# **Documentation for the "Tic-Tac-Toe" Game Project**

## **Overview**

This project is a simple **Tic-Tac-Toe** game created using **HTML**, **CSS**, and **JavaScript**. The game includes two modes: **playing with a bot** or **playing with a friend**. It also features a scoreboard to track scores, a reset button to restart the game, and dynamic game board rendering based on player actions.

## **File Structure**

index.html

## **HTML Code**

### **Basic Structure**

The HTML structure provides the layout and interactive elements of the Tic-Tac-Toe game.

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Крестики-нолики</title>  
 <style>  
 /\* CSS code goes here \*/  
 </style>  
</head>  
<body>  
 <h1>Крестики-нолики</h1>  
 <div class="mode-selector">  
 Выберите режим:  
 <button class="button" onclick="setMode('bot')">Игра с ботом</button>  
 <button class="button" onclick="setMode('friend')">Игра с другом</button>  
 </div>  
 <div class="scoreboard">  
 Игрок 1 (Yerassyl): <span id="scoreYerassyl">0</span> |  
 Ничья: <span id="scoreDraws">0</span> |  
 Игрок 2 (Daulet/Bot): <span id="scoreDaulet">0</span>  
 </div>  
 <div class="board" id="board"></div>  
 <button class="button" id="reset">Перезапуск</button>  
 <script>  
 /\* JavaScript code goes here \*/  
 </script>  
</body>  
</html>

### **Key HTML Elements**

1. **<h1>**: Displays the title "Крестики-нолики" (Tic-Tac-Toe).
2. **<div class="mode-selector">**: Contains buttons for selecting the game mode (playing with a bot or with a friend).
3. **<div class="scoreboard">**: Displays the current scores for Player 1, Player 2, and the number of draws.
4. **<div class="board" id="board">**: The container where the 3x3 game grid is dynamically rendered.
5. **<button class="button" id="reset">**: The reset button that allows the game to restart.

## **CSS Code**

### **Styling the Game Interface**

The CSS defines the visual appearance of the game, including the game board, cells, buttons, and text.

body {  
 font-family: 'Arial';  
 background-color: rgb(52 152 219 / 60%);  
 text-align: center;  
}  
  
h1 {  
 font-size: 3rem;  
 margin-top: 20px;  
}  
  
.board {  
 display: grid;  
 grid-template-columns: repeat(3, 100px);  
 gap: 10px;  
 justify-content: center;  
 margin: 30px auto;  
 max-width: 340px;  
}  
  
.cell {  
 width: 100px;  
 height: 100px;  
 font-size: 2rem;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 border: 2px solid #3498db;  
 background-color: #ecf0f1;  
 cursor: pointer;  
 transition: all 0.3s ease;  
 border-radius: 10px;  
}  
  
.cell.taken {  
 cursor: not-allowed;  
}  
  
.cell.X {  
 background-color: #003366;  
 color: white;  
}  
  
.cell.O {  
 background-color: #e74c3c;  
 color: white;  
}  
  
.cell:hover {  
 transform: scale(1.1);  
 box-shadow: 0 0 10px rgba(52, 152, 219, 0.6);  
}  
  
.button {  
 background-color: #3498db;  
 color: white;  
 border: none;  
 padding: 10px 20px;  
 font-size: 1.2rem;  
 cursor: pointer;  
 border-radius: 5px;  
 transition: background-color 0.3s ease;  
}  
  
.button:hover {  
 background-color: #2980b9;  
}  
  
.button:active {  
 background-color: #1abc9c;  
}

### **Explanation of Key CSS Styles**

1. **body**: Sets the background color to a semi-transparent blue and centers the text.
2. **h1**: Enlarges the title text.
3. **.board**: Uses CSS Grid to create a 3x3 grid with a maximum width of 340px, centered on the page.
4. **.cell**: Defines the size, background, and interactivity of each individual game cell.
5. **.cell.X and .cell.O**: Applies specific colors for player "X" and "O".
6. **.button**: Styles the buttons with a blue background and smooth hover transitions.

## **JavaScript Code**

### **Key JavaScript Variables**

const board = document.getElementById("board");  
const resetButton = document.getElementById("reset");  
const scoreYerassyl = document.getElementById("scoreYerassyl");  
const scoreDaulet = document.getElementById("scoreDaulet");  
const scoreDraws = document.getElementById("scoreDraws");  
  
let currentPlayer = "X"; // Starting player is X  
let scores = { X: 0, O: 0, Draws: 0 }; // Initial scores  
let gameState = Array(9).fill(null); // Board state (9 cells)  
let mode = "friend"; // Default game mode (friend vs. friend)

### **Functions**

#### **1. setMode(selectedMode)**

Sets the game mode to either "bot" or "friend".

function setMode(selectedMode) {  
 mode = selectedMode;  
 resetGame(); // Reset the game when the mode is changed  
}

#### **2. renderBoard()**

Renders the game board dynamically, creating cells and displaying the current state.

function renderBoard() {  
 board.innerHTML = ""; // Clear previous board  
 gameState.forEach((cell, index) => {  
 const cellElement = document.createElement("div");  
 cellElement.classList.add("cell");  
 if (cell) {  
 cellElement.classList.add("taken");  
 cellElement.classList.add(cell);  
 }  
 cellElement.textContent = cell || "";  
 cellElement.addEventListener("click", () => makeMove(index));  
 board.appendChild(cellElement);  
 });  
}

#### **3. checkWinner()**

Checks for a winner or a draw after each move.

function checkWinner() {  
 const winningCombos = [  
 [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 (const combo of winningCombos) {  
 const [a, b, c] = combo;  
 if (gameState[a] && gameState[a] === gameState[b] && gameState[a] === gameState[c]) {  
 return gameState[a]; // Return winner "X" or "O"  
 }  
 }  
 return gameState.includes(null) ? null : "Draw"; // Return "Draw" if no winner  
}

#### **4. makeMove(index)**

Handles the player's move, updates the game state, and checks if the game is won or drawn.

function makeMove(index) {  
 if (gameState[index] || checkWinner()) return;  
  
 gameState[index] = currentPlayer;  
 const winner = checkWinner();  
  
 if (winner) {  
 if (winner === "Draw") {  
 scores.Draws++;  
 alert("Ничья!");  
 } else {  
 scores[winner]++;  
 alert((winner === "X" ? "Yerassyl" : "Daulet") + " выиграл!");  
 }  
 updateScores();  
 return;  
 }  
  
 currentPlayer = currentPlayer === "X" ? "O" : "X";  
 renderBoard();  
  
 if (mode === "bot" && currentPlayer === "O") {  
 botMove(); // Bot makes a move  
 }  
}

#### **5. botMove()**

Simulates the bot's move by selecting a random empty cell.

function botMove() {  
 const emptyCells = gameState  
 .map((cell, index) => (cell === null ? index : null))  
 .filter(index => index !== null);  
  
 const randomIndex = emptyCells[Math.floor(Math.random() \* emptyCells.length)];  
 makeMove(randomIndex); // Bot makes a random move  
}

#### **6. updateScores()**

Updates the scoreboard based on the current game scores.

function updateScores() {  
 scoreYerassyl.textContent = scores.X;  
 scoreDaulet.textContent = scores.O;  
 scoreDraws.textContent = scores.Draws;  
}

#### **7. resetGame()**

Resets the game state and re-renders the board for a fresh game.

function resetGame() {  
 gameState = Array(9).fill(null);  
 currentPlayer = "X";  
 renderBoard();  
}

### **Game Flow**

1. **Page Load**: The board is rendered with 9 empty cells.
2. **Game Mode**: Players can choose between playing against a bot or with a friend.
3. **Player Actions**: Players click on cells to place their mark ("X" or "O").
4. **Game Status**: After each move, the game checks if there is a winner or a draw.
5. **Bot Play**: If playing against the bot, the bot makes random moves for "O".
6. **Game Reset**: The game can be reset by clicking the reset button.

This concludes the documentation for the Tic-Tac-Toe game. The code examples explain key functions, variables, and the flow of the game.