NARULA INSTITUTE OF TECHNOLOGY

PROJECT REPORT

TIC-TAC-TOE GAME

Name : Ayan Bhattacherjee

Class roll :35

University roll :431221010035

Department: CSBS

Subject Name: COMPUTER NETWORK

LAB Subject code: PCC-CS691

Date: 31-05-2024

Under guidance of

Mrs. Chetena Bhattacharya

Computer Network Faculty at NiT

CERTIFICATE OF APPROVAL

This is to certify that the project entitled "TIC-TAC- TCP/IP SOCKET PROGRAMMING IN JAVA" has been carried by Prabhat Ranjan and submitted as a requirement for the course PCC-CS691: COMPUTER NETWORKS LAB under the degree of Bachelor of Technology (B. TECH) in Computer Science & Engineering of Narula Institute of Technology, Agarpara affiliated to Maulana Abul Kalam Azad University of Technology during the academic year 2021-2025.

It is understood that by this approval the undersigned do not necessarily endorse any of the statements made or opinion expressed therein but approves it only for the purpose for which it is submitted

Submitted By:

Name: Ayan Bhattacherjee

Roll No.: 431221010035

Mrs. Chetana Bhattacharya

Computer Science and Engineering Department

Dr. Jayanta Paul

HOD, Computer Science and Business System Department

ACKNOWLEDGEMENT

I would like to express my profound gratitude to Mrs. Chetena Bhattacharya, of CSE department, of Narula Institute of Technology university for the contributions to the completion of my project titled '**Tic-Tac-Toe Game**'.

I would like to express my special thanks to our mentor Mrs. Chetena Bhattacharya for his time and efforts he provided throughout the year. Your useful advice and suggestions were really helpful to me during the project's completion. In this aspect, I am eternally grateful to you.

I would like to acknowledge that this project was completed entirely by me and not by someone else.

CONTENT

ABSTRACT

This project implements a multiplayer Tic-Tac-Toe game using Java's networking capabilities, demonstrating key concepts of client-server architecture. The game consists of a server application that handles game logic, player management, and communication, and a client application that enables players to interact with the game.

The server is responsible for:
- **Player Connections**: Accepting connections from two players and assigning them roles ('X' and 'O').
- **Game Management**: Maintaining the game state, alternating turns, and validating moves.
- **Communication:** Sending real-time updates to clients about game status, board updates, and game outcomes (win, lose, or draw).

The **client** application allows players to:
- **Connect to the Server**: Establish a connection to the game server.
- **Receive Role Assignments**: Get notified of their assigned role ('X' or 'O').
- **Interact with the Game**: Input their moves, receive updates on the board state, and get notified of game status.
- **User Interface**: Display the current board state in a user-friendly manner and provide prompts for player actions.

The game proceeds with players taking turns to make moves on the 3x3 board. The server validates each move to ensure it is within bounds and not on an already occupied cell. After each move, the server updates the board and checks for win or draw conditions, informing both players of the current game state.

This project highlights the use of Java's `Socket` and `ServerSocket` classes for network communication, along with input/output handling using `BufferedReader` and `PrintWriter`. It provides a practical example of building a real-time, interactive application using client-server architecture, suitable for educational purposes and illustrating fundamental network programming concepts.

INTRODUCTION

The Tic-Tac-Toe game is a classic two-player game where players take turns marking spaces in a 3x3 grid, aiming to be the first to get three of their marks in a horizontal, vertical, or diagonal row. Implementing this game in a networked environment involves creating both a server and client applications, allowing players to connect and play remotely.

This project demonstrates how to build a multiplayer Tic-Tac-Toe game using Java's networking capabilities. It covers the fundamentals of client-server architecture, where the server is responsible for managing the game logic and maintaining the state of the game board, while the clients interact with the game by sending their moves to the server and receiving updates.

Key concepts illustrated in this project include:

- Socket Programming: Using Java's `Socket` and `ServerSocket` classes to establish connections between the server and clients.

- Input/Output Handling: Utilizing `BufferedReader` and `PrintWriter` for reading from and writing to network streams.

- Concurrency: Managing multiple client connections and ensuring proper synchronization of game state and player turns.

- Game Logic: Implementing the rules of Tic-Tac-Toe, including move validation, win condition checks, and draw detection.

- Real-Time Communication: Ensuring that updates to the game board and player statuses are communicated promptly and accurately to both players.

By working through this project, developers can gain hands-on experience with Java networking, learn to handle real-time communication between multiple clients, and understand the intricacies of maintaining game state in a networked environment. This makes it an excellent educational tool for those looking to deepen their understanding of network programming and game development in Java.

IMPLEMENTATION

Server File

```
import java.io.*;
import java.net.*;


public class TicTacToeServer {
   private static char[][] board = new
char[3][3];
   private static char currentPlayer = 'X';
   private static Socket player1;
   private static Socket player2;
   private static PrintWriter player1Out;
   private static PrintWriter player2Out;


   public static void main(String[] args) {
      try {
         ServerSocket serverSocket = new
ServerSocket(5000);
         System.out.println("Server is
running...");
         player1 = serverSocket.accept();
         player1Out = new
PrintWriter(player1.getOutputStream(), true);
         player1Out.println("PLAYER_X");


         player2 = serverSocket.accept();
         player2Out = new
PrintWriter(player2.getOutputStream(), true);
         player2Out.println("PLAYER_O");


         initializeBoard();
while (true) {
   if (currentPlayer == 'X') {
      player1Out.println("YOUR_TURN");
      player2Out.println("WAIT");
      receiveMove(player1, player1Out);
      if (checkWin('X')) {
         player1Out.println("WIN");
         player2Out.println("LOSE");
         break;
      } else if (isBoardFull()) {
         player1Out.println("DRAW");
         player2Out.println("DRAW");
         break;
      }
      currentPlayer = 'O';
   } else {
      player2Out.println("YOUR_TURN");
      player1Out.println("WAIT");
      receiveMove(player2, player2Out);
      if (checkWin('O')) {
         player2Out.println("WIN");
         player1Out.println("LOSE");
         break;
      } else if (isBoardFull()) {
         player1Out.println("DRAW");
         player2Out.println("DRAW");
```

```
                break;
            }
            currentPlayer = 'X';
        }
    }

    serverSocket.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
}

private static void initializeBoard() {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      board[i][j] = '-';
    }
  }
}

private static void receiveMove(Socket playerSocket, PrintWriter playerOut) throws IOException {
  BufferedReader playerIn = new BufferedReader(new InputStreamReader(playerSocket.getInputStream()));
  int move = Integer.parseInt(playerIn.readLine());
  int row = (move - 1) / 3;
  int col = (move - 1) % 3;
  if (board[row][col] == '-') {
    board[row][col] = currentPlayer;
    sendBoardState();
  } else {
    playerOut.println("INVALID_MOVE");
    receiveMove(playerSocket, playerOut);
  }
}

private static void sendBoardState() {
  StringBuilder boardStr = new StringBuilder();
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      boardStr.append(board[i][j]);
    }
  }
  player1Out.println("UPDATE_BOARD:" + boardStr.toString());
  player2Out.println("UPDATE_BOARD:" + boardStr.toString());
}

private static boolean checkWin(char symbol) {
  // Check rows
  for (int i = 0; i < 3; i++) {
    if (board[i][0] == symbol && board[i][1] == symbol && board[i][2] == symbol) {
      return true;
    }
  }
  // Check columns
  for (int j = 0; j < 3; j++) {
```

```
        if (board[0][j] == symbol && board[1][j]
== symbol && board[2][j] == symbol) {
            return true;
        }
    }
    // Check diagonals
    if ((board[0][0] == symbol && board[1][1]
== symbol && board[2][2] == symbol) ||
        (board[0][2] == symbol &&
board[1][1] == symbol && board[2][0] ==
symbol)) {
        return true;
    }
    return false;
  }

  private static boolean isBoardFull() {
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
        if (board[i][j] == '-') {
          return false;
        }
      }
    }
    return true;
  }
}
```

Client File

```
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class TicTacToeClient {
  public static void main(String[] args) {
    try {
      Socket socket = new Socket("localhost",
5000);
      BufferedReader in = new
BufferedReader(new
InputStreamReader(socket.getInputStream()))
;
      PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);
      Scanner scanner = new
Scanner(System.in);
      String role = in.readLine();
      System.out.println("You are player " +
(role.equals("PLAYER_X") ? "X" : "O"));
      char playerSymbol =
(role.equals("PLAYER_X") ? 'X' : 'O');

      while (true) {
        String message = in.readLine();
        if (message.equals("YOUR_TURN")) {
          System.out.println("Your turn.
Enter position (1-9): ");
          int move = scanner.nextInt();
          out.println(move);
        } else if (message.equals("WAIT")) {
          System.out.println("Waiting for
opponent's move...");
```

```
            } else if
(message.startsWith("UPDATE_BOARD")) {
                String[] parts = message.split(":",
2);
                if (parts.length == 2) {
                    String boardStr = parts[1];
                    displayBoard(boardStr);
                }
            } else if (message.equals("WIN")) {

System.out.println("Congratulations! You
won!");
                break;
            } else if (message.equals("LOSE")) {
                System.out.println("Sorry! You
lost!");
                break;
            } else if (message.equals("DRAW")) {
                System.out.println("It's a draw!");
                break;
            }
        }
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    private static void displayBoard(String
boardStr) {
        System.out.println("Current Board:");
        System.out.println("  1   2   3");
        System.out.println("0 " +
boardStr.charAt(0) + " | " + boardStr.charAt(1)
+ " | " + boardStr.charAt(2));
        System.out.println(" -------- ");
        System.out.println("3 " +
boardStr.charAt(3) + " | " + boardStr.charAt(4)
+ " | " + boardStr.charAt(5));
        System.out.println(" -------- ");
        System.out.println("6 " +
boardStr.charAt(6) + " | " + boardStr.charAt(7)
+ " | " + boardStr.charAt(8));
    }
}
```

Output

Server

```
Microsoft Windows [Version 10.0.22631.3593]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ad807\OneDrive\Desktop\Computer Network\TIC-TAC-TOE\tictactoe>java TicTacToeServer
Server is running ...
```

Client

PLAYER X

```
C:\Users\ad807\OneDrive\Desktop\Computer Network\TIC-TAC-TOE\tictactoe>java TicTacToeClient
You are player X
Your turn. Enter position (1-9):
9
Current Board:
  1  2  3
0 - | - | -
  ---------
3 - | - | -
  ---------
6 - | - | X
Waiting for opponent's move ...
Current Board:
  1   2   3
0 - | - | -
  ---------
3 - | O | -
  ---------
6 - | - | X
Your turn. Enter position (1-9):
7
Current Board:
  1   2   3
0 - | - | -
  ---------
3 - | O | -
  ---------
6 X | - | X
Waiting for opponent's move ...
Current Board:
  1   2   3
0 - | - | -
  ---------
3 - | O | -
  ---------
6 X | O | X
Your turn. Enter position (1-9):
2
Current Board:
  1   2   3
0 - | X | -
  ---------
3 - | O | -
  ---------
6 X | O | X
Waiting for opponent's move ...
Current Board:
  1   2   3
0 - | X | -
  ---------
3 - | O | O
  ---------
6 X | O | X
Your turn. Enter position (1-9):
1
Current Board:
  1   2   3
0 X | X | -
  ---------
3 - | O | O
  ---------
6 X | O | X
Waiting for opponent's move ...
Current Board:
  1   2   3
0 X | X | -
  ---------
3 O | O | O
  ---------
6 X | O | X
Sorry! You lost!
```

PLAYER O

```
C:\Users\ad807\OneDrive\Desktop\Computer Network\TIC-TAC-TOE\tictactoe>java TicTacToeClient
You are player O
Waiting for opponent's move ...
Current Board:
  1  2  3
0 - | - | -
  ---------
3 - | - | -
  ---------
6 - | - | X
Your turn. Enter position (1-9):
5
Current Board:
  1   2   3
0 - | - | -
  ---------
3 - | O | -
  ---------
6 - | - | X
Waiting for opponent's move ...
Current Board:
  1   2   3
0 - | - | -
  ---------
3 - | O | -
  ---------
6 X | - | X
Your turn. Enter position (1-9):
8
Current Board:
  1   2   3
0 - | - | -
  ---------
3 - | O | -
  ---------
6 X | O | X
Waiting for opponent's move ...
Current Board:
  1   2   3
0 - | X | -
  ---------
3 - | O | -
  ---------
6 X | O | X
Your turn. Enter position (1-9):
6
Current Board:
  1   2   3
0 - | X | -
  ---------
3 - | O | O
  ---------
6 X | O | X
Waiting for opponent's move ...
Current Board:
  1   2   3
0 X | X | -
  ---------
3 - | O | O
  ---------
6 X | O | X
Your turn. Enter position (1-9):
4
Current Board:
  1   2   3
0 X | X | -
  ---------
3 O | O | O
  ---------
6 X | O | X
Congratulations! You won!
```

CONCLUSION

The implementation of a networked Tic-Tac-Toe game using Java serves as a comprehensive demonstration of core programming and networking concepts, providing a rich learning experience. By constructing both server and client applications, the project encapsulates the essence of real-time multiplayer game development.

1. Networking Fundamentals:
    - The project effectively utilizes Java's Socket and ServerSocket classes to establish network connections, forming the basis of a client-server architecture.
    - The server listens for incoming connections, while clients connect to the server, enabling two players to interact over a network.
2. Client-Server Communication:
    - The use of BufferedReader and PrintWriter ensures efficient data exchange between the server and clients. This is crucial for maintaining the flow of the game and ensuring both players receive timely updates.
    - Real-time communication is established where the server sends game state updates and turn notifications, and clients send their moves to the server.
3. Concurrency Management:
    - The server handles multiple client connections concurrently, ensuring that each player can interact with the game without interference.
    - Proper synchronization ensures that only one player can make a move at a time, while the other is notified to wait, thus maintaining the integrity of the game.
4. Game Logic Implementation:
    - The server encapsulates all game logic, including initializing the game board, validating moves, checking for win or draw conditions, and updating the game state.
    - This centralized management of game rules ensures consistency and fairness, as the server acts as the authoritative source of truth for the game state.
5. Real-Time Updates and User Interaction:
    - The client application is designed to provide a responsive and interactive experience for the player. It prompts the player for moves, displays the current state of the board, and provides feedback based on the server's messages.
    - The board is updated in real-time for both players, ensuring they see the same game state and can make informed decisions based on the current board configuration.

By building this Tic-Tac-Toe game, developers gain hands-on experience with network programming, reinforcing theoretical concepts with practical implementation. It serves as a stepping stone to more complex multiplayer games and networked applications, equipping developers with the skills needed to tackle a variety of networking challenges in software development.