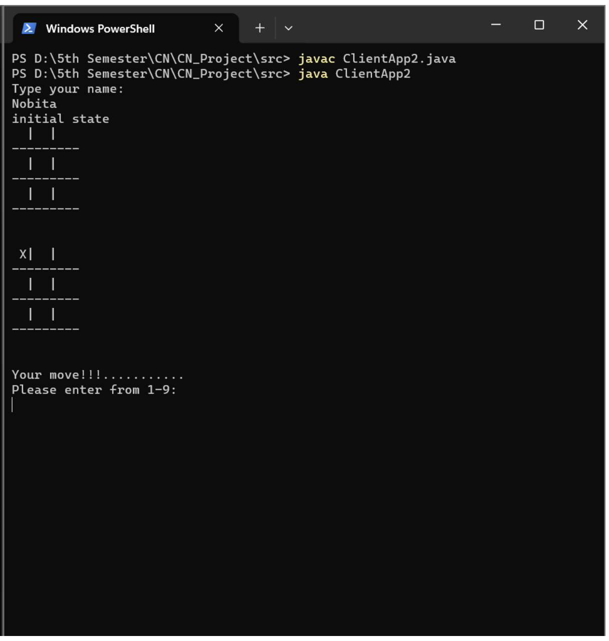
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

PS D:\5th Semester\CN\CN_Project\src> javac ClientApp.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp.java
Type your name:
Doraemon
initial state
  |  |
---------
  |  |
---------
  |  |
---------


Please enter from 1-9:
```

```
PS D:\5th Semester\CN\CN_Project\src> javac ClientApp2.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp2
Type your name:
Nobita
initial state
  |  |
---------
  |  |
---------
  |  |
---------
```

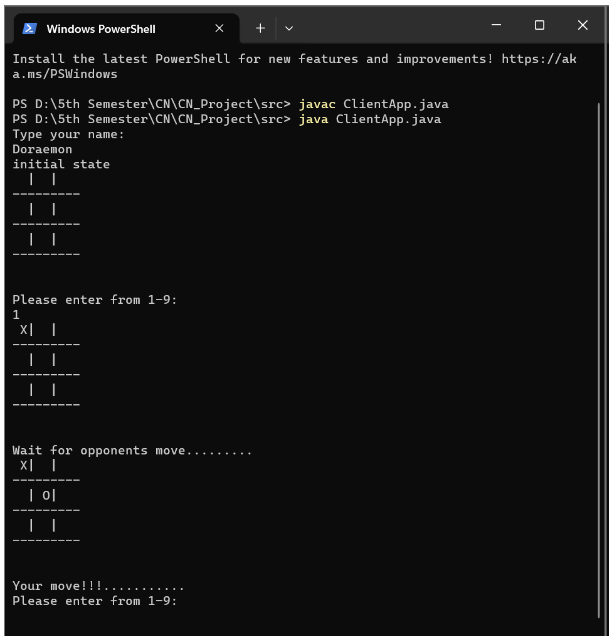
- The game starts and both the players (client1 and client2) are shown the initial Tic-Tac-Toe game board.
- The first client is assigned 'X' and the second client is assigned '0'.
- The server asks the first client to make a move first and then the second client.
- After each entry the updated game board is displayed to both players (clients).
- This continues until either of them wins or the game ends in a draw.



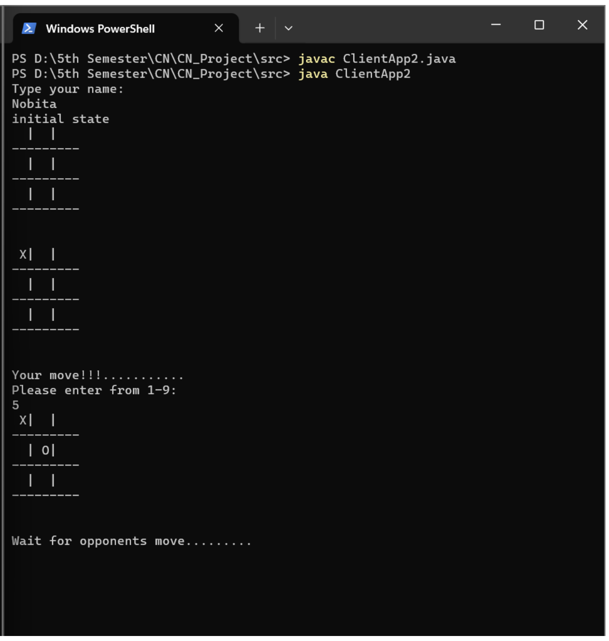
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

PS D:\5th Semester\CN\CN_Project\src> javac ClientApp.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp.java
Type your name:
Doraemon
initial state
  |  |
---------
  |  |
---------
  |  |
---------


Please enter from 1-9:
1
 X|  |
---------
  |  |
---------
  |  |
---------


Wait for opponents move.........
```



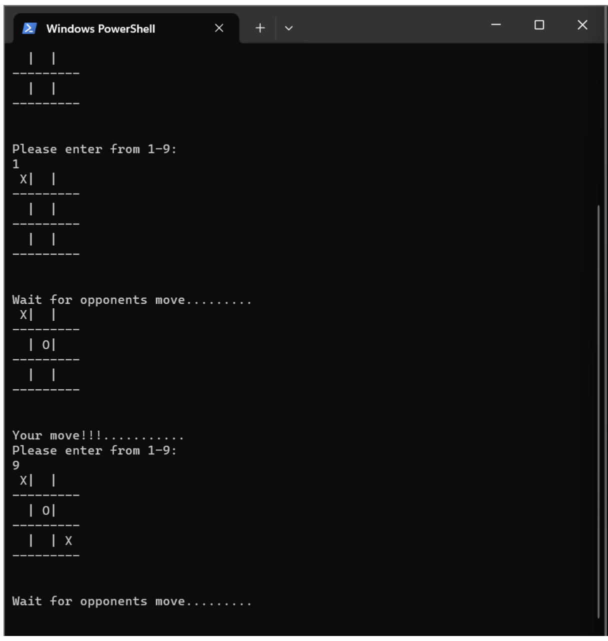
```
PS D:\5th Semester\CN\CN_Project\src> javac ClientApp2.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp2
Type your name:
Nobita
initial state
  |  |
---------
  |  |
---------
  |  |
---------


 X|  |
---------
  |  |
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
```



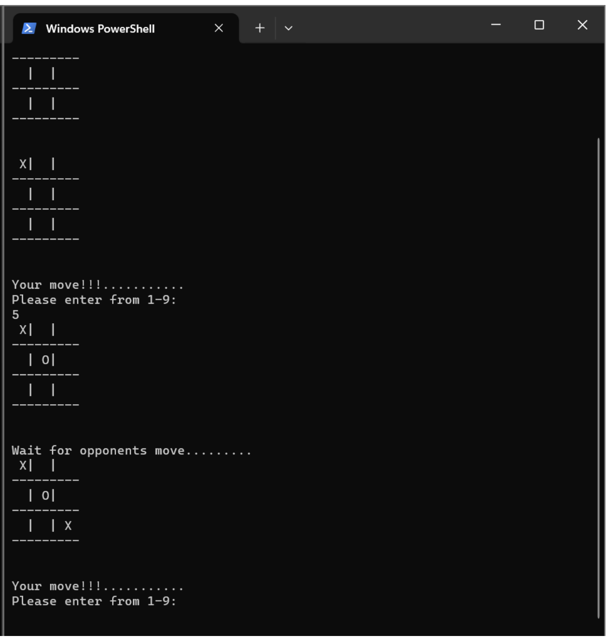
```
Install the latest PowerShell for new features and improvements! https://ak
a.ms/PSWindows

PS D:\5th Semester\CN\CN_Project\src> javac ClientApp.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp.java
Type your name:
Doraemon
initial state
  |  |
---------
  |  |
---------
  |  |
---------


Please enter from 1-9:
1
 X|  |
---------
  |  |
---------
  |  |
---------


Wait for opponents move.........
 X|  |
---------
  | O|
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
```



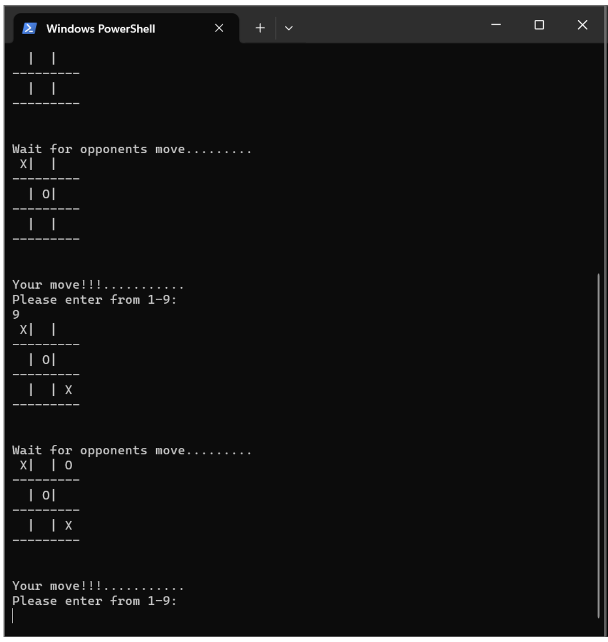
```
PS D:\5th Semester\CN\CN_Project\src> javac ClientApp2.java
PS D:\5th Semester\CN\CN_Project\src> java ClientApp2
Type your name:
Nobita
initial state
  |  |
---------
  |  |
---------
  |  |
---------


 X|  |
---------
  |  |
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
5
 X|  |
---------
  | O|
---------
  |  |
---------


Wait for opponents move.........
```



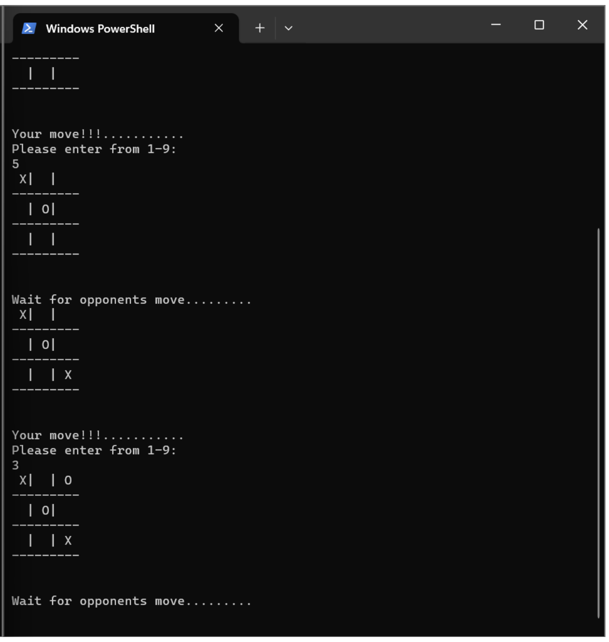
```
  |  |
---------
  |  |
---------


Please enter from 1-9:
1
 X|  |
---------
  |  |
---------
  |  |
---------


Wait for opponents move.........
 X|  |
---------
  | O|
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
9
 X|  |
---------
  | O|
---------
  |  | X
---------


Wait for opponents move.........
```



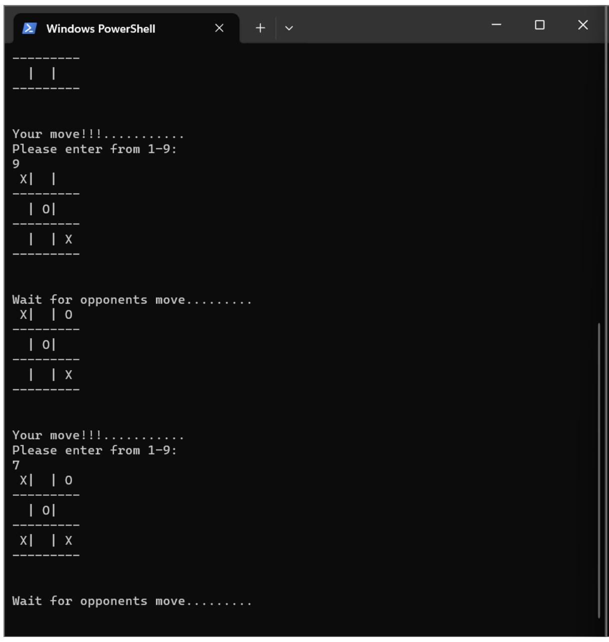
```
---------
  |  |
---------
  |  |
---------


 X|  |
---------
  |  |
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
5
 X|  |
---------
  | O|
---------
  |  |
---------


Wait for opponents move.........
 X|  |
---------
  | O|
---------
  |  | X
---------


Your move!!!...........
Please enter from 1-9:
```



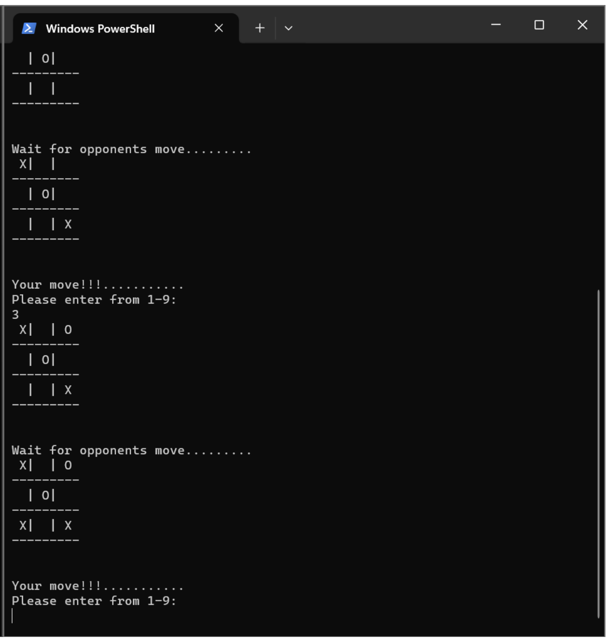
```
  |  |
---------
  |  |
---------


Wait for opponents move.........
 X|  |
---------
  | O|
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
9
 X|  |
---------
  | O|
---------
  |  | X
---------


Wait for opponents move.........
 X|  | O
---------
  | O|
---------
  |  | X
---------


Your move!!!...........
Please enter from 1-9:
```



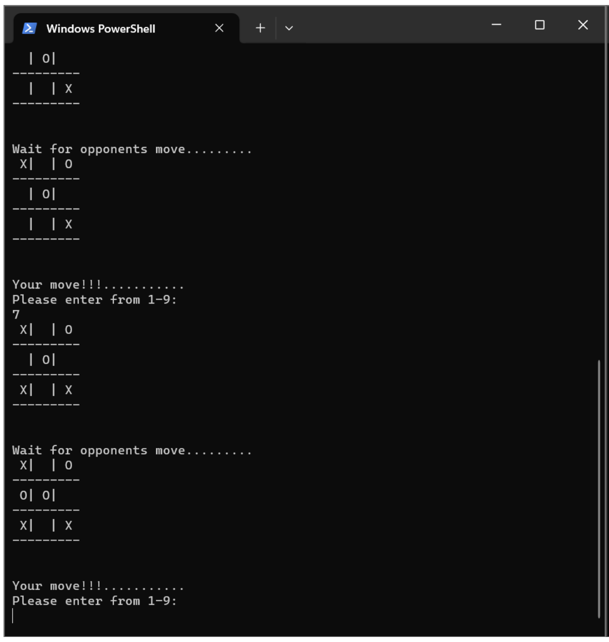
```
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
5
 X|  |
---------
  | O|
---------
  |  |
---------


Wait for opponents move.........
 X|  |
---------
  | O|
---------
  |  | X
---------


Your move!!!...........
Please enter from 1-9:
3
 X|  | O
---------
  | O|
---------
  |  | X
---------


Wait for opponents move.........
```



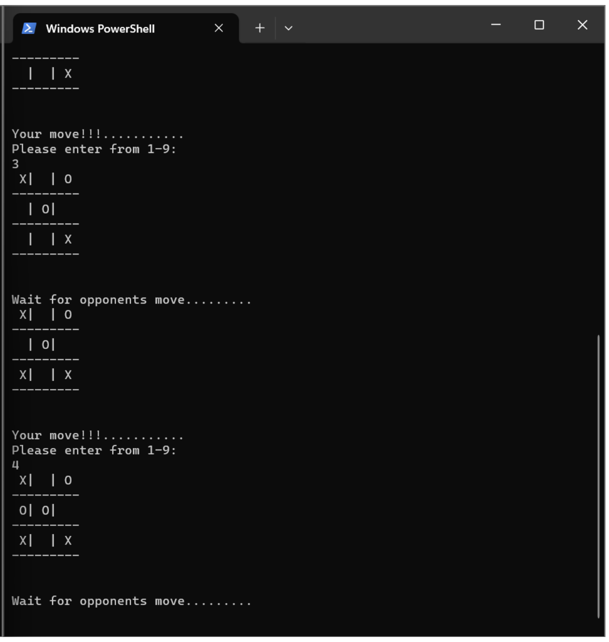
```
---------
  |  |
---------


Your move!!!...........
Please enter from 1-9:
9
 X|  |
---------
  | O|
---------
  |  | X
---------


Wait for opponents move.........
 X|  | O
---------
  | O|
---------
  |  | X
---------


Your move!!!...........
Please enter from 1-9:
7
 X|  | O
---------
  | O|
---------
 X|  | X
---------


Wait for opponents move.........
```



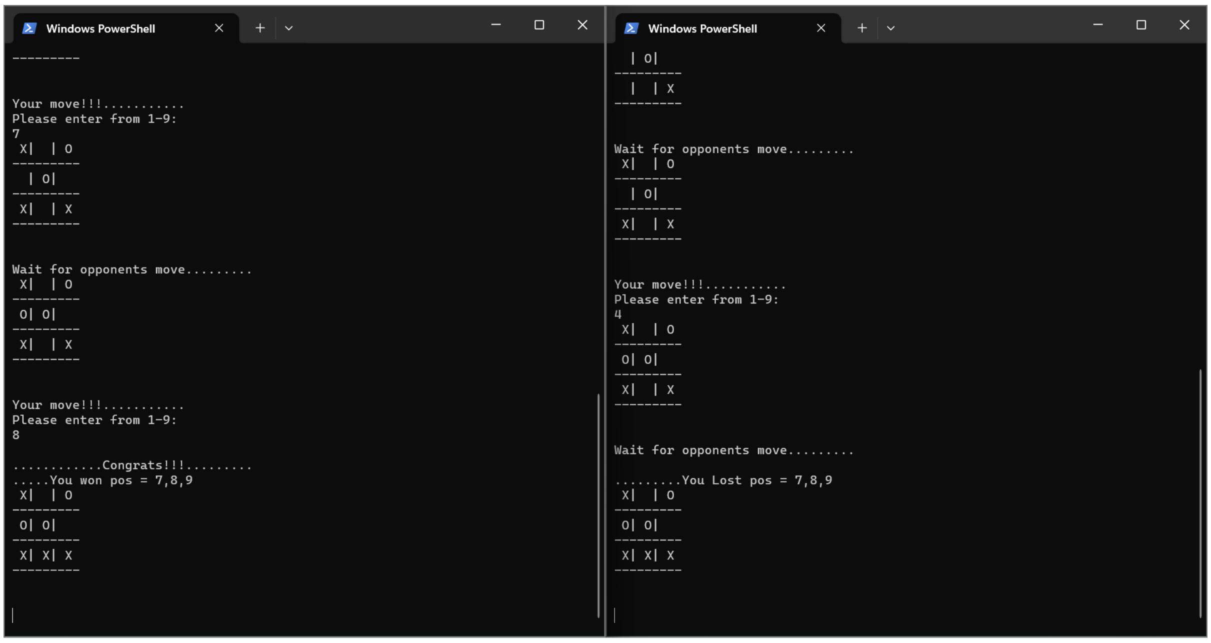
```
  | O|
---------
  |  |
---------


Wait for opponents move.........
 X|  |
---------
  | O|
---------
  |  | X
---------


Your move!!!...........
Please enter from 1-9:
3
 X|  | O
---------
  | O|
---------
  |  | X
---------


Wait for opponents move.........
 X|  | O
---------
  | O|
---------
 X|  | X
---------


Your move!!!...........
Please enter from 1-9:
```



```
  | O|
---------
  |  | X
---------


Wait for opponents move.........
 X|  | O
---------
  | O|
---------
  |  | X
---------


Your move!!!...........
Please enter from 1-9:
7
 X|  | O
---------
  | O|
---------
 X|  | X
---------


Wait for opponents move.........
 X|  | O
---------
 O| O|
---------
 X|  | X
---------


Your move!!!...........
Please enter from 1-9:
```



```
---------
  |  | X
---------


Your move!!!...........
Please enter from 1-9:
3
 X|  | O
---------
  | O|
---------
  |  | X
---------


Wait for opponents move.........
 X|  | O
---------
  | O|
---------
 X|  | X
---------


Your move!!!...........
Please enter from 1-9:
4
 X|  | O
---------
 O| O|
---------
 X|  | X
---------


Wait for opponents move.........
```



Windows PowerShell

```
---------


Your move!!!...........
Please enter from 1-9:
7
 X|  | O
---------
  | O|
---------
 X|  | X
---------


Wait for opponents move.........
 X|  | O
---------
 O| O|
---------
 X|  | X
---------


Your move!!!...........
Please enter from 1-9:
8

............Congrats!!!.........
.....You won pos = 7,8,9
 X|  | O
---------
 O| O|
---------
 X| X| X
---------
```

Windows PowerShell

```
  | O|
---------
  |  | X
---------


Wait for opponents move.........
 X|  | O
---------
  | O|
---------
 X|  | X
---------


Your move!!!...........
Please enter from 1-9:
4
 X|  | O
---------
 O| O|
---------
 X|  | X
---------


Wait for opponents move.........

.........You Lost pos = 7,8,9
 X|  | O
---------
 O| O|
---------
 X| X| X
---------
```

- First client wins the game.
- The game board along with the winning positions are displayed.

6 Conclusion

In culmination, the development and implementation of the Client-Server Tic Tac Toe game utilizing console interaction represent an amalgamation of fundamental networking concepts, input validation strategies, and concurrency management. This project has embraced the challenge of transforming a classic game into a networked, console-based experience, emphasizing the seamless interplay between users, the server, and the game interface.

Throughout this endeavor, the project has prioritized user experience, leveraging console-based interactions to enable players from diverse locations to engage in the game. The stringent input validation mechanisms have fortified the system against erroneous inputs, ensuring the integrity of the gameplay by accepting only valid moves within the bounds of the game grid and unoccupied cells. This meticulous validation fosters an immersive gaming environment where players can confidently strategize and execute moves without concerns about inconsistencies or disruptions.

Moreover, the project acknowledges the pivotal role of network connectivity in the gaming experience. By addressing potential network disruptions with robust error-handling mechanisms, the system endeavors to maintain the coherence of ongoing games, minimizing the impact of connectivity issues and preserving the competitive spirit and fairness of the gameplay.

A fundamental aspect of this project lies in managing concurrent interactions among multiple clients. The incorporation of multithreading and synchronization mechanisms at the server end ensures consistent and fair gameplay. These techniques orchestrate simultaneous client requests and synchronize game sessions, mitigating conflicts and upholding a level playing field for all participants.

Furthermore, this project serves as a testament to the synthesis of theoretical concepts and practical implementation. It navigates through the complexities of networking principles, input validation strategies, and concurrent systems, providing a hands-on learning experience for aspiring developers and enthusiasts delving into networked application development.

In summary, the Client-Server Tic Tac Toe game with console interaction encapsulates a convergence of technology, gaming, and collaborative design. It showcases the resilience of networked applications in providing seamless user experiences, overcoming connectivity challenges, and ensuring fairness in multiplayer interactions. This project stands not only as a functional gaming platform but also as an educational tool, inviting exploration and understanding of the intricate dynamics within networked systems.

7 References

1. GeeksforGeeks, 2023. [Online]. Available: https://www.geeksforgeeks.org/socket-programming-in-java/
2. JavaPoint, 2023. [Online]. Available: https://www.javatpoint.com/socket-programming
3. GeeksforGeeks, 2022. [Online]. Available: https://www.geeksforgeeks.org/tic-tac-toe-game-in-java/
4. Java threads. [Online]. Available: https://www.w3schools.com/java/java_threads.asp.
5. The client/server model. [Online]. Available: https://www.ibm.com/docs/en/cics-ts/6.1?topic=programs-clientserver-model.