**Tic Tac Toe**

**Introduction**

Tic-Tac-Toe is a classic two-player game where players take turns marking ‘X’ or ‘O’ on a 3x3 grid. The goal is to form a line of three marks either horizontally, vertically, or diagonally. In this project, we develop an AI-powered Tic-Tac-Toe game that plays against a human opponent. The AI uses the **Minimax algorithm with Alpha-Beta Pruning** to make optimal moves, ensuring that it never loses. This project demonstrates key concepts in **game theory, artificial intelligence, and search algorithms**.

**Module Description**

1. **User Interface (UI) Module:**
   * Developed using **HTML and CSS** to create a simple and interactive game board.
   * Displays the **game grid**, player turn status, and a reset button.
2. **Game Logic Module (JavaScript):**
   * Handles **player moves** and updates the board.
   * Detects **win conditions** and declares a winner.
   * Checks for **a tie when no empty cells remain**.
3. **AI Decision-Making Module:**
   * Implements the **Minimax algorithm with Alpha-Beta Pruning**.
   * Evaluates possible moves and selects the **best move** to maximize AI's chances of winning.
   * Ensures the AI is **unbeatable**.
4. **Event Handling Module:**
   * Detects **user clicks** on the board and registers moves.
   * Disables further moves once the game is over.

**Flow Diagram**

**Game Flow:**

**+---------------------+**

| Start Game |

**+---------------------+**

**|**

**v**

**+---------------------+**

**| Player Makes Move |**

**+---------------------+**

**|**

**v**

**+---------------------+**

**| Check for Winner |**

**+---------------------+**

**| |**

**| Yes | No**

**| v**

**| +----------------+**

**| | AI Makes Move |**

**| +----------------+**

**| |**

**| v**

**| +---------------------+**

**| | Check for Winner |**

**| +---------------------+**

**| |**

**| Yes | No**

**| v**

**| +---------------------+**

**| | Continue Playing |**

**| +---------------------+**

**| |**

**| No v**

**| +----------------+**

**| | Declare Result |**

**| +----------------+**

**| |**

**v v**

**+---------------------+**

**| Reset Game |**

**+---------------------+**

**Enhancements and Technologies Used in Tic-Tac-Toe AI**

**Technologies Used**

1. **HTML (HyperText Markup Language)**
   * Structures the game board and interface elements.
   * Provides a simple and interactive UI for users.
2. **CSS (Cascading Style Sheets)**
   * Enhances the visual appearance of the game.
   * Uses **grid layout** for structuring the board.
   * Provides styling for the game cells, status messages, and buttons.
3. **JavaScript (JS)**
   * Implements the **game logic** (handling user moves, AI moves, and checking for a winner).
   * Uses the **Minimax algorithm with Alpha-Beta Pruning** to make the AI unbeatable.
   * Manages **event listeners** for user interactions.
4. **AI Algorithm (Minimax with Alpha-Beta Pruning)**
   * A **decision-making algorithm** that evaluates all possible moves.
   * Ensures the AI **never loses** by making the optimal move.
   * **Alpha-Beta Pruning** reduces unnecessary computations, improving efficiency.

**Enhancements & Future Improvements**

**1. Difficulty Levels**

* Add **Easy, Medium, and Hard** modes.
* **Easy Mode:** AI makes random moves.
* **Medium Mode:** AI makes strategic but not perfect moves.
* **Hard Mode:** Uses the **Minimax algorithm** (current implementation).

**2. Multiplayer Mode**

* Allow **two players** to play against each other instead of AI.
* Use **WebSockets** for **real-time online multiplayer.**

**3. Improved UI/UX**

* Use **CSS animations** for smooth transitions.
* Highlight winning moves when a player wins.
* Add **sound effects** for moves and game results.

**4. Mobile-Friendly Design**

* Use **CSS media queries** to optimize the game for mobile devices.
* Ensure touch-screen support for a better experience.

**5. Voice-Activated Gameplay**

* Implement **Speech Recognition API** to allow players to make moves using voice commands.

**6. Machine Learning-based AI**

* Instead of **rule-based Minimax**, use **Reinforcement Learning** or **Neural Networks** to train an AI that learns from past games.

**7. Leaderboard and Score Tracking**

* Store game history using **Local Storage** or a **database**.
* Track the number of wins, losses, and ties.

**8. Integration with PWA (Progressive Web App)**

* Convert the game into a **PWA**, so users can play offline.
* Enable installation on mobile devices like a native app.

**Coding:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Tic-Tac-Toe AI</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            text-align: center;

        }

        .board {

            grid-template-columns: repeat(3, 100px);

            grid-gap: 5px;

            width: 320px;

            margin: 20px auto;

        }

        .cell {

            width: 100px;

            height: 100px;

            display: flex;

            align-items: center;

            justify-content: center;

            font-size: 2em;

            background-color: #f0f0f0;

            cursor: pointer;

            border: 1px solid #000;

        }

        .cell.taken {

            cursor: not-allowed;

        }

        .status {

            font-size: 1.5em;

            margin: 10px;

        }

        .reset-btn {

            padding: 10px 20px;

            font-size: 1em;

            cursor: pointer;

            margin-top: 10px;

        }

    </style>

</head>

<body>

    <h1>Tic-Tac-Toe</h1>

    <div class="status">Your Turn!</div>

    <div class="board" id="board"></div>

    <button class="reset-btn" onclick="resetGame()">Reset Game</button>

    <script>

        const boardElement = document.getElementById("board");

        const statusElement = document.querySelector(".status");

        let board = Array(9).fill(null);

        const human = "X", ai = "O";

        function createBoard() {

            boardElement.innerHTML = "";

            board.forEach((value, index) => {

                const cell = document.createElement("div");

                cell.classList.add("cell");

                if (value) cell.classList.add("taken");

                cell.textContent = value || "";

                cell.addEventListener("click", () => makeMove(index));

                boardElement.appendChild(cell);

            });

        }

        function makeMove(index) {

            if (board[index] || checkWinner(board)) return;

            board[index] = human;

            if (!checkWinner(board)) {

                aiMove();

            }

            updateGame();

        }

        function aiMove() {

            let bestScore = -Infinity, bestMove;

            for (let i = 0; i < 9; i++) {

                if (!board[i]) {

                    board[i] = ai;

                    let score = minimax(board, 0, false, -Infinity, Infinity);

                    board[i] = null;

                    if (score > bestScore) {

                        bestScore = score;

                        bestMove = i;

                    }

                }

            }

            if (bestMove !== undefined) board[bestMove] = ai;

        }

        function minimax(board, depth, isMaximizing, alpha, beta) {

            let winner = checkWinner(board);

            if (winner) return winner === human ? -10 + depth : 10 - depth;

            if (!board.includes(null)) return 0;

            if (isMaximizing) {

                let maxEval = -Infinity;

                for (let i = 0; i < 9; i++) {

                    if (!board[i]) {

                        board[i] = ai;

                        let eval = minimax(board, depth + 1, false, alpha, beta);

                        board[i] = null;

                        maxEval = Math.max(maxEval, eval);

                        alpha = Math.max(alpha, eval);

                        if (beta <= alpha) break;

                    }

                }

                return maxEval;

            } else {

                let minEval = Infinity;

                for (let i = 0; i < 9; i++) {

                    if (!board[i]) {

                        board[i] = human;

                        let eval = minimax(board, depth + 1, true, alpha, beta);

                        board[i] = null;

                        minEval = Math.min(minEval, eval);

                        beta = Math.min(beta, eval);

                        if (beta <= alpha) break;

                    }

                }

                return minEval;

            }

        }

        function checkWinner(board) {

            const winPatterns = [

                [0, 1, 2], [3, 4, 5], [6, 7, 8],

                [0, 3, 6], [1, 4, 7], [2, 5, 8],

                [0, 4, 8], [2, 4, 6]

            ];

            for (let pattern of winPatterns) {

                const [a, b, c] = pattern;

                if (board[a] && board[a] === board[b] && board[a] === board[c]) {

                    return board[a];

                }

            }

            return null;

        }

        function updateGame() {

            createBoard();

            let winner = checkWinner(board);

            if (winner) {

                statusElement.textContent = winner === human ? "You Win!" : "AI Wins!";

                document.querySelectorAll(".cell").forEach(cell => cell.classList.add("taken"));

            } else if (!board.includes(null)) {

                statusElement.textContent = "It's a Tie!";

            } else {

                statusElement.textContent = "Your Turn!";

            }

        }

        function resetGame() {

            board = Array(9).fill(null);

            statusElement.textContent = "Your Turn!";

            createBoard();

        }

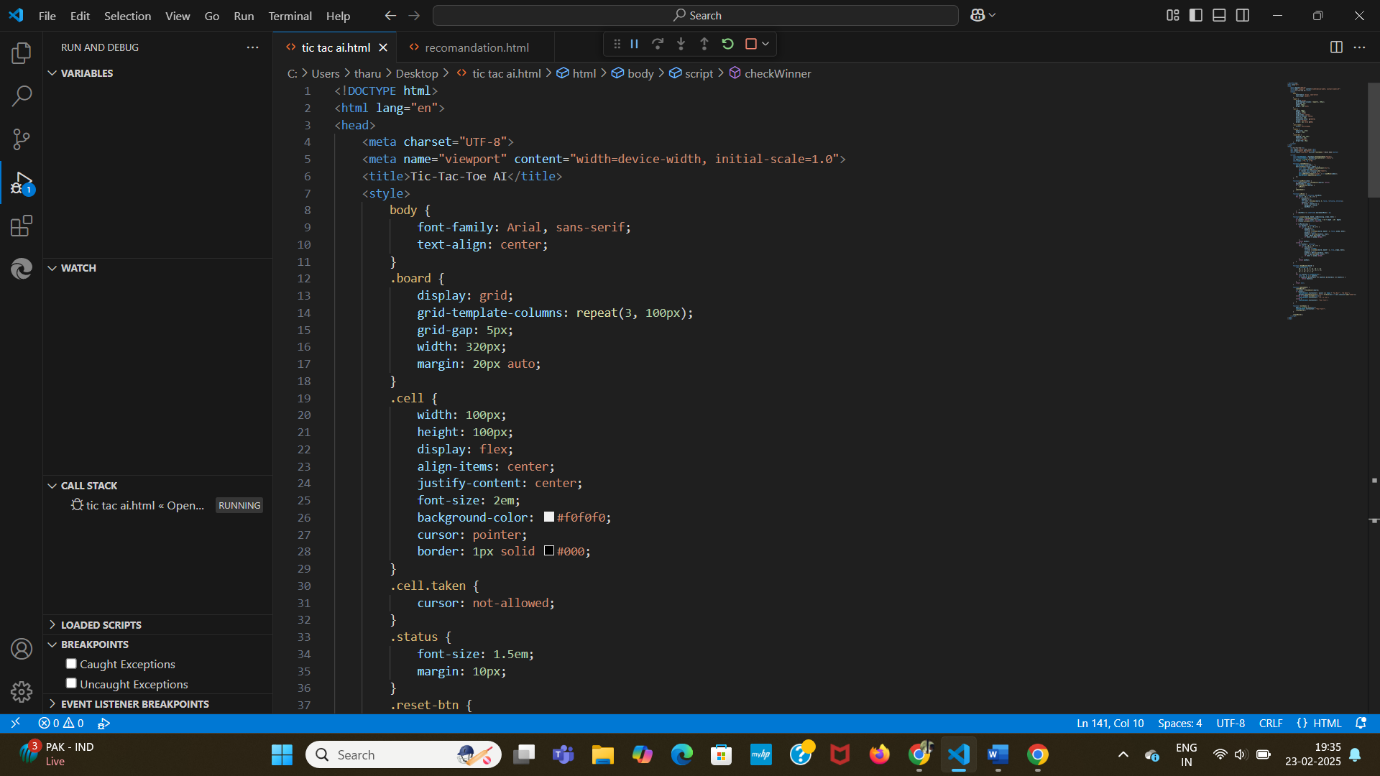
        createBoard();

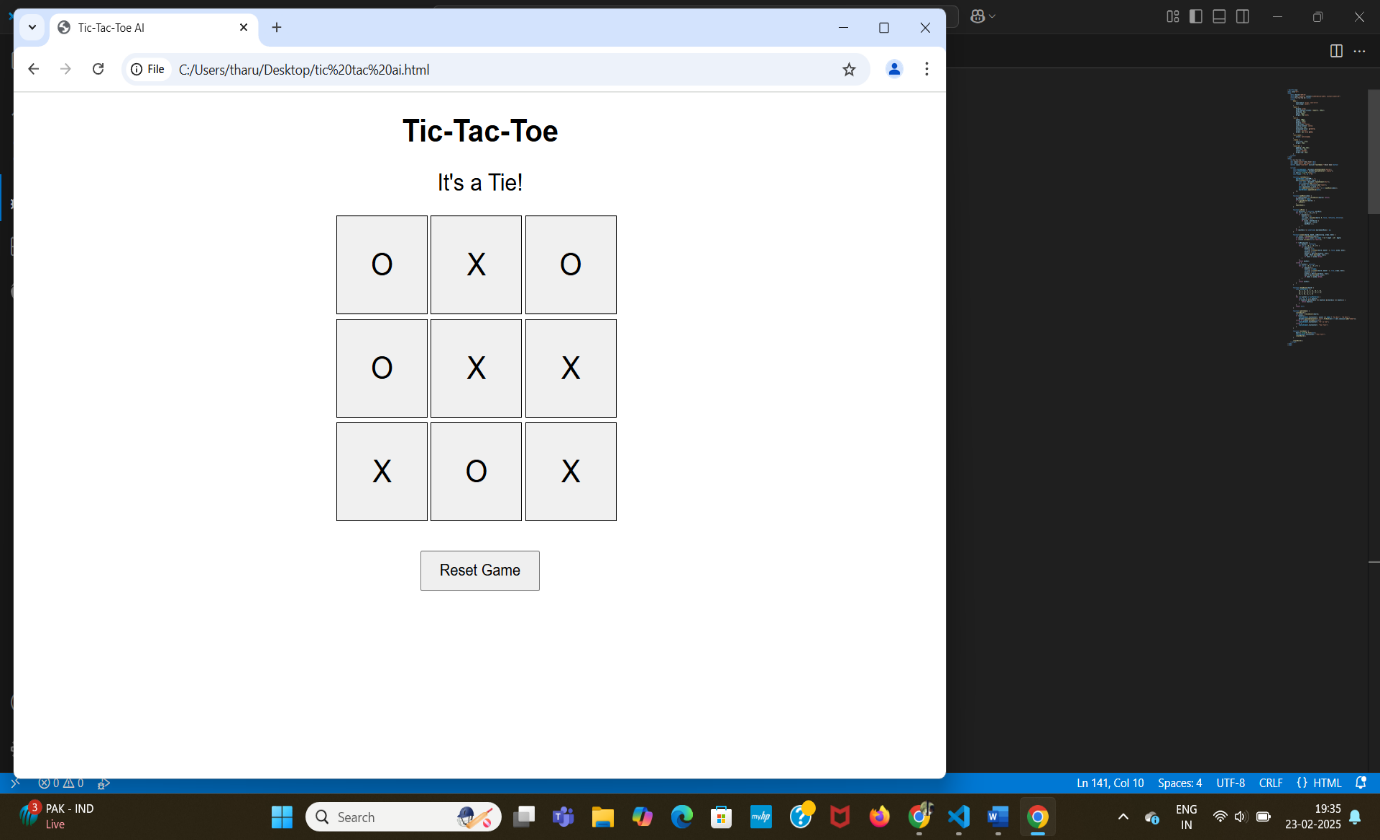
    </script>

</body>

</html>

**Output:**

****



**Conclusion:**

This project successfully implements an unbeatable Tic-Tac-Toe AI using the Minimax algorithm with Alpha-Beta Pruning. It enhances problem-solving skills in AI, game development, and algorithm optimization. Players can test their skills against an AI that makes optimal decisions, ensuring a challenging experience. This project can be further improved by adding difficulty levels, a multiplayer option, or a more visually appealing UI. The Tic-Tac-Toe AI project currently implements an unbeatable AI using Minimax with Alpha-Beta Pruning. Future enhancements such as multiplayer mode, difficulty levels, improved UI, and machine learning-based AI will make it even more engaging and challenging. Using technologies like WebSockets, Speech Recognition, and Reinforcement learning can further improve the user experience and intelligence of the AI**.**