TICTACTOE JAVA

alexandrubrabete@yahoo.com

TEHNICAL UNIVERSITY OF CLUJ- NAPOCA, SOFTWARE ENGINEERING DEPARTMENT

TicTacToe Java Game Documentation

Project Overview

This project is a **TicTacToe game** implemented in **Java**. It allows users to play TicTacToe against either another human player or an AI opponent. The AI opponent can operate in different difficulty levels, and players can choose the board size before the game begins. The game offers a clean, user-friendly GUI built using Java Swing components, which facilitates easy interaction.

Key Features:

- **Multiplayer Mode:** Play against another human player.
- **Single-Player Mode:** Play against an AI with different difficulty levels.
- **Customizable Board Size:** The game allows you to select different board sizes (3x3, 4x4, etc.).
- **AI Difficulty Levels:** The AI opponent's difficulty can be set to easy, medium, or hard.
- **Animation of Winning Lines:** When a player wins, a line is drawn across the winning cells for visual feedback.
- **Game History and Saving State:** Optionally, save game states to allow players to resume their game later.
- **Optimized AI with Minimax Algorithm** (for medium to hard difficulties).

System Requirements

Software Requirements:

- **Java SDK**: JDK 8 or higher is required to compile and run the application.
- **IDE**: IntelliJ IDEA, Eclipse, or any Java IDE that supports Swing development.

Hardware Requirements:

- **Operating System**: Windows, macOS, or Linux (any system capable of running Java applications).
- **Processor**: Intel i3 or equivalent.
- **Memory**: 4GB RAM minimum.

Core Components

1. Game Board

The game board is implemented as a grid, and the grid can be customized to any size (3x3, 4x4, etc.). The user clicks on empty cells to place their mark (X or O). The game ends when a player wins or the board is filled (resulting in a draw).

Methods:

- `createGameBoard()`: Creates a grid of buttons to represent the TicTacToe board.
- `updateBoard()`: Updates the board when a player moves or when the AI moves.
- `checkWinner()`: Checks if a player has won by evaluating all possible winning lines.

2. Player vs. Player Mode

In this mode, two human players take turns to place their marks on the board. The game automatically alternates between Player 1 and Player 2.

Methods:

- `playerMove()`: Called when a player clicks a button to place their mark.
- `isDraw()`: Determines if the game ends in a draw.

3. AI Opponent

The AI plays against a human player. The AI opponent is powered by a decision-making algorithm (e.g., **Minimax Algorithm**) to evaluate moves and select the optimal one.

Difficulty Levels:

- **Easy**: Random move selection.
- **Medium**: AI blocks winning moves and tries to win when possible.
- **Hard**: AI uses the **Minimax algorithm** with alpha-beta pruning for optimal play.

Methods:

- `chooseMove()`: Selects the best move for the AI depending on the difficulty.
- `minimax()`: Implements the Minimax algorithm to compute the best possible move for the AI.
- `minimaxScore()`: Evaluates a move by simulating the outcome of placing a mark in each cell.

4. Game State

The game state can be saved and loaded. This allows players to pause the game and resume later. The state is stored in a file and includes the current board configuration, turn, and winner (if any).

Methods:

- `saveGame()`: Saves the current game state to a file.
- `loadGame()`: Loads a saved game state.

5. Animations

The game includes animations for winning lines, where a line is drawn through the winning cells, making the victory visually obvious.

Methods:

- `drawWinningLine()`: Draws a line across the winning cells.
- `animateVictory()`: Animates the winning move to provide a better visual experience.

Class Descriptions

Main Class (`TicTacToe.java`)

This is the entry point of the application. It initializes the game, sets up the user interface, and starts the game loop.

Key Responsibilities:

- **Initialize the GUI**: Sets up the TicTacToe board and relevant buttons for user interaction.
- **Start Game Logic**: Starts a new game and manages the state transitions (e.g., player move, AI move, game over).

AI Logic (`AIModule.java`)

This class handles the AI decision-making process. Depending on the selected difficulty level, it computes the best possible move using either a random strategy or the **Minimax Algorithm**.

Key Responsibilities:

- **`chooseMove()`**: Calls the appropriate AI method to decide the next move.

- `minimax()`: Computes the best possible move for the AI.
- `minimaxScore()`: Evaluates potential moves and their outcomes for the AI.

Game Board (`GameBoard.java`)

This class encapsulates the game board and is responsible for rendering the board and keeping track of game states. It updates the board as players make moves.

Key Responsibilities:

- `createGameBoard()`: Creates the visual grid of buttons that represent the board.
- `updateBoard()`: Updates the state of the buttons after each move.
- `checkWinner()`: Checks for a winner by analyzing rows, columns, and diagonals.

Utility Methods (`GameUtils.java`)

This utility class contains helper methods to manage game state transitions and perform validations such as checking for a draw or winner.

Key Responsibilities:

- `isDraw()`: Determines if the game has ended in a draw.
- `isValidMove()`: Checks if the chosen move is valid (i.e., the spot is not already taken).

Algorithm Details

Minimax Algorithm (for AI)

The **Minimax Algorithm** is used for determining the optimal move for the AI in competitive two-player games like TicTacToe. The algorithm works by evaluating all possible future moves and selecting the one that maximizes the AI's chances of winning.

- **Base Case**: If the board is in a win or lose state, return the score (+1 for AI win, -1 for opponent win, 0 for draw).
- **Recursive Case**: Recursively explore all possible moves, alternating between the AI's turn (maximize score) and the opponent's turn (minimize score).
- **Pruning**: Use alpha-beta pruning to optimize the algorithm by eliminating branches of the search tree that don't need to be explored.

Example minimax code:

private static int minimaxScore(JButton[][] buttons, int boardSize, String aiSymbol, String opponentSymbol, boolean isMaximizing) {

```
    if (checkWinner(buttons, boardSize, aiSymbol)) {
        return 1; // AI wins
    }
    if (checkWinner(buttons, boardSize, opponentSymbol)) {
        return -1; // Opponent wins
    }
    if (isDraw(buttons, boardSize)) {
        return 0; // Draw
    }

    int bestScore = isMaximizing ? Integer.MIN_VALUE : Integer.MAX_VALUE;

    for (int i = 0; i < boardSize; i++) {
        for (int j = 0; j < boardSize; j++) {
            if (buttons[i][j].getText().equals("")) {
                buttons[i][j].setText(isMaximizing ? aiSymbol : opponentSymbol);
                int score = minimaxScore(buttons, boardSize, aiSymbol, opponentSymbol,
!isMaximizing);
                buttons[i][j].setText(""); // Undo the move
                bestScore = isMaximizing ? Math.max(bestScore, score) : Math.min(bestScore, score);
            }
        }
    }
    return bestScore;
}
```

User Interface Design

Game Window:

- The main window displays the game board grid of buttons. Each button represents a cell on the board and is clickable.
- The window includes buttons to start a new game, load a saved game, and select AI difficulty.

Animations:

- When a player wins, the winning line is animated to visually indicate the victory.
- The animation includes a smooth transition as the line is drawn over the winning cells.

Conclusion

This TicTacToe game provides a fun and interactive experience with varying difficulty levels, a clean UI, and smooth gameplay. It allows you to either challenge yourself against an AI or play with friends. The Minimax algorithm enhances the AI opponent, making it a more challenging opponent as the difficulty increases.