Report on Tic Tac Toe Game Development

AM.EN.U4CSE21463(S.YOUGESH KUMAR)

1. Introduction

Tic Tac Toe, also referred to as Noughts and Crosses, is a simple yet strategic two-player game that has stood the test of time. Its easy-to-learn rules and compact 3x3 grid make it a universally recognized and loved game.

This project brings the classic game to mobile devices through a user-friendly Android application. Developed in **Java** using **Android Studio**, the application offers an intuitive interface, multiple gameplay modes, and a seamless user experience, even in offline mode.

This report details the development process, challenges encountered, and the features implemented. It also includes the rationale behind technical decisions, snippets of significant code, and areas for potential enhancement.

---

2. Objective

The primary objectives of the Tic Tac Toe application are:

1. **Recreation of the Classic Game**: Develop a fully functional mobile version of Tic Tac Toe.
2. **Engagement**: Provide a fun and interactive way to pass time.
3. **User Accessibility**: Ensure the game is accessible offline.
4. **Skill Development**: Promote logical thinking and strategic gameplay.
5. **Technical Excellence**: Implement robust programming practices to make the app stable and scalable.

---

3. Features

Core Gameplay Features

1. **Single-Player Mode (AI)**:
    - The AI opponent simulates a human player.
    - Ensures the AI balances difficulty without becoming unbeatable.
2. **Two-Player Mode**:
    - Allows two players to play on the same device.
    - Players take turns marking their respective symbols on the board.
3. **Game Statistics**:
    - Tracks and displays the number of wins, losses, and draws for single-player mode.
    - Stores the statistics persistently using Android's SharedPreferences.
4. **Sound and Vibration Settings**:

- Toggle sound effects and vibrations for a personalized user experience.

UI and Design

- Clean, minimalistic design with a focus on usability.
- Buttons and text are dynamically scaled for different device sizes.
- Contrasting colors for clear visibility of symbols and game status.

---

4. Tools and Technologies

1. **Android Studio**: The primary IDE for development, offering a robust platform for building Android applications.
2. **Java**: Used for application logic and game functionality.
3. **XML**: Employed for designing responsive and attractive layouts.
4. **Android SDK**: Utilized for accessing platform-specific features like sound, vibration, and persistent storage.
5. **Testing Environment**: Android Virtual Device (AVD) and physical devices were used to test compatibility and performance.

---

5. Development Process

5.1 Planning

Before starting development, key aspects of the application were outlined:

- **Functional Requirements**: Single-player AI, multiplayer mode, and customizable settings.
- **Non-Functional Requirements**: Responsiveness, offline access, and minimal resource usage.

5.2 Game Logic Implementation

The core game mechanics were implemented using Java, following these steps:

1. **Board Representation**: A 2D character array (char[][]) was used to represent the 3x3 grid.
2. **Win and Draw Conditions**:
    - Checked rows, columns, and diagonals for three consecutive marks.
    - Identified draw situations when all cells were filled without a winner.

5.3 AI Implementation

The AI operates based on the following logic:

1. Searches for available cells in the grid.
2. Places its mark in the first free cell found (basic implementation).
3. Future scalability to include difficulty levels by implementing algorithms like Minimax.

5.4 UI Design

The UI was designed in XML with a focus on:

- **ConstraintLayout**: Ensures responsive layouts across devices.

- **Interactive Elements**: Buttons for each grid cell and settings options.
- **Dynamic Updates**: Real-time updates to the board based on user or AI input.

6. Code Implementation

6.1 Game Initialization

The following code snippet demonstrates the initialization of the game board:

```
// Initialize an empty game board
private char[][] board = new char[3][3];
private void initializeBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = ' '; // Empty cells
        }
    }
}
```

6.2 Checking for Win and Draw

This snippet determines whether a player has won or the game is a draw:

```
// Check if a player has won
private boolean isWin(char player) {
    for (int i = 0; i < 3; i++) {
        if ((board[i][0] == player && board[i][1] == player && board[i][2] == player) ||
            (board[0][i] == player && board[1][i] == player && board[2][i] == player)) {
            return true;
        }
    }
    return (board[0][0] == player && board[1][1] == player && board[2][2] == player) ||
           (board[0][2] == player && board[1][1] == player && board[2][0] == player);
}
```

```
// Check if the game is a draw
private boolean isDraw() {
    for (char[] row : board) {
        for (char cell : row) {
            if (cell == ' ') return false; // Empty cell found
        }
    }
    return true; // No empty cells
}
```

6.3 AI Move

Basic AI that places its mark in the first available position:

```
private void makeAIMove() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                board[i][j] = 'O'; // AI uses 'O'
                return;
            }
        }
    }
}
```

6.4 Updating the UI

Updating the game board visually after each move:

```
 private void updateUI(int row, int col) {
     buttons[row][col].setText(String.valueOf(board[row][col]));
     buttons[row][col].setEnabled(false);
 }
```

6.5 Persistent Storage

Game statistics are saved and retrieved using SharedPreferences:

```
// Save statistics
private void saveStatistics() {
    SharedPreferences preferences = getSharedPreferences("GameStats", MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();
    editor.putInt("wins", wins);
    editor.putInt("losses", losses);
    editor.putInt("draws", draws);
    editor.apply();
}

// Load statistics
private void loadStatistics() {
    SharedPreferences preferences = getSharedPreferences("GameStats", MODE_PRIVATE);
    wins = preferences.getInt("wins", 0);
    losses = preferences.getInt("losses", 0);
    draws = preferences.getInt("draws", 0);
}
```

7. Testing

Test Cases

1. **Single-Player Mode**: Tested AI responses for different moves and scenarios.
2. **Multiplayer Mode**: Validated turn alternation and win conditions.
3. **Draw Conditions**: Ensured the game correctly identified draw situations.
4. **UI Responsiveness**: Verified compatibility across multiple devices and screen sizes.

8. Challenges and Solutions

8.1 AI Implementation

- **Challenge**: Developing an AI that provides a challenging yet fair opponent.
- **Solution**: Implemented a basic AI and planned for upgrades using Minimax.

8.2 Responsive Design

- **Challenge**: Ensuring the UI adapts to various screen sizes.
- **Solution**: Used ConstraintLayout and scalable dimensions.

8.3 Bug Testing

- **Challenge**: Handling edge cases like repeated moves or simultaneous inputs.
- **Solution**: Added checks to ensure game logic integrity.

9. Future Enhancements

1. **Advanced AI**: Introduce difficulty levels using algorithms like Minimax.
2. **Online Multiplayer**: Enable real-time gameplay with friends.
3. **Leaderboards**: Track top players' scores globally.
4. **Customization**: Add themes and color options for personalization.

10. Conclusion

The Tic Tac Toe application achieves its objective of creating an engaging, accessible, and intuitive game. It highlights the importance of design simplicity and logical consistency in mobile app development. Future upgrades will further enhance user experience and expand its features.