ACKNOWLEDGEMENT

We are greatly indebted to our seminar guided by Ms.Nitu.L.Pariyal for her able guidance throughout this work. It 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.A.M.Rajurkar. (Head of Computer Science & Engineering, MGM's College of Engineering, Nanded).

We are heartily thankful to Dr. Lathkar G. S. (Director, MGM's College of Engineering, Nanded) for providing facility during progress of Mini project; also for her kindly help, guidance and inspiration.

Last but not least we are also thankful to all those who help directly or in directly to develop this seminar and complete it successfully.

With Deep Reverence,

Shelke Shweta

[SY-CSE-A]

ABSTRACT

This project implements the classic Tic-Tac-Toe game in Java, designed for two players who take turns to place their respective marks ('X' or 'O') on a 3x3 grid. The game aims to achieve three consecutive marks either horizontally, vertically, or diagonally. The Java application provides an interactive console-based interface, where players can input their moves by specifying row and column positions. The program performs checks after every move to determine if a player has won, if the game is a draw, or if the game continues. The game also includes input validation to ensure that moves are valid and that players cannot overwrite existing marks.

The Java-based Tic-Tac-Toe game is designed with the following features:

- Board Representation: The game board is represented using a 2D array (char[][]), where each element holds the current state of the grid (either 'X', 'O', or an empty space).

- Game Flow: Players take turns to make moves, which are accepted via input coordinates (row and column). After each move, the game checks for a win or draw condition.

- Turn Management: The game alternates between players, allowing one to play as 'X' and the other as 'O'. The game continues until there is a winner or a draw.

- User Interface: The game uses a simple console interface for input and output, displaying the current board after each move and prompting players for their next action.

Shweta bhagawan shelke [174]

SY-A[CSE]

CONTENT

Introduction

Problem Statement

Create a simple, interactive game of Tic Tac Toe for two players. The game should allow players to make moves in a grid format and determine the outcome based on standard game rules.

Purpose of the Project

Develop a functional, user-friendly game to demonstrate core Java programming principles, including object-oriented programming, data structures, and graphical user interfaces (if a GUI is included).

Scope of the Project

The project covers game mechanics, win/tie conditions, and an intuitive user interface. It does not include AI or multiplayer networking features.

Project Requirements

Hardware Requirements

Processor: Minimum 1 GHz

RAM: 2GB or higher

Storage: 50MB of free space

Software Requirements

Java Development Kit (JDK): Version 8 or higher

Integrated Development Environment (IDE): IntelliJ IDEA, Eclipse, or NetBeans

Libraries: Standard Java libraries (no external libraries are needed)

System Requirements

Operating System: Windows, macOS, or Linux

Java Runtime Environment (JRE): Java 8 or higher installed

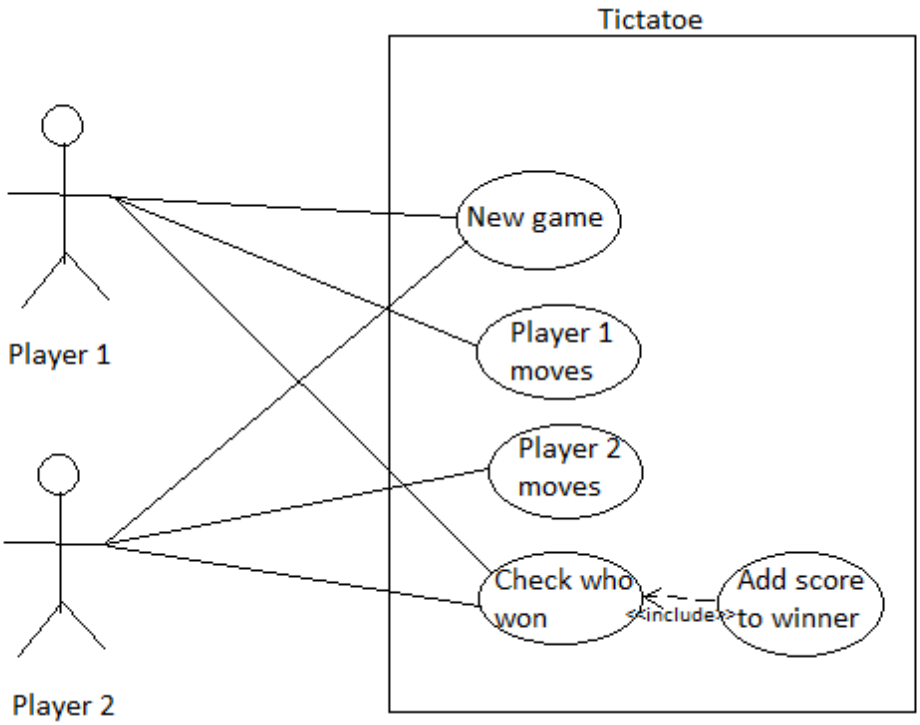
System Design

Architecture

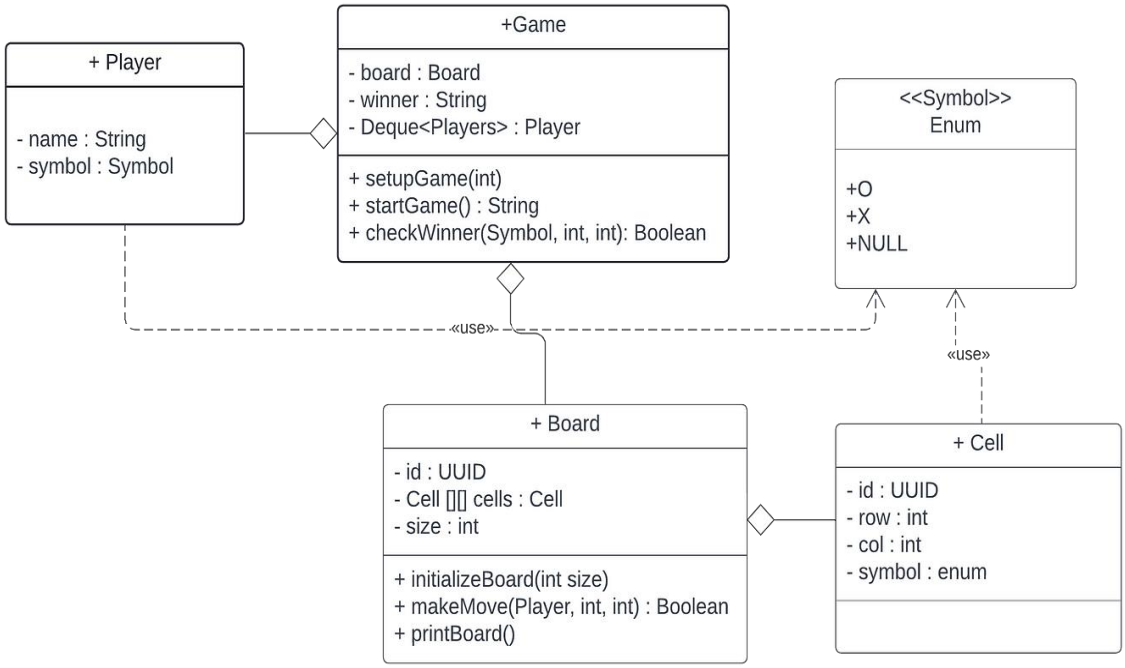
The project follows a modular design:

1. Main Game Class: Controls the flow of the game, switching turns, checking for win conditions, and managing the game state.
2. Board Class: Handles the 3x3 game grid, printing the current board state, and checking if all cells are filled.
3. Player Class: Represents a player with a symbol ('X' or 'O') and the logic for making a move.
4. InputValidator Class: Ensures that the user's input is valid (e.g., within bounds and in an empty space).

Class case Diagram



UML Diagram



Game Class: Main class for game logic (methods: startGame(), checkWin(), resetGame()).

Board Class: Manages the game board (methods: printBoard(), updateBoard(), isBoardFull()).

Player Class: Represents the player (methods: getMove()).

InputValidator Class: Validates player inputs (methods: validateMove()).

Implementation

Overview of Classes and Methods:

Game: Manages the game flow, player turns, and checks for win/tie conditions.

Board: Represents the Tic Tac Toe grid and handles board updates.

Player: Manages player information, including symbols (X or O) and moves.

Core Functionalities:

Game initialization, turn-taking, move validation, and win/tie detection.(Optional) GUI version with buttons representing grid cells.

Challenges and Solutions:

Discuss handling input validation, tie conditions, and possible null values for unoccupied cells.Sample Code Snippets (optional): Include critical code for methods like checkWin() and playMove().

Source Code

```
import java.util.Scanner;

public class TicTacToe {
    private static char[][] board = {{'1', '2', '3'}, {'4', '5', '6'}, {'7', '8', '9'}};
    private static char currentPlayer = 'X';

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int move;
        boolean gameWon = false;

        while (true) {
            printBoard();
            System.out.println("Player " + currentPlayer + ", enter your move (1-9): ");
            move = scanner.nextInt();

            if (!isValidMove(move)) {
                System.out.println("This move is not valid. Try again.");
                continue;
            }

            makeMove(move);
            gameWon = checkWin();

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

```
            }

            if (isBoardFull()) {
                printBoard();
                System.out.println("The game is a draw!");
                break;
            }
currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
        }

        scanner.close();
    }

    private static void printBoard() {
        System.out.println(" " + board[0][0] + " | " + board[0][1] + " | " + board[0][2]);
        System.out.println("---|---|---");
        System.out.println(" " + board[1][0] + " | " + board[1][1] + " | " + board[1][2]);
        System.out.println("---|---|---");
        System.out.println(" " + board[2][0] + " | " + board[2][1] + " | " + board[2][2]);
    }

    private static boolean isValidMove(int move) {
        if (move < 1 || move > 9) {
            return false;
        }

        int row = (move - 1) / 3;
        int col = (move - 1) % 3;
        return board[row][col] != 'X' && board[row][col] != 'O';
    }
```

```
    private static void makeMove(int move) {
        int row = (move - 1) / 3;
        int col = (move - 1) % 3;
        board[row][col] = currentPlayer;
    }

    private static boolean checkWin() {
        // 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;
            }
        }
        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;
    }

    private static boolean isBoardFull() {
        for (int i = 0; i < 3; i++) {
```

```
            for (int j = 0; j < 3; j++) {
                if (board[i][j] != 'X' && board[i][j] != 'O') {
                    return false;
                }
            }
        }
        return true;
    }
}
```

- **Explanation**

Key Components of the Program:

- **Game Board:**

The board is a 3x3 grid represented as a 2D character array (board[3][3]).

Initially, the board is filled with numbers ('1' to '9'), which correspond to the positions where players can make a move.

- **Game Loop:**

The game runs inside an infinite while (true) loop where players take turns making moves until the game ends (either by a win or a draw).

The loop performs the following actions in each iteration:

1. Print the current state of the board.

2. Prompt the current player to make a move.

3. Validate the move.

4. Make the move on the board if valid.

5. Check if the current player has won.

6. Check if the board is full (indicating a draw).

7. Switch to the next player ('X' or 'O').

Player Move:

The player enters a number between 1 and 9 corresponding to an empty spot on the board.

The isValidMove() method checks if the entered move is within the valid range (1-9) and if the spot has not already been occupied by either player ('X' or 'O').

- **Making the Move:**

When a valid move is entered, the program calculates the row and column in the board array based on the move number (1-9), then updates the board with the current player's symbol ('X' or 'O').

- **Win Condition:**

The checkWin() method checks if the current player has won. It checks:

Rows: If all three cells in any row contain the current player's symbol.

Columns: If all three cells in any column contain the current player's symbol.

Diagonals: If all three cells in either diagonal contain the current player's symbol.

If any of these conditions are met, the player wins.

- **Switching Players:**

After each valid move, the current player is alternated. If the current player is 'X', it changes to 'O', and vice versa.

- **Game End:**

The game will end in one of two ways:

Win: If the current player wins, a message is displayed, and the game ends.

Draw: If the board is full and no player has won, a message is displayed indicating the game is a draw.

Output:

1 | 2 | 3

---|---|---

4 | 5 | 6

---|---|---

7 | 8 | 9

Player X, enter your move (1-9):

1

X | 2 | 3

---|---|---

4 | 5 | 6

---|---|---

7 | 8 | 9

Player O, enter your move (1-9):

5

X | 2 | 3

---|---|---

4 | O | 6

---|---|---

7 | 8 | 9

Player X, enter your move (1-9):

2

X | X | 3

---|---|---

4 | O | 6

---|---|---

7 | 8 | 9

Player O, enter your move (1-9):

6

X | X | 3

---|---|---

4 | O | O

---|---|---

7 | 8 | 9

Player X, enter your move (1-9):

3

X | X | X

---|---|---

4 | O | O

---|---|---

7 | 8 | 9

Player X wins!

Testing and Validation

Test Plan:

Unit Testing: Testing individual methods like updateBoard(), checkWin(), and isBoardFull().

Integration Testing: Testing interactions between the Game, Board, and Player classes.

User Acceptance Testing: Ensuring the game works as intended from a user perspective.

Test Cases:

Test Case 1: Player X wins by completing a horizontal row.

Input: Player X chooses (0,0), (0,1), (0,2).

Expected Output: "Player X wins!"

Test Case 2: Game results in a draw (all cells filled, no winner).

Input: Players alternate turns, and the board is filled without any winner.

Expected Output: "It's a draw!"

Bug Tracking:

Minor bug in input validation (handling negative indices). Fixed by adding additional checks for input range.

Results

Screenshots of output

Output:

1 | 2 | 3

---|---|---

4 | 5 | 6

---|---|---

7 | 8 | 9

Player X, enter your move (1-9):

1

X | 2 | 3

---|---|---

4 | 5 | 6

---|---|---

7 | 8 | 9

Player O, enter your move (1-9):

5

X | 2 | 3

---|---|---

4 | O | 6

---|---|---

7 | 8 | 9

Player X, enter your move (1-9):

2

X | X | 3

---|---|---

4 | O | 6

---|---|---

7 | 8 | 9

Player O, enter your move (1-9):

6

X | X | 3

---|---|---

4 | O | O

---|---|---

7 | 8 | 9

Player X, enter your move (1-9):

3

X | X | X

---|---|---

4 | O | O

---|---|---

7 | 8 | 9

Player X wins!

Discussion

Advantages:

Simple, interactive game with minimal setup.

Provides a good starting point for learning Java and game logic.

Easy to extend (adding AI, GUI, or network play).

Limitations:

Console-based; no graphical interface.

Only supports two players; no single-player mode.

Future Enhancements:

Implement an AI opponent to play against the user.

Build a Graphical User Interface (GUI) using JavaFX or swing

Conclusion

The Tic Tac Toe game was successfully implemented using Java, demonstrating key programming concepts like loops, arrays, and object-oriented design. It offers a basic yet fully functional game that is easy to play and understand. Through this project, I gained valuable experience in Java development, problem-solving, and game logic.

References

1.Java libraries

2.frameworks