INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

TAREA #1:

"Juego Sencillo Tic-Tac-Toe"

Integrantes:

Anastasio Salas Juan Carlos

Cabrera Duran Itzel Amayramy

Materia:

Tecnologías de Aplicaciones Web

Horario:

14:00 - 15:00 hrs.

Cd. Madero, Tamaulipas

DEJANDO HUELLA

TAREA NO. 1: Juego Sencillo Tic-Tac-Toe

INSTRUCCIONES:

- **Funcionalidad:** Implementa la interfaz y la lógica de un juego simple.
- **Punto clave:** Fortalecerás tu lógica de programación en JavaScript, la gestión del estado del juego y la interacción de la interfaz de usuario.
- **Lenguajes por utilizar:** HTML 5, CSS3 y Javascript.

Utilice 2 opciones para el juego

- Computadora vs Jugador
- Jugador vs Jugador

DESARROLLO:

Concepto del Proyecto

El proyecto consiste en una implementación web del clásico juego "Tres en Raya" (Tic Tac Toe). Su enfoque principal es ofrecer una experiencia configurable y funcional, permitiendo al usuario personalizar aspectos clave de la partida antes de jugar. A diferencia de versiones básicas, esta aplicación integra un panel de configuración que controla la lógica del juego en tiempo real.

¿Qué hace?

La aplicación permite a los usuarios jugar partidas de Tic Tac Toe en dos modalidades principales:

- **Multijugador (Local):** Dos personas juegan en el mismo dispositivo alternando turnos.
- **Un Jugador (vs AI):** El usuario juega contra una "Inteligencia Artificial" básica (movimientos aleatorios).

Además, el usuario puede configurar:

- **Turno inicial:** Elegir quién hace el primer movimiento (Jugador 1 o Jugador 2).
- **Símbolo:** Elegir con qué ficha jugar ('X' o 'O').

Análisis de Funciones

A continuación, se presenta el desglose de las funciones del archivo tictactoe.js, ordenadas jerárquicamente según su flujo de ejecución (desde la inicialización hasta la lógica de juego).

1. initializeGame()

Propósito: Es la función de arranque ("entry point") lógico del script. Se encarga de preparar el juego para la primera interacción.

Props/Argumentos: Ninguno.

Proceso:

1. Asigna los "Event Listeners" a todas las celdas del tablero (cells) para detectar clicks.
2. Asigna el evento de click al botón de reinicio (resetButton) y al botón de inicio (startButton).
3. Inicializa el texto de estado indicando el turno actual.
4. Establece la variable running en true para permitir el juego.

```
function initializeGame() {
    cells.forEach(cell => cell.addEventListener('click', cellClicked));
    resetButton.addEventListener('click', resetGame);
    startButton.addEventListener('click', applySettings);
    statusText.textContent = `${currentPlayer}'s turn`;
    running = true;
}
```

2. applySettings()

Propósito: Capturar la configuración elegida por el usuario en el panel "Game Settings" y aplicar esos cambios al estado del juego.

Props/Argumentos: Ninguno.

Proceso:

1. Lee los valores actuales de los selectores del DOM (modeSelect, symbolSelect, startSelect).
2. Actualiza las variables globales mode, player1Symbol, player2Symbol y currentPlayer basándose en estas lecturas.
3. Si el modo es "single", asigna qué símbolo corresponde al humano y cuál a la IA.
4. Llama a resetGame() para limpiar el tablero y comenzar con las nuevas reglas.

```
function applySettings() {
    mode = modeSelect.value;
    player1Symbol = symbolSelect.value;
    player2Symbol = player1Symbol === "X" ? "O" : "X";

    const whoStarts = startSelect.value;
    currentPlayer = whoStarts === "player1" ? player1Symbol : player2Symbol;

    if (mode === "single") {
        humanPlayer = player1Symbol;
        aiPlayer = player2Symbol;
    }

    resetGame();
    console.log("Cambios de configuracion detectado.")

}
```

3. resetGame()

Propósito: Reiniciar el estado del tablero y las variables de control para comenzar una nueva partida limpia.

Props/Argumentos: Ninguno.

Proceso:

- Determina quién debe iniciar según la configuración (whoStarts).
- Limpia el array options (el estado interno del tablero) llenándolo de cadenas vacías.
- Actualiza el texto de estado (statusText) para indicar de quién es el turno.
- Limpia visualmente el tablero borrando el texto de todas las celdas HTML.
- Si el modo es "single" y le toca iniciar a la IA, programa la llamada a aiMove() con un pequeño retraso para simular "pensamiento".

```
function resetGame() {
    // Determinar quién inicia basado en la configuración
    const whoStarts = startSelect ? startSelect.value : "player1";
    currentPlayer = whoStarts === "player1" ? player1Symbol : player2Symbol;

    options = ["", "", "", "", "", "", "", "", ""];

    if (mode === "single") {
        statusText.textContent = currentPlayer === humanPlayer ? "Your turn" : "AI's turn";
    } else {
        statusText.textContent = `It's ${currentPlayer}'s turn`;
    }

    cells.forEach(cell => cell.textContent = "");
    running = true;

    // Si es modo single player y la mÃ¡quina inicia, hacer movimiento
    if (mode === "single" && currentPlayer === aiPlayer) {
        statusText.textContent = "AI is thinking...";
        setTimeout(() => {
            aiMove();
        }, 500);
    }
}
```

4. cellClicked()

Propósito: Manejador del evento de click en una celda. Es el disparador de la acción del usuario.

Props/Argumentos: La función recibe implícitamente el contexto this (la celda clickeada).

Proceso:

1. Obtiene el índice de la celda desde el atributo data-cell-index.
2. **Validación:** Verifica si la celda ya está ocupada o si el juego ha terminado (!running). Si es así, no hace nada.
3. **Bloqueo:** Si es turno de la IA en modo "single", ignora los clicks del usuario para evitar jugar fuera de turno.
4. Llama a updateCell() para registrar el movimiento.
5. Llama a checkWinner() para verificar si el movimiento causó una victoria o empate.
6. Si el juego continúa y es modo "single", programa el turno de la IA (aiMove) tras 500ms.

```
function cellClicked() {
    const cellIndex = this.getAttribute('data-cell-index');

    if (options[cellIndex] !== "" || !running) {
        return;
    }
        Ctrl+I for Command, Ctrl+L for Agent
    if (mode === "single" && currentPlayer === aiPlayer) {
        return;
    }

    updateCell(this, cellIndex);
    checkWinner();

    if (mode === "single" && running && currentPlayer === aiPlayer) {
        statusText.textContent = "AI is thinking...";
        setTimeout(() => {
            aiMove();
        }, 500);
    }
}
```

Stylish Tic Tac Toe - Antigravity - tictactoe.js

5. updateCell(cell, index)

Propósito: Actualizar visual y lógicamente una celda específica.

Props/Argumentos:

- **cell:** El elemento DOM (div) de la celda.
- **index:** El índice numérico de la celda en el array.

Proceso:

1. Actualiza el array options en la posición index con el símbolo del jugador actual.
2. Modifica el contenido de texto del elemento cell para mostrar el símbolo.

```
function updateCell(cell, index) {
    options[index] = currentPlayer;
    cell.textContent = currentPlayer;
}
```

6. checkWinner()

Propósito: Verificar el estado del juego después de cada movimiento para determinar si hay un ganador, un empate o si el juego debe continuar.

Props/Argumentos: Ninguno.

Proceso:

1. Itera sobre el array winConditions (que contiene las combinaciones de índices ganadores: filas, columnas, diagonales).
2. Compara si las tres celdas de alguna combinación tienen el mismo símbolo (y no están vacías).
3. **Si hay ganador:** Muestra mensaje de victoria y detiene el juego (running = false).
4. **Si no hay ganador, pero el tablero está lleno:** Muestra mensaje de empate ("Draw").
5. **Si no ocurre nada de lo anterior:** Llama a changePlayer() para ceder el turno al siguiente jugador.

```
function checkWinner() {
    let roundWon = false;
    for(let i = 0; i < winConditions.length; i++) {
        const condition = winConditions[i];
        const cellA = options[condition[0]];
        const cellB = options[condition[1]];
        const cellC = options[condition[2]];

        if (cellA === "" || cellB === "" || cellC === "") {
            continue;
        }
        if (cellA === cellB && cellB === cellC) {
            roundWon = true;
            break;
        }
    }
    if (roundWon) {
        if (mode === "single") {
            statusText.textContent = currentPlayer === humanPlayer ? "You win! 🎉" : "AI wins! 🤖";
        } else {
            statusText.textContent = `${currentPlayer} wins! 🎉`;
        }
        running = false;
    } else if (!options.includes("")) {
        statusText.textContent = `It's a draw! 🤝`;
        running = false;
    } else {
        changePlayer();
    }
```

Tab to Jump Line 139

7. changePlayer()

Propósito: Alternar el turno entre los jugadores.

Props/Argumentos: Ninguno.

Proceso:

- Cambia el valor de currentPlayer al símbolo opuesto.
- Actualiza el texto de la interfaz (statusText) indicando de quién es el nuevo turno (personalizando el mensaje si es turno del usuario o de la IA).

```
function changePlayer() {
    currentPlayer = (currentPlayer === player1Symbol) ? player2Symbol : player1Symbol;

    if (mode === "single") {
        statusText.textContent = currentPlayer === humanPlayer ? "Your turn" : "AI's turn";
    } else {
        statusText.textContent = `It's ${currentPlayer}'s turn`;
    }
}
```

8. aiMove()

Propósito: Ejecutar el movimiento automático de la computadora.

Props/Argumentos: Ninguno.

Proceso:

1. **Filtrado:** Crea una lista de índices disponibles (celdas vacías) en el tablero actual.
2. **Validación:** Si no hay celdas (empate técnico no detectado antes), termina.
3. **Decisión:** Selecciona un índice aleatorio (Math.random()) de la lista de celdas disponibles (IA básica, no estratégica).
4. Selecciona el elemento DOM correspondiente y llama a updateCell().
5. Llama a checkWinner()para finalizar su turno.

```
function aiMove() {
    if (!running) return;

    // Obtener celdas vacías usando filter
    const availableCells = options
        .map((cell, index) => cell === "" ? index : null)
        .filter(index => index !== null);

    if (availableCells.length === 0) return;

    // Seleccionar una celda aleatoria
    const randomIndex = Math.floor(Math.random() * availableCells.length);
    const cellIndex = availableCells[randomIndex];

    // Actualizar la celda
    const cell = document.querySelector(`[data-cell-index="${cellIndex}"]`);
    updateCell(cell, cellIndex);
    checkWinner();
}
```