ACKNOWLEDGEMENT

We are greatly indebted to our mini project guide **Ms.Nitu L Pariyal** for her able guidance throughout this work. It We has been an altogether different experience to work with her and we would like to thank her for her help, suggestions and numerous discussions. We gladly take this opportunity to thank **Dr.Mrs. Rajurkar A. M.** (Head of Computer Science & Engineering, MGM' s College of Engineering, Nanded). We are heartily thankful to **Dr. Mrs. Lathkar G. S.** (Director, MGM's College of Engineering, Nanded) for providing facility during progress of mini project in java, also for her kindly help, guidance and inspiration. Last but not least we are also thankful to all those who help directly or indirectly to develop this mini project and complete it successfully.

With Deep Reverence,

Ashitosh Charpilwar

ABSTRACT

The Tic Tac Toe game in Java is a console-based implementation of the classic two-player board game. This game is designed to run in a terminal or command-line interface where players take turns marking spaces on a 3x3 grid. The objective is to place three of their marks (either 'X' or 'O') in a horizontal, vertical, or diagonal line, while preventing the opponent from doing the same.

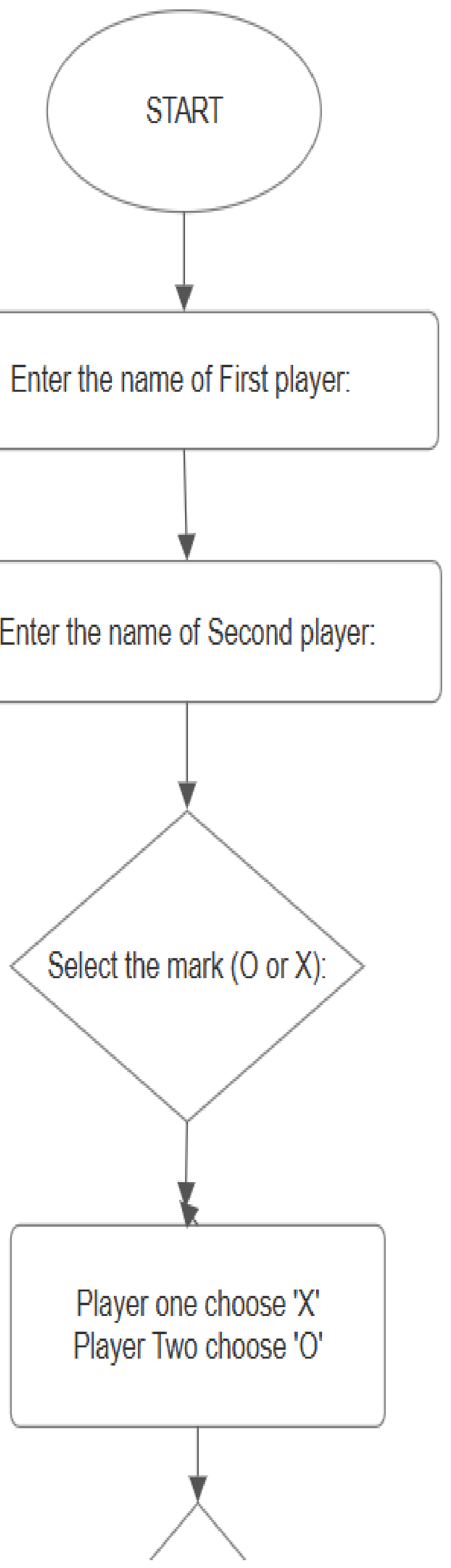
The game utilizes basic Java features such as arrays to represent the game board, loops for turn-based interactions, and conditional statements to check for win conditions, draw, or invalid moves. It also handles user input and provides an interface for players to make their moves, with appropriate error handling to ensure valid input. Once the game ends, it displays the result (win, draw, or ongoing) and allows players to restart the game if desired.

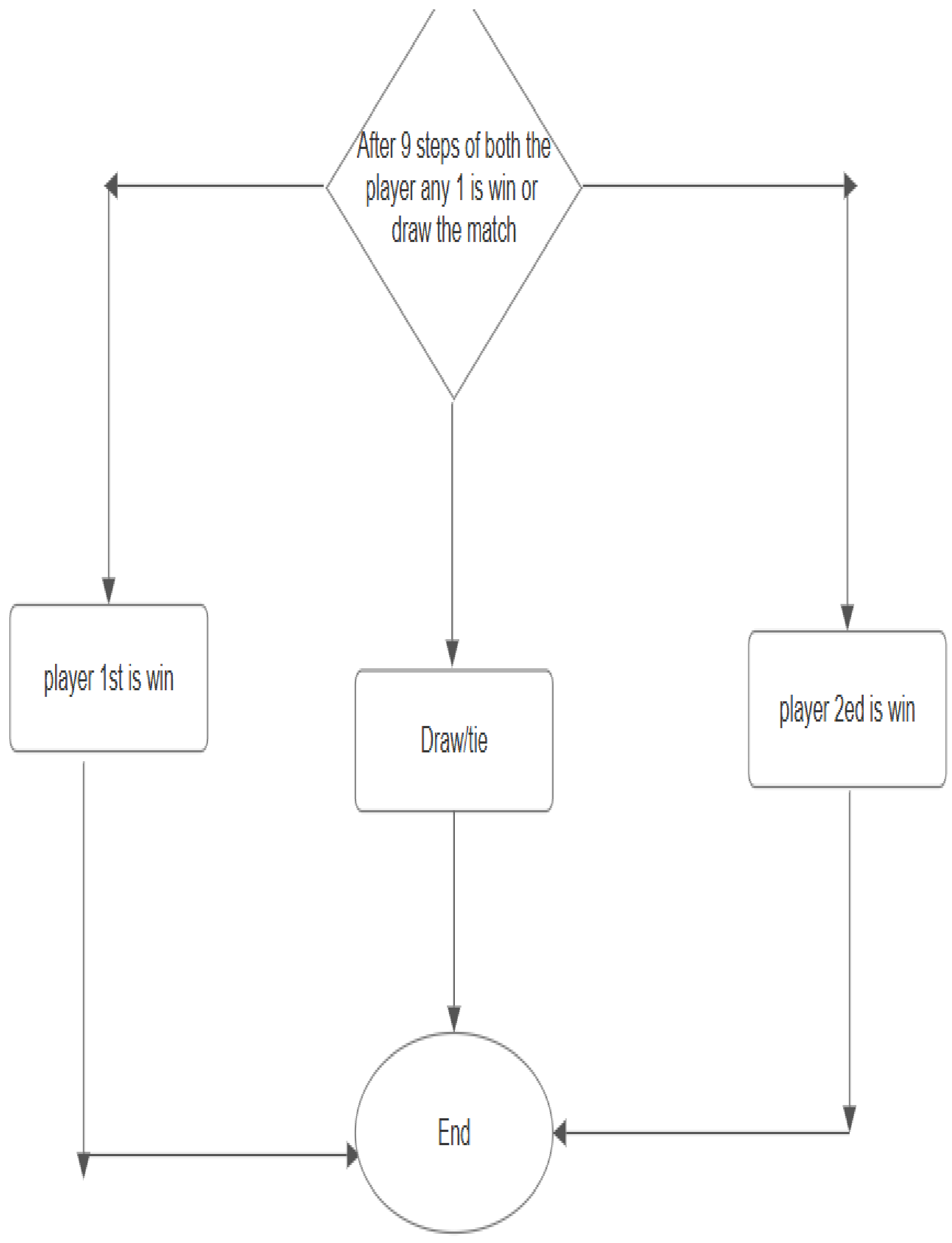
TABLE OF CONTENTS

1.UML Diagram

Tic tac Toe Game
- SIZE : int -table : int[][]
+ main(args : String[]) - players: Player[2] - initialize Table() - display Table() - update Board(row: int, col: int, | | symbol: char): boolean - is Cell Empty (row: int, col: int): Boolean - name: string - symbol: char - make Move(row: int, col: int): Move - row: int - col: int - player: Player - validate Move(): boolean

2.Flowchart


START
Enter the name of First player:
Enter the name of Second player:
Select the mark (O or X):
Player one choose 'X'
Player Two choose 'O'


After 9 steps of both the player any 1 is win or draw the match
player 1st is win
Draw/tie
player 2ed is win
End

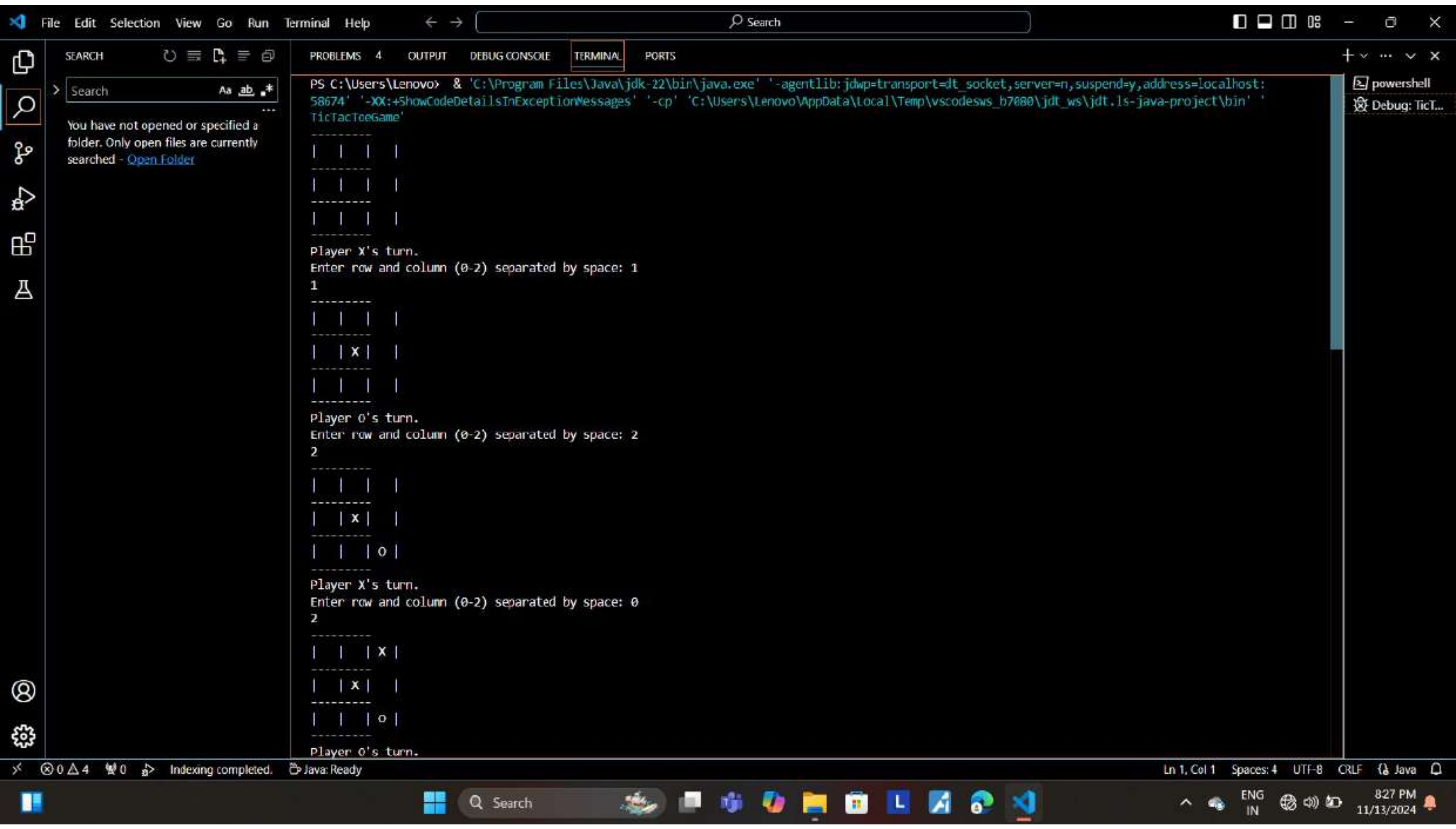
3. Code of TicTacToeGame

```
import java.util.Scanner;
public class TicTacToeGame {

    // 3x3 game board
    private char[][] board;
    private char currentPlayer;

    // Constructor initializes the board and starts with 'X'
    public TicTacToeGame() {
        board = new char[3][3];
        currentPlayer = 'X';
        initializeBoard();
    }

    // Initialize the board with empty spaces
    private void initializeBoard() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
```

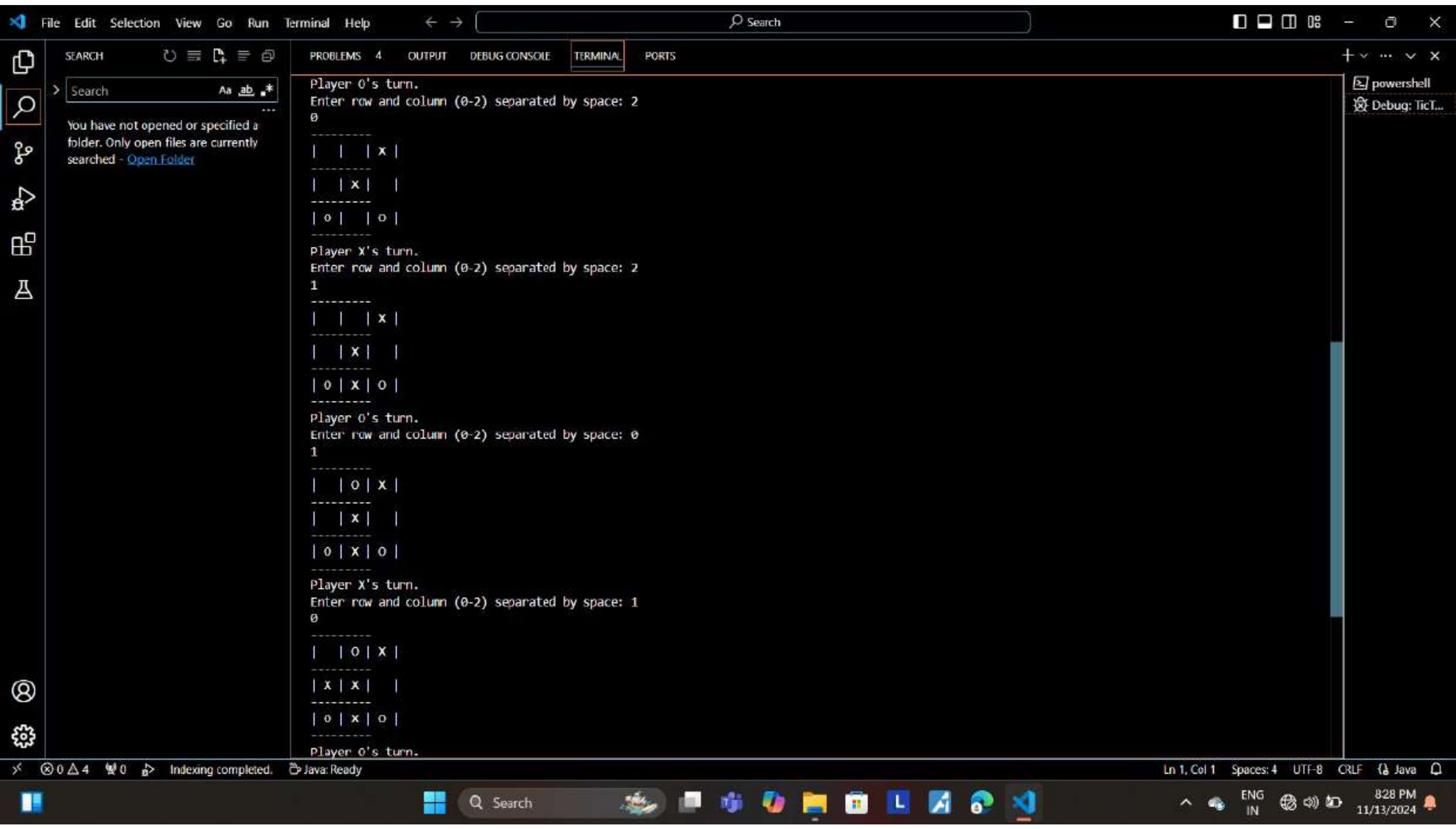
```
            board[i][j] = ' ';
        }
    }
}

// Display the board to the players
public void displayBoard() {
    System.out.println("---------");
    for (int i = 0; i < 3; i++) {
        System.out.print("| ");
        for (int j = 0; j < 3; j++) {
            System.out.print(board[i][j] + " | ");
        }
        System.out.println("\n---------");
    }
}

// Check if there is a winner
public boolean checkWinner() {
        // Check rows, columns, and diagonals
        for (int i = 0; i < 3; i++) {
            if (board[i][0] == currentPlayer && board[i][1] ==
    currentPlayer && board[i][2] == currentPlayer)
                return true;
            if (board[0][i] == currentPlayer && board[1][i] ==
    currentPlayer && board[2][i] == currentPlayer)
                return true;
        }

        // Check diagonals
        if (board[0][0] == currentPlayer && board[1][1] ==
    currentPlayer && board[2][2] == currentPlayer)
            return true;
        if (board[0][2] == currentPlayer && board[1][1] ==
    currentPlayer && board[2][0] == currentPlayer)
            return true;

        return false;
      }
```

```
// Check if the board is full (no more moves possible)
public boolean isBoardFull() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false;
            }
        }
    }
    return true;
}

// Switch to the other player
public void switchPlayer() {
    currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
}

// Make a move at the specified position
```

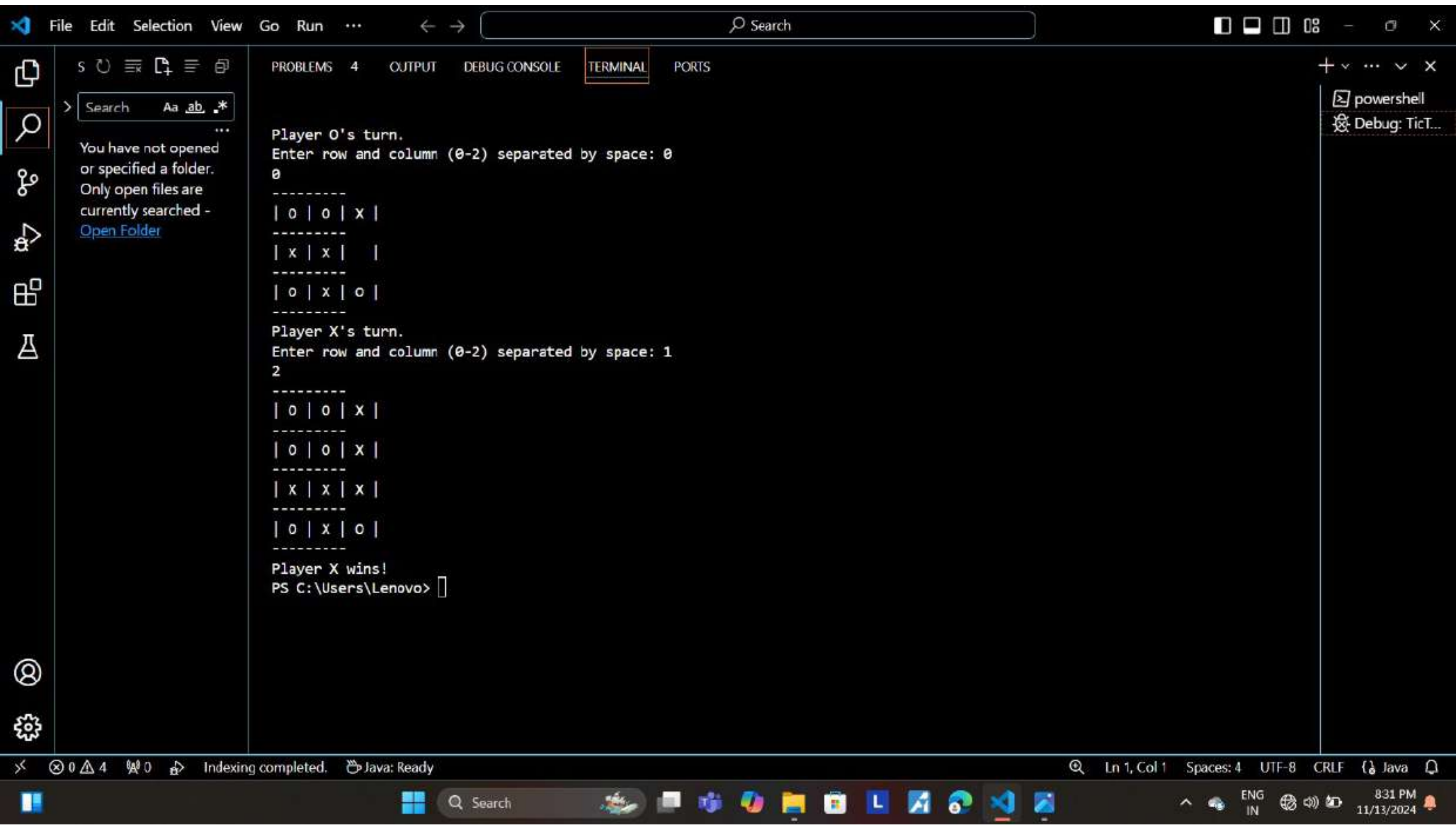
```
    public boolean makeMove(int row, int col) {
        if (row < 0 || row >= 3 || col < 0 || col >= 3 ||
board[row][col] != ' ') {
            return false;  // Invalid move
        }
        board[row][col] = currentPlayer;
        return true;
    }

    // Start the game
    public void playGame() {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            displayBoard();
            System.out.println("Player " + currentPlayer + "'s
turn.");
            System.out.print("Enter row and column (0-2)
separated by space: ");
            int row = scanner.nextInt();
            int col = scanner.nextInt();

            // Make the move if valid
            if (!makeMove(row, col)) {
                System.out.println("Invalid move. Try again.");
                continue;
            }

            // Check for winner
            if (checkWinner()) {
                displayBoard();
                System.out.println("Player " + currentPlayer + "
wins!");
                break;
            }

            // Check for draw
            if (isBoardFull()) {
                displayBoard();
                System.out.println("The game is a draw!");
                break;
            }

            // Switch player
            switchPlayer();
        }
        scanner.close();
    }

    public static void main(String[] args) {
        TicTacToeGame game = new TicTacToeGame();
        game.playGame();
    }
}
```



```
PS C:\Users\Lenovo>  & 'C:\Program Files\Java\jdk-22\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:58674' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Lenovo\AppData\Local\Temp\vscodesws_b7080\jdt_ws\jdt.ls-java-project\bin' 'TicTacToeGame'
---------
|   |   |   |
---------
|   |   |   |
---------
|   |   |   |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 1
1
---------
|   |   |   |
---------
|   | X |   |
---------
|   |   |   |
---------
Player O's turn.
Enter row and column (0-2) separated by space: 2
2
---------
|   |   |   |
---------
|   | X |   |
---------
|   |   | O |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 0
2
---------
|   |   | X |
---------
|   | X |   |
---------
|   |   | O |
---------
Player O's turn.
```



```
Player O's turn.
Enter row and column (0-2) separated by space: 2
0
---------
|   |   | X |
---------
|   | X |   |
---------
| O |   | O |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 2
1
---------
|   |   | X |
---------
|   | X |   |
---------
| O | X | O |
---------
Player O's turn.
Enter row and column (0-2) separated by space: 0
1
---------
|   | O | X |
---------
|   | X |   |
---------
| O | X | O |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 1
0
---------
|   | O | X |
---------
| X | X |   |
---------
| O | X | O |
---------
Player O's turn.
```



```
Player O's turn.
Enter row and column (0-2) separated by space: 0
0
---------
| O | O | X |
---------
| X | X |   |
---------
| O | X | O |
---------
Player X's turn.
Enter row and column (0-2) separated by space: 1
2
---------
| O | O | X |
---------
| O | O | X |
---------
| X | X | X |
---------
| O | X | O |
---------
Player X wins!
PS C:\Users\Lenovo>
```

5.Explanation of code

1. Board Representation:

- The Tic-Tac-Toe board is represented by a 2D character array:

```
private static char[][] board = new char[3][3];
```

Each cell in the board is initialized with a space ' ', which means it's empty. As the game progresses, the cells will be filled with either 'X' or 'O' depending on the current player.

2. Initialization of the Board:

- The initializeBoard method is used to set each element of the board to a space ' ', which represents an empty spot:

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

3. Printing the Board:

- The printBoard method displays the current state of the board after every move:

```
public static void printBoard() {
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         System.out.print(board[i][j]);
         if (j < 2) {
                System.out.print(" | ");
            }
        }
        System.out.println();
        if (i < 2) {
            System.out.println("---------");
        }
    }
}
```

- It prints the characters in the array, separating them with pipes | to create a grid-like format, and prints dashes --------- to separate rows.

4. Checking for a Winner:

- The checkWin method checks for a win condition:
    - It checks each row, column, and both diagonals to see if all three cells are the same and contain the current player's symbol:

```
public static boolean checkWin() {
    // Check rows, columns, and diagonals for a winning condition
    for (int i = 0; i < 3; i++) {
        if (board[i][0] == currentPlayer && board[i][1] == currentPlayer && board[i][2]
== currentPlayer) {
            return true;
        }
        if (board[0][i] == currentPlayer && board[1][i] == currentPlayer && board[2][i]
== currentPlayer) {
            return true;
        }
    }
    if (board[0][0] == currentPlayer && board[1][1] == currentPlayer && board[2][2]
== currentPlayer) {
        return true;
    }
    if (board[0][2] == currentPlayer && board[1][1] == currentPlayer && board[2][0]
== currentPlayer) {
        return true;
    }
    return false;
}
```

5. Checking for a Draw (Full Board):

- The isBoardFull method checks if all cells in the board are filled:

```
public static boolean isBoardFull() {
for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false; // There is still an empty space
            }
        }
    }
    return true; // No empty spaces left
}
```

6. Game Loop:

- The game loop is contained in the playGame method. It continually prompts the current player for their move, checks for a win or draw after each move, and switches the player:

```
public static void playGame() {
    Scanner scanner = new Scanner(System.in);

    while (true) {
        // Print the current board
        printBoard();

        // Ask for the current player's move
        System.out.println("Player " + currentPlayer + ", enter your move (row and
column [1-3]):");
        int row = scanner.nextInt() - 1;
        int col = scanner.nextInt() - 1;

        if (row < 0 || row >= 3 || col < 0 || col >= 3 || board[row][col] != ' ') {
            System.out.println("Invalid move. Try again.");
            continue;
        }

        board[row][col] = currentPlayer; // Make the move

        if (checkWin()) {
            printBoard();
            System.out.println("Player " + currentPlayer + " wins!");
            break;
        }

        if (isBoardFull()) {
            printBoard();
            System.out.println("It's a draw!");
            break;
        }

        currentPlayer = (currentPlayer == 'X') ? 'O' : 'X'; // Switch player
    }
}
```

7. Main Method:

- The main method starts the game by initializing the board and calling the playGame method:

CONCLUSIONS

The Tic-Tac-Toe game in Java serves as a simple yet effective example of utilizing basic programming concepts such as loops, conditionals, arrays, and user input handling. The game features a two-player mode where players take turns placing their respective marks ('X' and 'O') on a 3x3 grid until one player wins, or the board is filled without a winner, resulting in a draw.