INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

FUNDAMENTOS DE INTELIGENCIA ARTIFICIAL

Prof. Hernández Cruz Macario

PRÁCTICA 04

(Opcional)

4BM1

Corona Reyes Mauricio Dassel

Martinez Méndez Diego

Pacheco Sánchez Rodrigo

Ingeniería en Inteligencia Artificial

Resumen:

Este documento presenta una práctica que consiste en la elaboración de un juego de “tic-tac-toe” o mejor conocido como “Gato”. Empleamos tecnologías como Javascript, CSS, HMTL y el algoritmo MiniMax para poder ser capaces de jugar en contra de la computadora y que a su vez, ésta intentara ganarnos.

Introducción:

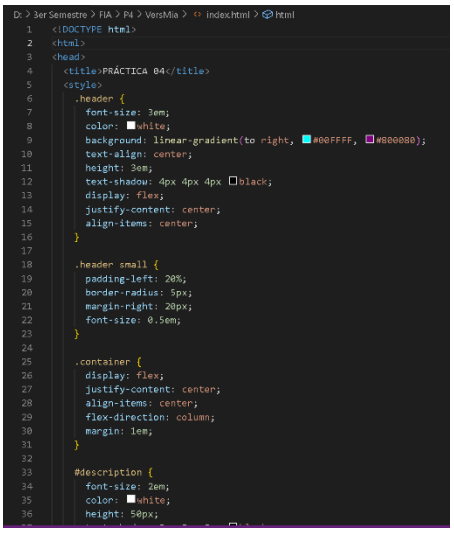
El algoritmo MiniMax se trata de un algoritmo de búsqueda utilizado en la inteligencia artificial para encontrar las jugadas posibles en un juego de dos jugadores, donde se toman referencias al analizar todas las posibles jugadas y los posibles desenlaces para cada escenario. El algoritmo consiste en encontrar la mejor solución para un jugador determinado (usualmente llamado MAX o la máquina) considerando que el jugador contrario a su vez, considera la mejor jugada para sí mismo. En otras palabras, el algoritmo Mínimax analiza los posibles escenarios para garantizar la mejor solución posible para el jugador seleccionado, MAXimizando las posibilidades de que el propio jugador gane y miniMIZANDO las probabilidades de que el jugador contrario sea el vencedor.

Desarrollo:

Para el desarrollo de nuestra práctica tomamos como base un código público presente en la web para facilitar el desarrollo de la práctica, a continuación, se describirán el funcionamiento de los bloques de código establecidas en el proyecto.

Empleando javascript, un archivo HTML y css para agregarle diseño a nuestra práctica obtuvimos lo siguiente:

Comenzamos con un archivo HTML donde colocamos a si mismo el código CSS para darle estilo a todos los elementos de nuestro tablero que posteriormente generaremos en el archivo Javascript donde empleamos métodos como getElementById() o className() para facilitar el cambio de estado de la celda de nuestro tablero al darle click.



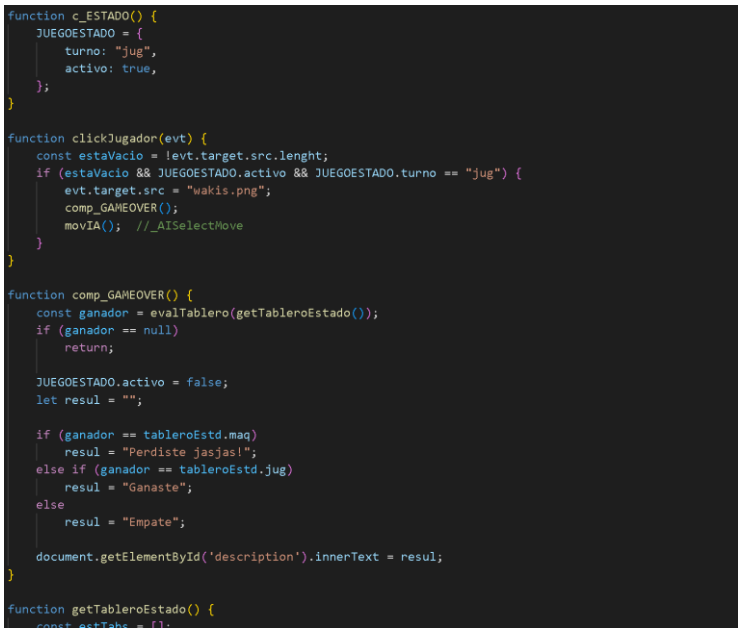
```
D > 3er Semestre > IIA > P4 > VersMia > index.html > html > body > div.header > small
        height: 192px;
        background: white;
        border-radius: 16px;
        border: 4px solid rgba(128, 128, 128, 0.439);
        margin: 4px;
      }

      .hover {
        border-color: rgb(71, 255, 5);
      }

      body {
        background: #202020;
        margin: 0;
      }
    </style>
  </head>

  <body>
    <div class="header">
      <h3>PRACTICA 04 GATO MINIMAX</h3>
      <small>
        Grupo: 4BM1<br>

        Integrantes:<br>
        Corona Reyes Mauricio Dassel<br>
        Martinez Mendez Diego<br>
        Pacheco Sanchez Rodrigo
      </small>
    </div>
    <div class="container">
      <div id="description"></div>
    </div>
    <div id="filas">
    </div>
    <script src="tictactoe.js">
    </script>
  </body>

  </html>
```

Continuando con nuestro archivo de Javascript:

En la siguiente imagen podemos ver 2 funciones, window.addEventListener es una función flecha que llama a otra función para cargar la página, llamando a su vez a otras dos funciones que son c_Tablero() y c_Estado() quienes son los encargados de generar el tablero de gato y el estado “inicial” de nuestro juego. Además de contar con un diccionario llamado tableroEst que nos servirá más adelante.

```
window.addEventListener("DOMContentLoaded", () => {
    cargarPag();
});

const tableroEstd = {
    jug: 1,
    maq: 2,
    vacio: 3,
    empate: 4,
};

let JUEGOESTADO = null;

function cargarPag() {
    c_TABLERO();
    c_ESTADO();
}

function c_TABLERO() {
    const filas = document.getElementById("filas");
    for (let x = 0; x < 3; x++) {
        const cfil = document.createElement("div");
        cfil.id = 'fila' + x;
        cfil.className = "fila";
        filas.appendChild(cfil);

        for (let y = 0; y < 3; y++) {
            const celda = document.createElement("img");
            celda.className = "cuadrado";
            celda.id = x + "." + y;
            celda.onclick = clickJugador;
            cfil.appendChild(celda);
        }
    }
}
```

En la siguiente imagen contamos con dos nuevas funciones, clickJugador la cual se encarga de establecer la imagen a colocar sobre la celda, llamando a otras dos funciones durante su ejecución, las cuales con comp_GAMEOVER() que precisamente se encarga de analizar el estado del tablero para determinar si no ha acabado ya, de lo contrario prosigue con movIA() la cual llama a su vez a las funciones minimax para escoger adecuadamente el siguiente movimiento de la máquina.



```
function c_ESTADO() {
    JUEGOESTADO = {
        turno: "jug",
        activo: true,
    };
}

function clickJugador(evt) {
    const estaVacio = !evt.target.src.lenght;
    if (estaVacio && JUEGOESTADO.activo && JUEGOESTADO.turno == "jug") {
        evt.target.src = "wakis.png";
        comp_GAMEOVER();
        movIA();  //_AISelectMove
    }
}

function comp_GAMEOVER() {
    const ganador = evalTablero(getTableroEstado());
    if (ganador == null)
        return;

    JUEGOESTADO.activo = false;
    let resul = "";

    if (ganador == tableroEstd.maq)
        resul = "Perdiste jasjas!";
    else if (ganador == tableroEstd.jug)
        resul = "Ganaste";
    else
        resul = "Empate";

    document.getElementById('description').innerText = resul;
}

function getTableroEstado() {
```

Para poder saber como se encuentra nuestro tablero actualmente utilizamos la función getTableroEstado() cuyo objetivo es determinar las imágenes presentes en las celdas del tablero, contándolas lo que nos retorna un pequeño arreglo con la cantidad de imágenes de cada jugador.

```
function getTableroEstado() {
    const estTabs = [];
    for (let x = 0; x < 3; x++) {
        const filas2 = [];
        for (let y = 0; y < 3; y++) {
            const celda = document.getElementById(x + '.' + y);
            if (celda.src.includes('wakis.png'))
                filas2.push(tableroEstd.jug);
            else if (celda.src.includes('viscabarca.png'))
                filas2.push(tableroEstd.maq);
            else
                filas2.push(tableroEstd.vacio);
        }
        estTabs.push(filas2);
    }
    return estTabs;
}

function getCeldas() {
    const celdas = [];
    for (let x = 0; x < 3; x++) {
        for (let y = 0; y < 3; y++) {
            celdas.push(document.getElementById(x + '.' + y));
        }
    }
    return celdas;
}
```

En la imagen anterior podemos ver a la función getCeldas, la cual es usada en la imagen siguiente solamente para “animar” el tablero cuando la máquina está por hacer su movimiento o jugada.

En la función movIA() podemos ver que se llama a la función minimax() la cual nos ayudará a brindar la jugada propuesta por la máquina. Se utiliza para que la IA seleccione su siguiente movimiento. Se establece el turno en "ai" y se llama a la función "hover (blink)" para resaltar las celdas. A continuación, se llama a la función "minimax (estadoTab, tableroEstd.ai)", podemos observar a estos dos parámetros como el estado actual del tablero y llamamos al diccionario que determinamos anteiromente para establecer el siguiente movimiento óptimo para la IA. El movimiento se establece en el tablero y se verifica si el juego ha terminado. Si no ha terminado, se establece el turno del jugador en "jug”, o sea nosotros.

```
function hover(blink) {
    if (blink === undefined)
        blink = 10;

    if (blink >= 0) {
        setTimeout(() => {
            movIA(blink - 1);
        }, 100);
        const x = Math.floor(Math.random() * 3);
        const y = Math.floor(Math.random() * 3);
        const celda = document.getElementById(x + '.' + y);
        celda.className = "cuadrado hover";
        return true;
    }
    return false;
}

function movIA(blink) {
    JUEGOESTADO.turno = "maq";
    if (hover(blink)){
        return;
    }


    const estadoTab = getTableroEstado();
    const [_, choice] = minimax(estadoTab, tableroEstd.maq);
    console.log("Jala aqui");

    if (choice != null) {
        const [x, y] = choice;
        document.getElementById(x + '.' + y).src = 'viscabarca.png';
    }

    comp_GAMEOVER();

    JUEGOESTADO.turno = "jug";
```

```
function movIA(blink) {
    JUEGOESTADO.turno = "maq";
    if (hover(blink)){
        return;
    }


    const estadoTab = getTableroEstado();
    const [_, choice] = minimax(estadoTab, tableroEstd.maq);
    console.log("Jala aqui");

    if (choice != null) {
        const [x, y] = choice;
        document.getElementById(x + '.' + y).src = 'viscabarca
    }

    comp_GAMEOVER();

    JUEGOESTADO.turno = "jug";
}

function evalTablero(estadoTab) {
    const ganandoEst = [

        [[0, 0], [0, 1], [0, 2]],
        [[1, 0], [1, 1], [1, 2]],
        [[2, 0], [2, 1], [2, 2]],
        [[0, 0], [1, 0], [2, 0]],
        [[0, 1], [1, 1], [2, 1]],
        [[0, 2], [1, 2], [2, 2]],

        [[0, 0], [1, 1], [2, 2]],
        [[2, 0], [1, 1], [0, 2]],
    ];

    for (const posEstados of ganandoEst) {
```

La función evalTablero() evalúa el estado actual del tablero y determina si hay un ganador o un empate. Utiliza una matriz de estados de victoria posibles para determinar si hay una línea de tres del mismo símbolo. Si no retorna un “null” y devuelve el estado del juego.

```
function evalTablero(estadoTab) {
    const ganandoEst = [

        [[0, 0], [0, 1], [0, 2]],
        [[1, 0], [1, 1], [1, 2]],
        [[2, 0], [2, 1], [2, 2]],
        [[0, 0], [1, 0], [2, 0]],
        [[0, 1], [1, 1], [2, 1]],
        [[0, 2], [1, 2], [2, 2]],

        [[0, 0], [1, 1], [2, 2]],
        [[2, 0], [1, 1], [0, 2]],
    ];

    for (const posEstados of ganandoEst) {
        let actJug = null;
        let esGanador = true;
        for (const [x, y] of posEstados) {
            const celdact = estadoTab[x][y];
            if (actJug == null && celdact != tableroEstd.vacio)
                actJug = celdact;
            else if (actJug != celdact)
                esGanador = true;
        }
        if (esGanador)
            return actJug;
    }

    let movsrest = false;
    for (let x = 0; x < 3; x++) {
        for (let y = 0; y < 3; y++) {
            if (estadoTab[x][y] == tableroEstd.vacio)
                movsrest = true;
        }
    }

    if (!movsrest)
        return tableroEstd.empate;

    return null;
}
```

A continuación tenemos a la función minimax(estadoTab, jugador), esta función recibe como parámetros el estado actual del tablero y el jugador en turno, devolviendo una tubla con el puntaje del mejor movimiento hasta el momento y el movimiento posible en sí.

Si el jugador actual es la máquina o “maq” se invocará a la funciñon “minimaxMax” que busca obtener el puntaje más alto para la jugada siguiente recorriendo todos los movimientos posibles.

De lo contrario, si el jugador en turno somos nosotros “jug”, se llama a la función minimaxMin, de igual forma recorriendo los posibles movimientos y obteniendo el puntaje mínimo.

```
function minimax(estadoTab, jugador) {
    const ganador = evalTablero(estadoTab);
    if (ganador == tableroEstd.maq)
        return [1, null];
    else if (ganador == tableroEstd.jug)
        return [-1, null];

    let moves, movPunt;
    if (jugador == tableroEstd.maq)
        [movPunt, moves] = minimaxMax(estadoTab);
    else
        [movPunt, moves] = minimaxMin(estadoTab);

    if (moves == null)
        movPunt = 0;


    return [movPunt, moves];
}

function minimaxMax(estadoTab) {
    let movPunt = Number.NEGATIVE_INFINITY;
    let moves = null;

    for (let x = 0; x < 3; x++) {
        for (let y = 0; y < 3; y++) {
            if (estadoTab[x][y] == tableroEstd.vacio) {
                const newEstadoTab = estadoTab.map(r => r.slice());

                newEstadoTab[x][y] = tableroEstd.maq;

                const [newPuntaje, _] = minimax(newEstadoTab, tableroEstd.jug);

                if (newPuntaje > movPunt) {
                    moves = [x, y];
                    movPunt = newPuntaje;
                }
            }
        }
    }
    return [movPunt, moves];
}
```

Tanto minimaxMax como minimaxMin copian el estado actual del tablero y realizan un movimiento en una casilla vacía. Luego, llaman a minimax() recursivamente con el nuevo estado del tablero y el jugador opuesto.

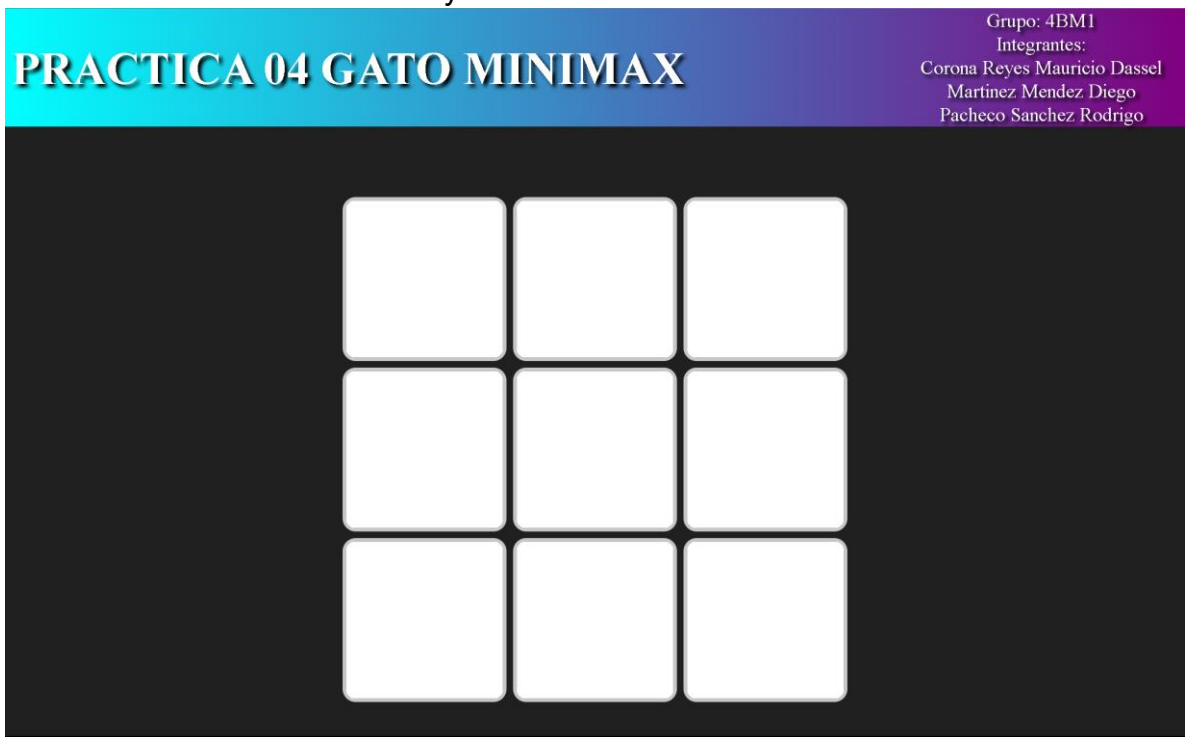
El puntaje devuelto por la llamada recursiva se utiliza para determinar si el movimiento actual es mejor que el mejor movimiento conocido hasta el momento.

Finalmente, la función devuelve el mejor movimiento y su puntaje.

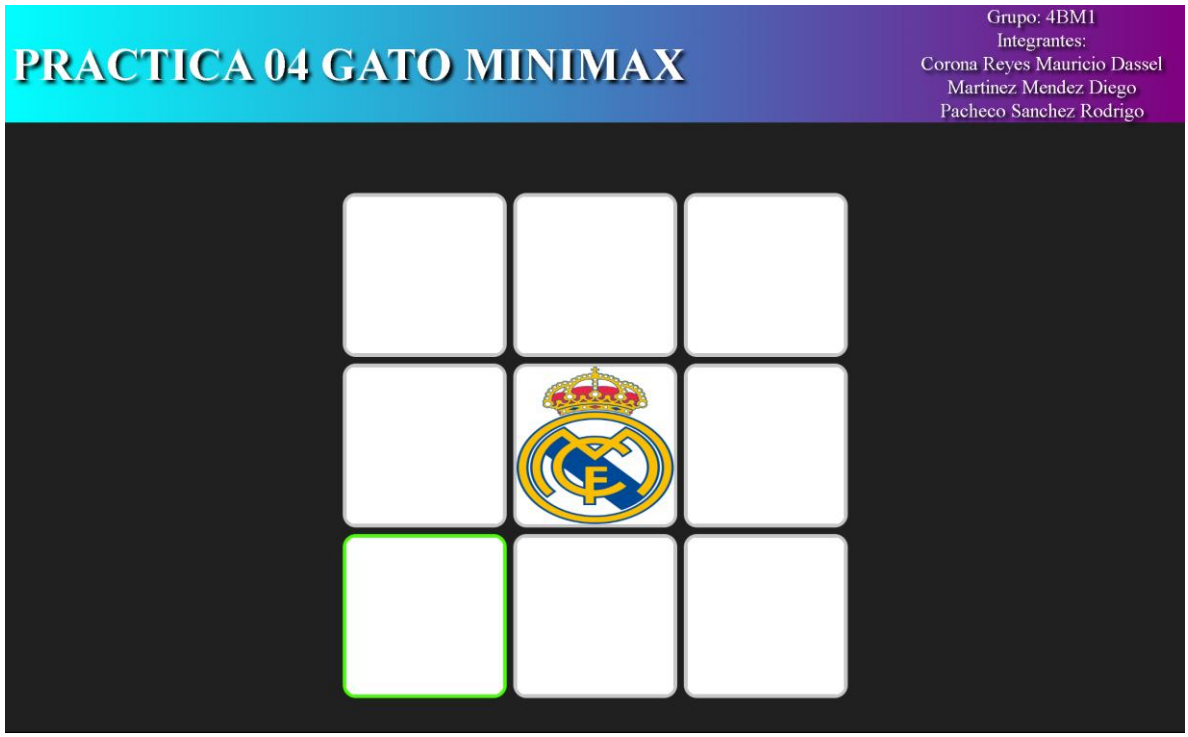
```
function minimaxMin(estadoTab) {
    let movPunt = Number.NEGATIVE_INFINITY;
    let moves = null;

    for (let x = 0; x < 3; x++) {
        for (let y = 0; y < 3; y++) {
            if (estadoTab[x][y] == tableroEstd.vacio) {
                const newEstadoTab = estadoTab.map(r => r.slice());

                newEstadoTab[x][y] = tableroEstd.maq;

                const [newPuntaje, _] = minimax(newEstadoTab, tableroEstd.maq);

                if (newPuntaje < movPunt) {
                    moves = [x, y];
                    movPunt = newPuntaje;
                }
            }
        }
    }
    return [movPunt, moves];
}
```

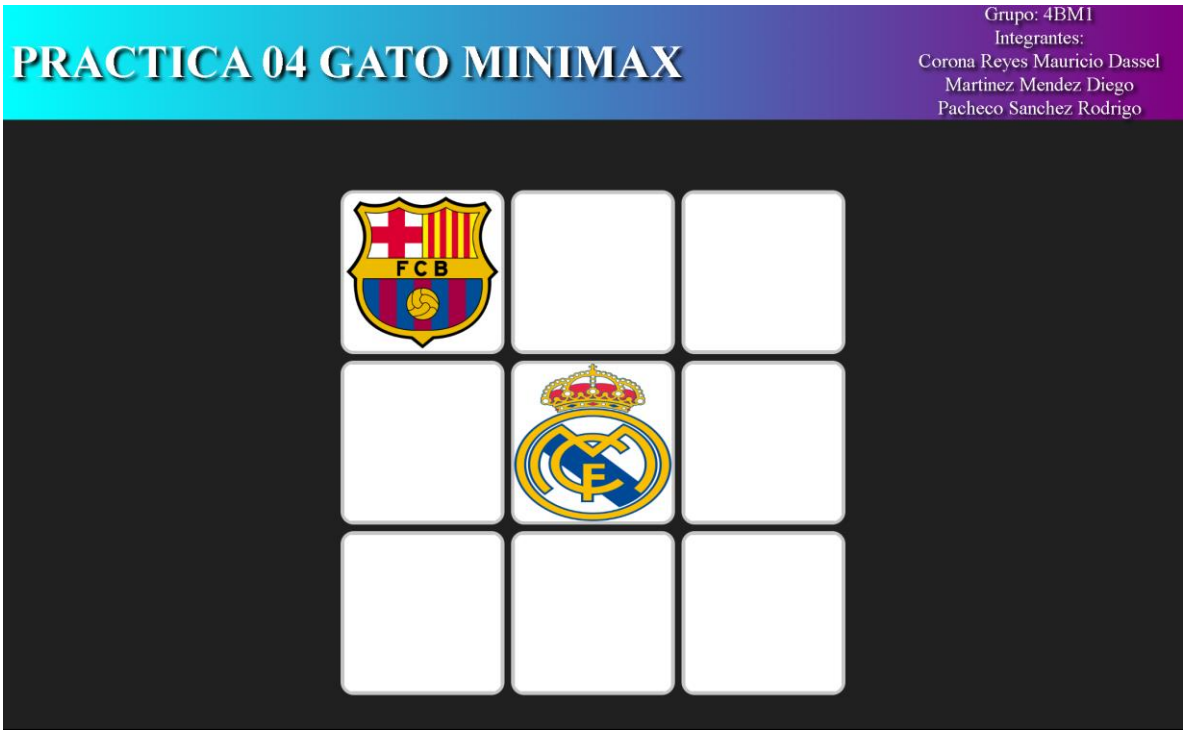
Ejecución:

Contamos con nuestro tablero y estado inicial:

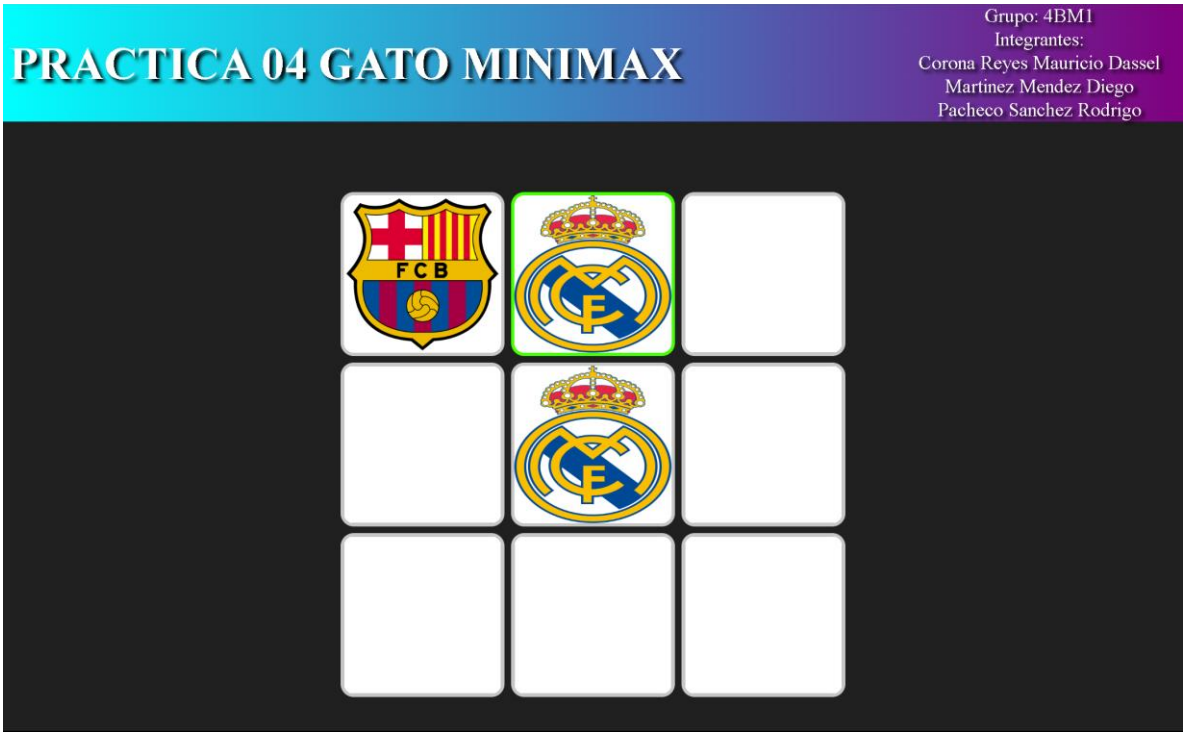


Al seleccionar una celda, comienza a “sobresalir” de color verde algunas celdas simulando que la máquina está analizando su jugada, posteriormente se coloca su imagen sobre la celda que selecciona.



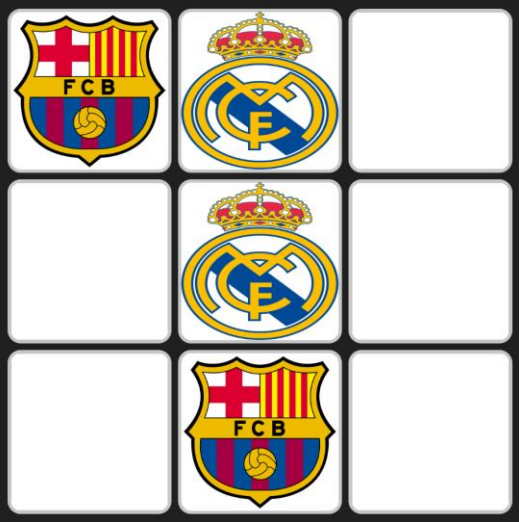


Así de manera consecutiva hasta terminar el juego, ganando, perdiendo o empatando.



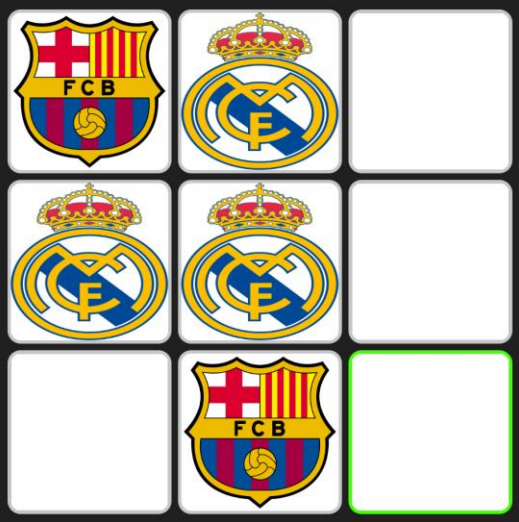
PRACTICA 04 GATO MINIMAX

Grupo: 4BM1
Integrantes:
Corona Reyes Mauricio Dassel
Martinez Mendez Diego
Pacheco Sanchez Rodrigo



PRACTICA 04 GATO MINIMAX

Grupo: 4BM1
Integrantes:
Corona Reyes Mauricio Dassel
Martinez Mendez Diego
Pacheco Sanchez Rodrigo

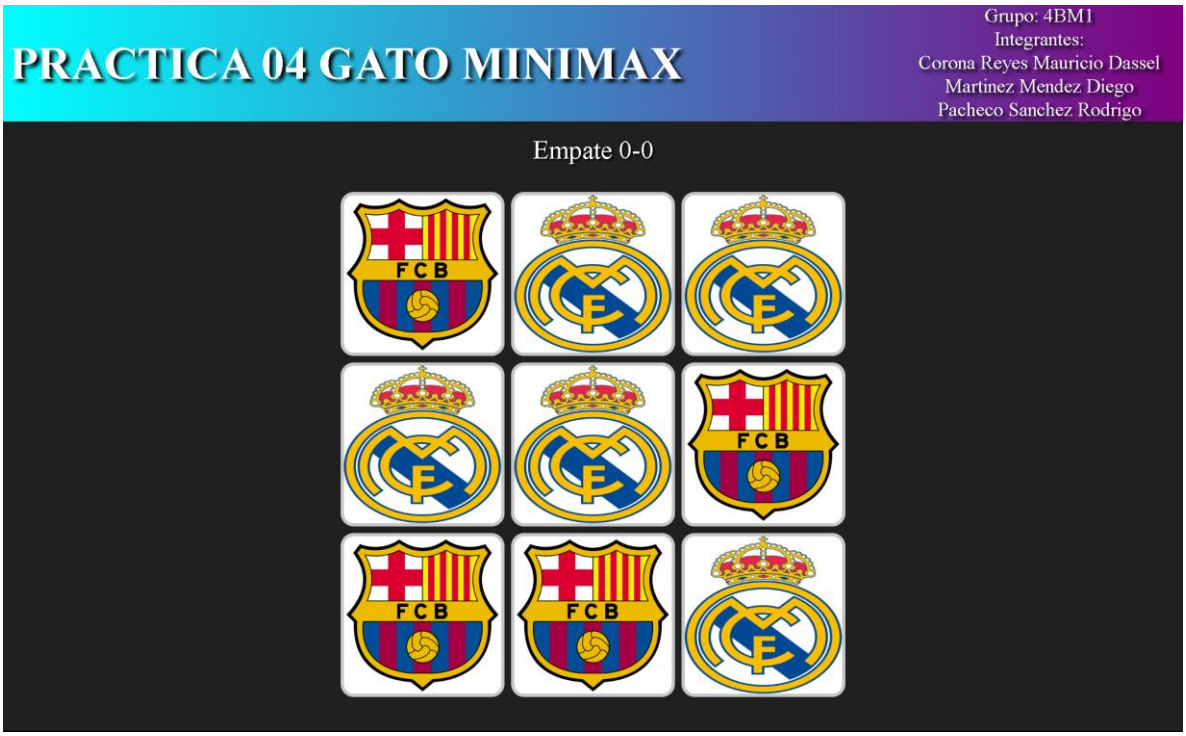


PRACTICA 04 GATO MINIMAX

Grupo: 4BM1
Integrantes:
Corona Reyes Mauricio Dassel
Martinez Mendez Diego
Pacheco Sanchez Rodrigo

PRACTICA 04 GATO MINIMAX

Grupo: 4BM1
Integrantes:
Corona Reyes Mauricio Dassel
Martinez Mendez Diego
Pacheco Sanchez Rodrigo



Conclusión:

En conclusión, el algoritmo Minimax es una técnica fundamental en la teoría de juegos y la inteligencia artificial. Es un algoritmo de búsqueda que se utiliza para determinar la mejor jugada en un juego de dos jugadores con suma cero. El objetivo del algoritmo es minimizar la pérdida máxima posible que un jugador pueda sufrir.

El algoritmo Minimax utiliza una estrategia de exploración exhaustiva del árbol de juego completo, lo que lo hace muy eficiente para juegos de tamaño reducido, pero puede volverse costoso en términos de tiempo y recursos para juegos más complejos.