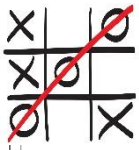

Data Structures

Course Project: Tic-Tac-Toe

Project Update

Athul Chandran A K
ASI23CA021

Athul M B
ASI23CA022

Atul Biju
ASI23CA023

Avinash V Bhaskaran
ASI23CA024

Ayisha Shana Perumballi
ASI23CA025

Project Overview

The Tic Tac Toe game is designed to simulate the classic two-player game using C and arrays. It involves a 3x3 grid where players take turns marking the cells with either an 'X' or an 'O'. The game determines the winner when one player manages to align their marks either horizontally, vertically, or diagonally. If all cells are filled without a winning combination, the game results in a draw. This version uses a simple console-based interface without a GUI and focuses on efficient array manipulation.

Key Features of the System:

- **Game Setup**: A 3x3 array is used to represent the Tic Tac Toe grid.
- **Player Input**: Players alternate turns to input their desired positions.
- **Move Validation**: The game checks if a cell is already occupied before allowing a player to make a move.
- **Winner Detection**: The system checks for winning conditions after every move (row, column, diagonal).
- **Draw Condition**: If all the cells are filled and there is no winner, the game declares a draw.
- **Replay Option**: After the game ends, players are given the option to play again.

Methodology

I. **Game Setup**
   - A 3x3 two-dimensional array is initialized at the start of the program. Each cell in the array is filled with numbers '1' to '9' to represent the empty cells available for player moves.
   - Player 1 is assigned 'X', and Player 2 is assigned 'O'.

II. **Main Game Loop**
   - The game loop runs until a win condition or a draw condition is met.
   - On each player's turn:
     - The current state of the grid is displayed using ASCII art.
     - The player is prompted to choose a position (1-9) to place their mark ('X' or 'O').
     - The program checks if the chosen position is valid (i.e., within the bounds of 1-9 and not already occupied).

III. **Move Validation**
   - After the player enters their move, the program checks whether the chosen cell is already occupied. If it is, the player is asked to re-enter a valid move.
   - If the move is valid, the array is updated with the player's mark at the chosen position, and the game proceeds to the next turn.

IV. **Winner Detection Algorithm**
   - After every move, the program checks if any of the following conditions are satisfied:
     - **Row-wise win**: All three elements in any row are the same ('X' or 'O').
     - **Column-wise win**: All three elements in any column are the same.
     - **Diagonal win**: The three elements in the main diagonal or the opposite diagonal are the same.
   - If a win condition is detected, the game announces the winner and exits the loop.

V. **Draw Condition**
   - If all cells are filled and no-win condition is satisfied, the game declares a draw. This is achieved by counting the number of occupied cells in the array. If all are filled but no winner is found, the game ends in a draw.

VI. **Game Restart Option**
   - After a game concludes (either by a win or a draw), the players are prompted with the option to restart the game. If they choose to play again, the array is reset to its initial state (with cells labelled 1-9), and the game starts anew.

Data Structures Used

- **2D Array**: A 3x3 two-dimensional array is used to represent the Tic Tac Toe grid.

Algorithms Used

- **Input Validation**: Ensures that the player's input corresponds to an empty cell within the grid boundaries.
- **Winning Algorithm**: Checks each row, column, and diagonal after every move to determine if a player has won.
- **Draw Check**: Verifies if all the cells are filled without any winning combinations.

Time Complexity

- **Best Case**: The best-case scenario occurs when a player wins in the minimum number of moves (3), resulting in a time complexity of **O(1)**.
- **Average Case**: The average case typically involves between 5 to 7 moves, maintaining a complexity of **O(n)**, where n is the number of moves played.
- **Worst Case**: The worst-case scenario arises when the game reaches a draw after all 9 moves, also resulting in a complexity of **O(1)** due to the fixed number of moves.

Conclusion

In summary, the Tic Tac Toe game operates efficiently, with both best and worst-case scenarios exhibiting constant time complexity. The limited number of moves ensures that the game's performance remains optimal throughout its execution.

Use Case Diagram: