Cne Amine Bouatay

Alumno DIM XLII

PRÁCTICA CUATRO EN RAYA

1- Propuesta de trabajo: Este trabajo consiste en una práctica de la primera parte del curso de Inteligencia Artificial del curso de Informática Militar, que consiste en desarrollar el juego de 4 en Raya.

2- **Objetivo del trabajo:** El objetivo es implementar el algoritmo minimax a un juego de un humano contra el ordenador que permita la configuración de la dificultad mediante la configuración por adelantado del numero de jugadas que se anualizarán.
El algoritmo minimax conoce el estado del otro jugador, selecciona el mejor movimiento.
La implementación del algoritmo minimax se ha llevado a cabo en el juego cuatro en Raya, que consiste en un tablero de 6x7 (6 filas, 7 columnas), cuyo objetivo es colocar fichas y conseguir alinear 4 del mismo color, en cualquier dirección, según sea el turno de cada jugador.

3- un poco de historia del Cuatro en raya: El Cuatro en raya, juego distribuido desde 1984 por Hasbro, fue creado en 1973 por Ned Strongin y Howard Wexler para Milton Bradley Company, empresa que lo comercializó por primera vez bajo la marca registrada “Connect Four” en 1974.

4- Juego de Cuatro en raya:

a- Descripción del juego :

Cuatro en raya o Cuatro en línea es un juego para dos personas,cada jugador posee 21 fichas identicas de un color diferente. En el original, se utilizan los colores amarillo y rojo. Se juega en un tablero vertical que consiste de 7 columnas y 6 cuadrados en cada una (6x7) Una jugada se puede definir como situar una ficha dentro de un cuadrado del tablero, dentro de turnos. Cada jugador debe conectar 4 fichas de diferentes formas, ya sea horizontal, vertical o diagonal. El primero que logre este resultado gana la partida, si las 42 fichas han sido utilizadas y ningun jugador ha logrado su meta, se considera un empate.

b- Objetivo del juego:

El objetivo del juego es alinear cuatro fichas de un mismo color en horizontal, vertical o diagonal.

c- Jugadores :

Es un juego para 2 jugadores, que se sitúan uno frente al otro.

d- Tablero :

El tablero está formado por una rejilla vertical de 7 columnas (verticales) y 6 filas (horizontales), para un total de 42 casillas. El tablero tiene 42 agujeros redondos a ambos lados del tablero, lo que permite a los jugadores observar la posición de todas las fichas.

e- Fichas :

Cada jugador dispone de 21 fichas de un mismo color (de distinto color para cada jugador), un poco más grandes que el diámetro del agujero pasante.

f- Inicio del juego :

Sortear quién juega primero, por ejemplo lanzando una moneda. Puesto que el jugador que empieza tiene una cierta ventaja, los jugadores alternarán turnos. Por ello, el jugador que comience la primera ronda jugará segundo en la siguiente ronda.

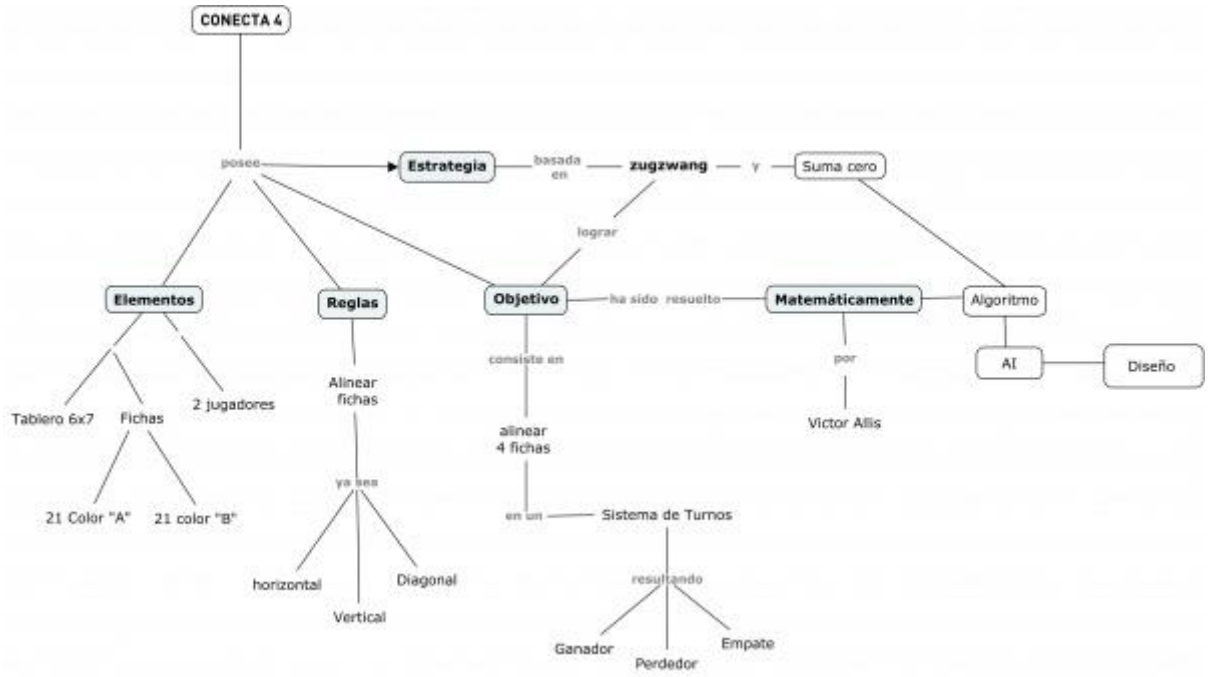
g- Desarrollo del juego :

Por turnos, los jugadores deben introducir una ficha en la columna que prefieran, siempre que no esté completa, y ésta caerá a la posición más baja, ya que la rejilla está dispuesta de forma vertical.

h- Fin del juego:

- Gana la ronda el primer jugador que consiga alinear cuatro de sus fichas en horizontal, vertical o diagonal.
- Si todas las columnas están llenas pero nadie ha conseguido alinear cuatro de sus fichas, hay empate.

i- Estrtegias:



El juego pertenece a la categoría “suma cero” en donde la ganancia o pérdida de un participante equilibra las pérdidas o ganancias de los demás participantes.

> *Zugzwang*/ tsu:ktsvaŋ: Concepto alemán utilizado dentro del juego, se traduce como "obligación de mover". Se dice que un jugador está en zugzwang si cualquier movimiento permitido supone empeorar su situación de juego y eventualmente, perder la partida.

Este juego posee una estrategia matemática resuelta, se analiza el número de posibilidades y estados de las fichas dentro del tablero de 6x7 Existen 4,531,984,531,985,219,092 posiciones para cada espacio del tablero, ya que puede tener una ficha del color “A”, del color “B” o estar vacía.

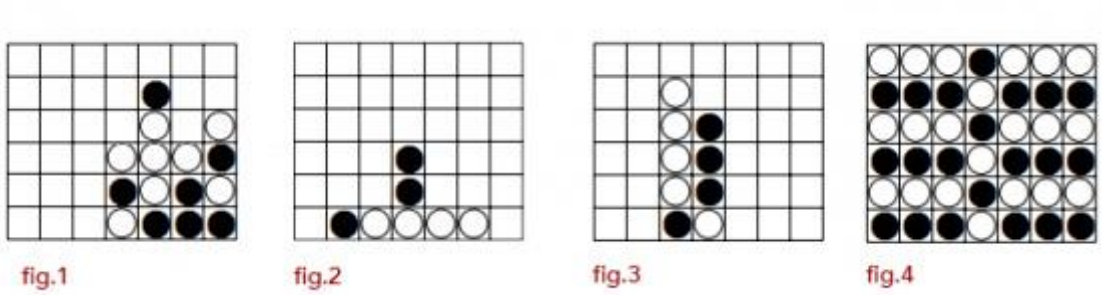


fig.1-2-3 muestran forman en el que el jugador “Blanco” gana,
fig.4 muestra un empate.

5- Jugar ajedrez contra el ordenador:

a-Presentación de trabajo:

Este trabajo presenta el juego de 4 en línea. Los jugadores son: humanos y PC. El juego consiste en colocar 4 fichas del mismo símbolo, El color “rojo” para humano y el color “azul” para el ordenador, de forma colineal, ya sea vertical, horizontal o diagonal.
Hemos desarrollado una función de evaluación simple pero efectiva, que analiza los cuadrados ocupados y desocupados para hacer la mejor forma de una heurística aceptable.

b- Interfaz de la aplicación:

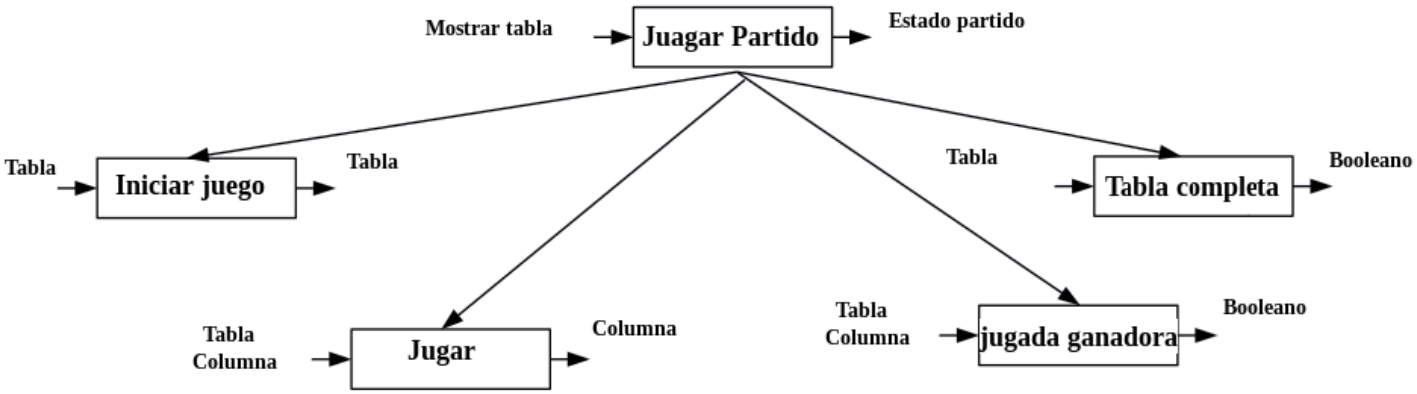


Diagrama de Jugar una partida.

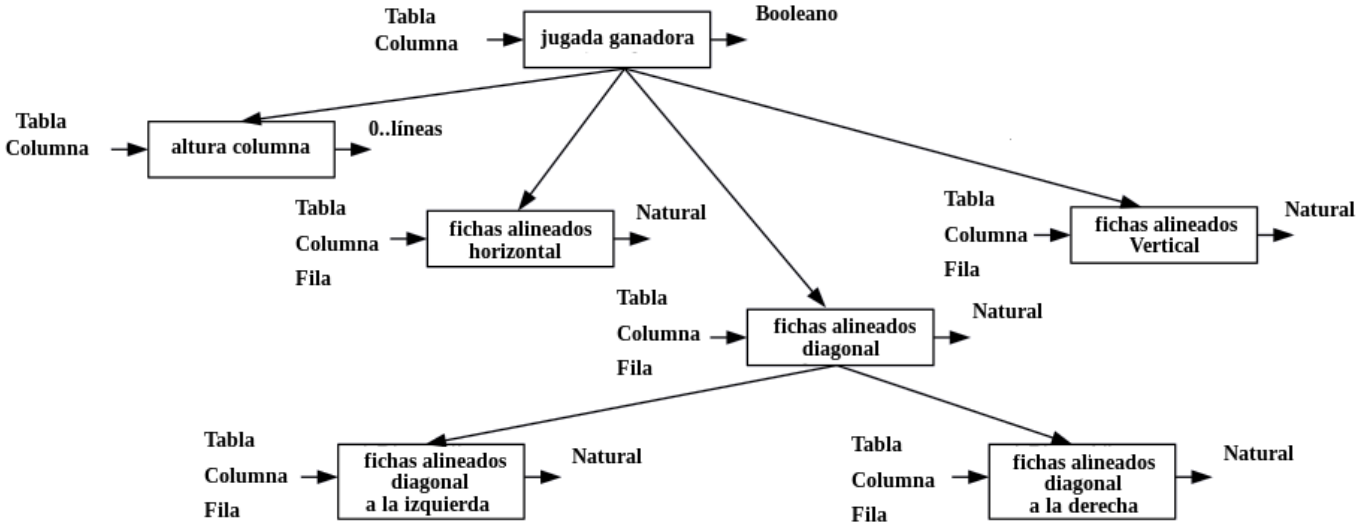


Diagrama de Ganar una partida.

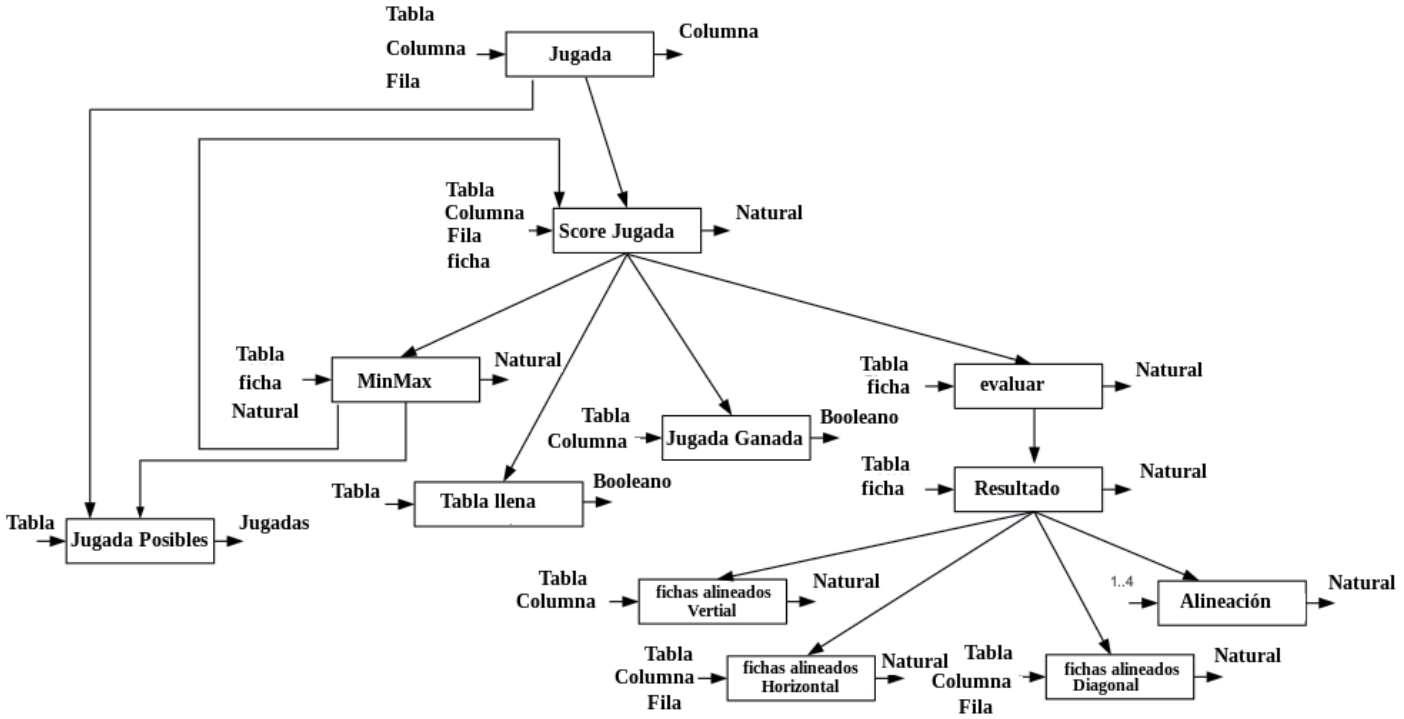
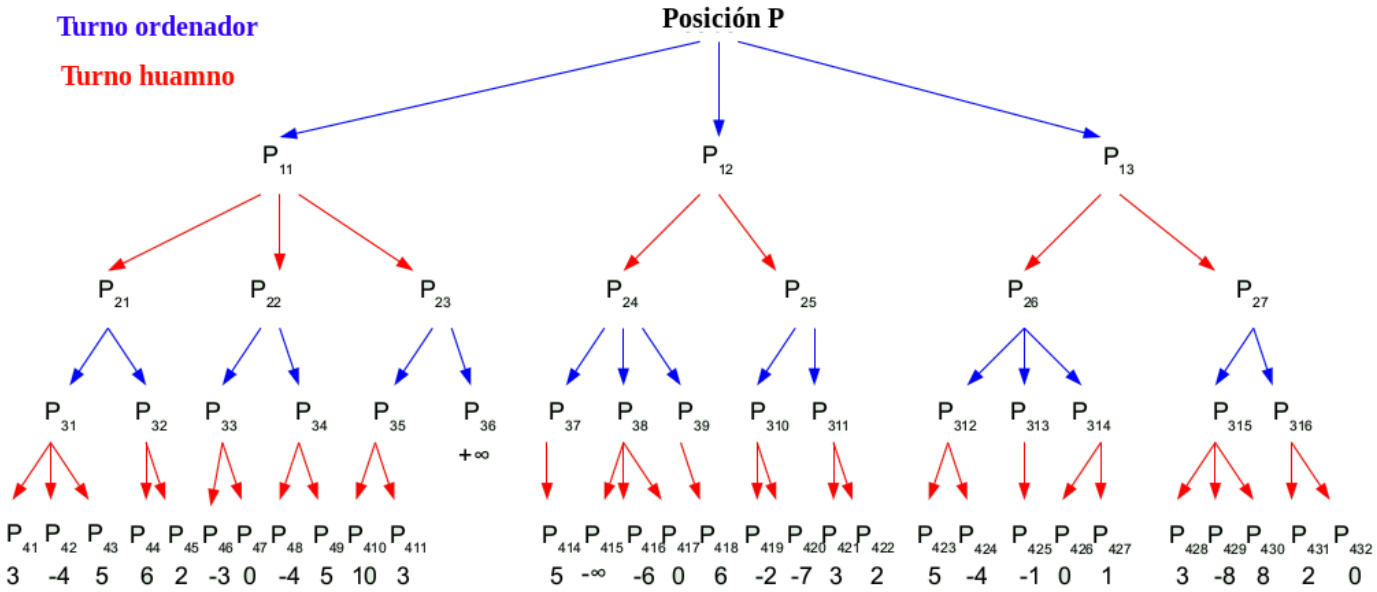


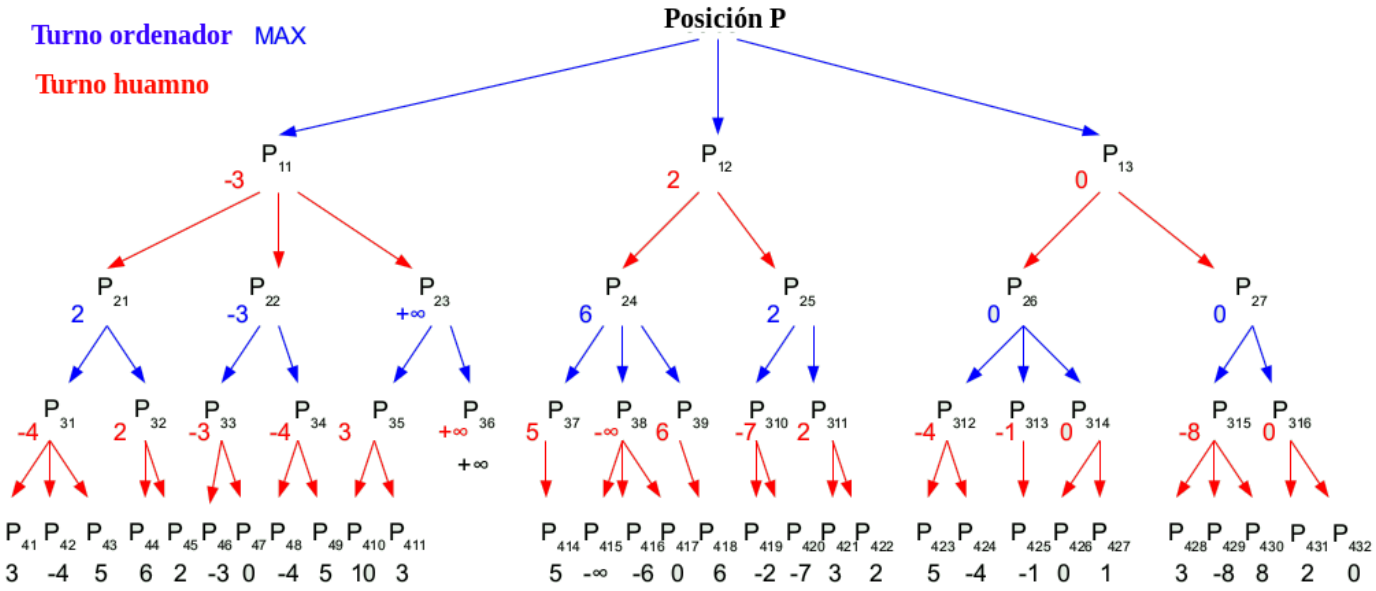
Diagrama de jugar una partida.

c-Introducción del algoritmo MinMax :
Antes de hacer un movimiento dentro del juego, se deben analizar todos los movimientos posibles que podría hacer el enemigo y determinar cuál es el más adecuado, basándose en la secuencia de estos.
El computador pretende que cada uno de los movimientos han tomado lugar, por cada movimiento se analizan todos los posibles movimientos que el contrincante podría realizar en respuesta.
el algoritmo se llama minimax, porque el computador busca minimizar las pérdidas a un máximo.

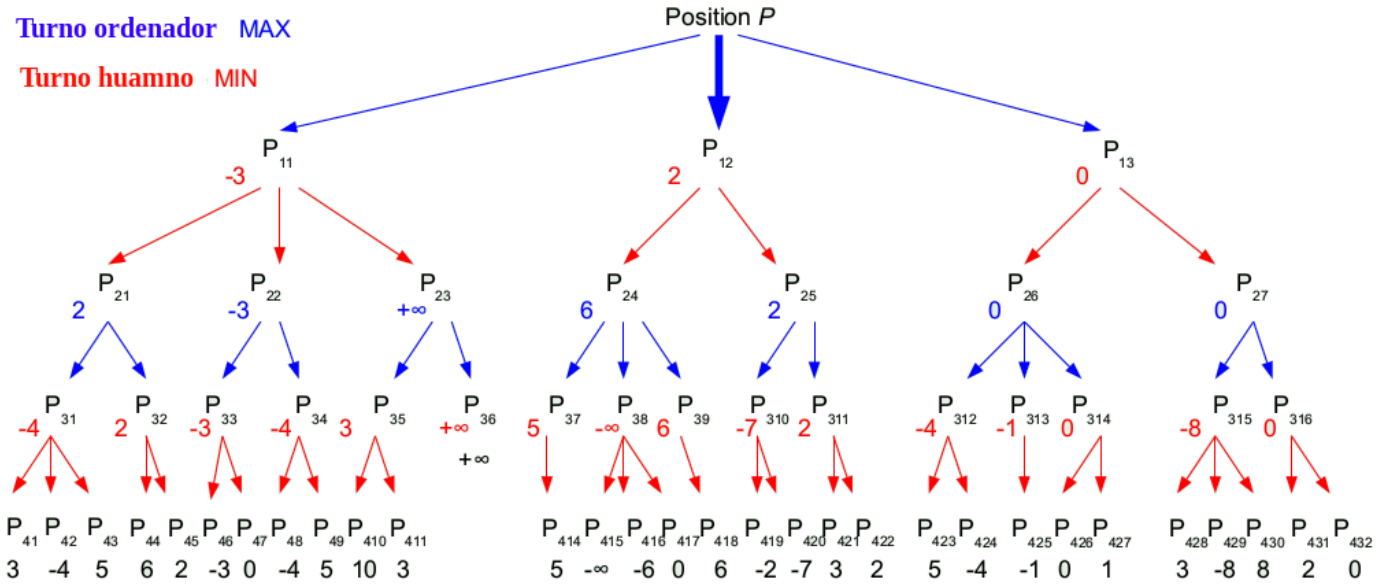
El algoritmo minimax es una toma de decisión para minimizar la perdida máxima en un juego con adversarios. En este algoritmo se conoce el estado del otro jugador, selecciona el mejor movimiento para cada jugador, suponiendo que el contrincante escogerá el peor.
La implementación del algoritmo minimax se llevará a cabo en el juego conocido como conecta 4 o four in line, el cual consiste de un tablero de 6x7 (6 filas, 7 columnas), cuyo objetivo es colocar fichas y conseguir alinear 4 del mismo color según sea el turno de cada jugador.



Algoritmo MinMax: Generar juego y función de evaluación.



Algoritmo MinMax: Resultado (min,max)



Algoritmo MinMax: Elige el juego.

d- Desarrollo

Identificaremos a cada jugador como el jugador MAX y el jugador MIN. MAX será el jugador CPU (la computadora) marcara como objetivo encontrar el conjunto de movimientos que le proporcionen la victoria, para ello se requiere hacer tiradas por max y hacer tiradas por min donde min será el rival a vencer.

Bajo este punto siempre se calcularan jugadas positivas a nuestro favor (cpu) y negativas para el oponente (humano por ejemplo).

Con ello nos lleva a decir que una jugada vencedora para MAX tendrá valor infinito y jugada ganadora para MIN valor negativo infinito.

Deberá existir para el juego una **función de evaluación heurística** que devuelva los valores elevados para indicar buenas situaciones y valores negativos para indicar situaciones favorables al oponente., con ello el algoritmo será capaz de identificar el mejor movimiento.

La búsqueda se realizará hasta una profundidad limitada puesto a que se vuelve costoso en tiempo y memoria realizar la búsqueda completa.

e- Función minimax: Retorna la posición del mejor movimiento en el juego.

Puesto que el juego consta de hacer tiradas en cada una de las 7 columnas para coneguir 4 en fila ya sea diagonal, horizontal o vertical, la función principal hara llamadas a min y max de forma recursiva de acuerdo a una tirada en cada una de e las columnas.

```
 const minimax = (tabla) => {
        let max, beastf = -1,
            beastc = -1,
            turnoMax = 0;
        max = Number.NEGATIVE_INFINITY;

        for (let j = 0; j < 7; j++) {
            if (movimientoValido(tabla, j)) {
                let fila = nuevoMovimiento(tabla, j)

                tabla[fila][j] = 2;

                turnoMax = valormin(tabla, 0, Number.NEGATIVE_INFINITY,
Number.POSITIVE_INFINITY);

                tabla[fila][j] = 0;

                if (max < turnoMax) {
                    max = turnoMax;
                    beastc = j;
                    beastf = fila;

                }
            }
```

```
        }
        tabla[beastf][beastc] = 2
        for (let index = plazas.length - 1; index > -1; index--) {
            if (beastc == plazas[index].getAttribute('id')) {
                if (!plazas[index].classList.contains("turno")) {
                    plazas[index].classList.add("turno");
                    plazas[index].classList.add("jugador-ordenador");
                    break;
                }
            }
        }
    }
```

1) Si hay filas disponibles en la columna a tirar, realiza tirada. Si no trata siguiente columna
2) Coloca la ficha en el tablero, es decir pone un 2(CPU) en la matriz del tablero
3) Guarda la posición en la fila que se colocó la ficha
4) Asigna el valor mínimo retornado por la función valMin en max actual, como parámetro se nota que se envía –infinito como alfa y +infinito como beta.
5) Restauramos el tablero ya que la tirada aún no es oficial.
6) Verifica si el valor retornado es mayor al que se creía era, cambia valores y toma a esa columna como mejor movimiento.
7) Se regresa el mejor movimiento encontrado.

En la función principal del algoritmo se busca el máximo valor es decir la peor puntuación para el rival.
Las siguientes son llamadas a valorMax y valorMin que harán uso de la podada alfa-beta, el cual es una mejora al algoritmo para reducir tiempo de búsquedas, al ralizar podados en las ramas del árbol y expandir todo el árbol puesto a que no existirían mejores jugadas de acuerdo al nodo evaluado.

```
const valormax = (m, profundidad, alpha, beta) => {

        if (estaGanado(m) == 1 || estaGanado(m) == 2)
            return Heuristica(m);
        if (tablaMovimiento(m))
            return Heuristica(m);

        for (let j = 0; j < 7; j++) {
            if (movimientoValido(tabla, j)) {
                let fila = nuevoMovimiento(tabla, j);

                tabla[fila][j] = 2;

                alpha = Math.max(alpha, valormin(m, profundidad + 1, alpha, beta));

                tabla[fila][j] = 0;

                if (alpha > beta) {
                    return beta;
                }
            }
        }
        return alpha;
    }
```

1) Si con la jugada actual gano el jugador, retornar valor calculado
2) Si el tablero está lleno retornar valor calculado
3) Si la profundidad actual supera a la máxima establecida retornar valor calculado
4) Coloca la ficha en el tablero, es decir pone un 2(CPU) en la matriz del tablero
5) Le asigna a alfa el mayor de los valores entre **alfa** y **el Minimo** valor calculado por Min
6) Si la alfa actual es mayor a beta quiere decir que no existe algún otro valor de alfa mejor para el resto del árbol por lo cual se retorna beta. En caso contrario seguirá buscando en las demás posiciones. Si se expandió todo el árbol quiere decir que la última exploración fue el mejor resultado por lo que regresa alfa.

La función max busca la mejor jugada para la computadora y la peor para el rival.
Ahora tenemos a la función min que es similar a max pero busca la mejor jugada para el rival (humano) y la peor para la computadora.

```
    const valormin = (m, profundidad, alpha, beta) => {
```

```
        if (estaGanado(m) == 1 || estaGanado(m) == 2)
            return Heuristica(m);
        if (tablaMovimiento(m))
            return Heuristica(m);
        if (profundidad > Profundidad)
            return Heuristica(m)

        for (let j = 0; j < 7; j++) {
            if (movimientoValido(m, j)) {
                let fila = nuevoMovimiento(m, j);

                m[fila][j] = 1;

                beta = Math.min(beta, valormax(m, profundidad + 1, alpha, beta));

                m[fila][j] = 0;

                if (alpha > beta) {
                    return alpha;
                }
            }
        }
        return beta;
    }
```

El regreso de la heurística aplica de igual manera que en max, el cambio se ve en las líneas marcadas 1 y 2
1) Se le asigna a beta el menor valor que existe entre **beta** y **valorMax** que es el máximo valor calculado por Max.
2) Se realiza la evaluación para el podado, en esta ocasión si alfa es mayor o igual a beta quiere decir que no existe una beta menor por lo que no es necesario seguir explorando el árbol y retornamos alfa. Si el mínimo valor o mejor jugada para oponente se encontró en la última opción a tirar se retorna beta.

f- Función de utilidad (heurística)

```
 const Heuristica = (m) => {
        return Costo(m, 2) - Costo(m, 1);
    }

    const Costo = (m, jugador) => {
        var value = 0;
        for (let i = 0; i < 6; i++) {
            for (let j = 0; j < 7; j++) {
                //para 4
                if (j + 3 < 7) {
                    if (m[i][j] == jugador && m[i][j + 1] == jugador && m[i][j + 2] ==
jugador && m[i][j + 3] == jugador) {
                        return 10000;
                    }
                }
                if (i + 3 < 6) {
                    if (m[i][j] == jugador && m[i + 1][j] == jugador && m[i + 2][j] ==
jugador && m[i + 3][j] == jugador) {
                        return 10000;
                    }
                }
                if (j + 3 < 7 && i + 3 < 6) {
                    if (m[i][j] == jugador && m[i + 1][j + 1] == jugador && m[i + 2][j +
2] == jugador && m[i + 3][j + 3] == jugador) {
                        return 10000;
                    }
                }
                if (j + 3 < 7 && i - 3 > -1) {
                    if (m[i][j] == jugador && m[i - 1][j + 1] == jugador && m[i - 2][j +
2] == jugador && m[i - 3][j + 3] == jugador) {
                        return 10000;
                    }
                }
                if (i + 3 < 6 && j - 3 > -1) {
```

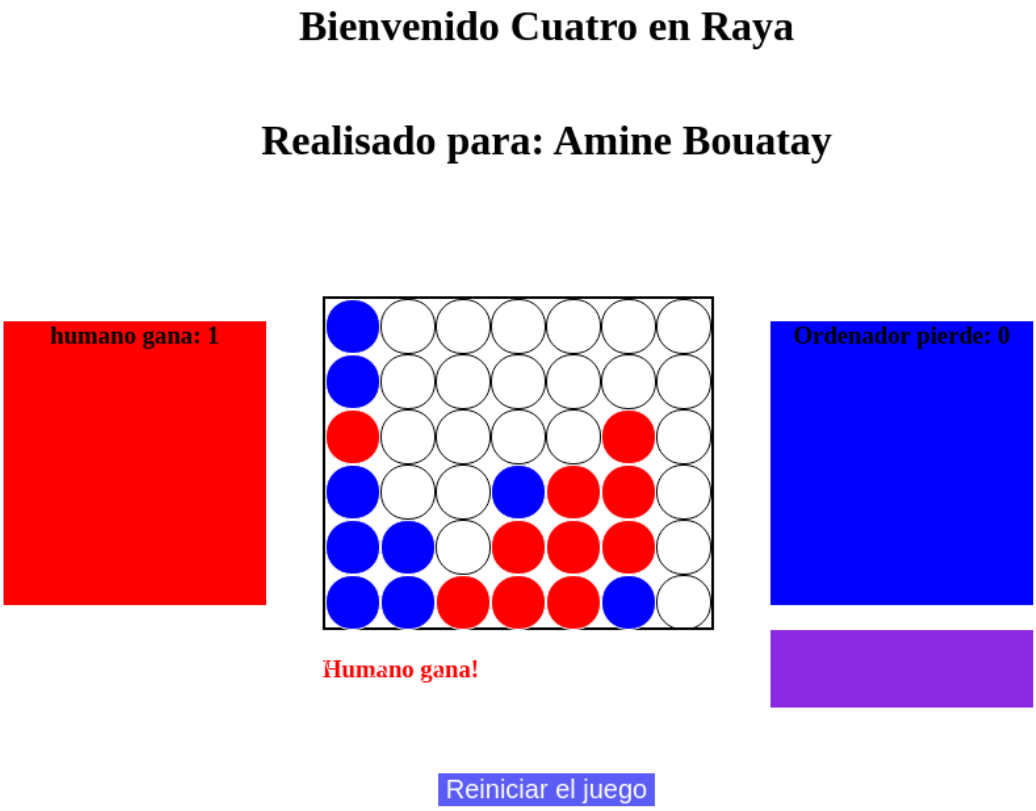
```
                    if (m[i][j] == jugador && m[i + 1][j - 1] == jugador && m[i + 2][j -
2] == jugador && m[i + 3][j - 3] == jugador) {
                        return 10000;
                    }
                }

                //para 3
                if (j + 2 < 7) {
                    if (m[i][j] == jugador && m[i][j + 1] == jugador && m[i][j + 2] ==
jugador) {
                        value = 1000;
                    }
                }
                if (i + 2 < 6) {
                    if (m[i][j] == jugador && m[i + 1][j] == jugador && m[i + 2][j] ==
jugador) {
                        value = 1000;
                    }
                }
                if (j + 2 < 7 && i + 2 < 6) {
                    if (m[i][j] == jugador && m[i + 1][j + 1] == jugador && m[i + 2][j +
2] == jugador) {
                        value = 1000;
                    }
                }
                if (j + 2 < 7 && i - 2 > -1) {
                    if (m[i][j] == jugador && m[i - 1][j + 1] == jugador && m[i - 2][j +
2] == jugador) {
                        value = 1000;
                    }
                }
                if (i + 2 < 6 && j - 2 > -1) {
                    if (m[i][j] == jugador && m[i + 1][j - 1] == jugador && m[i + 2][j -
2] == jugador) {
                        value = 1000;
                    }
                }
                //para 2
                if (j + 1 < 7) {
                    if (m[i][j] == jugador && m[i][j + 1] == jugador) {
                        if (value < 300)
                            value = 300;
                    }
                }
                if (i + 1 < 6) {
                    if (m[i][j] == jugador && m[i + 1][j] == jugador) {
                        if (value < 300)
                            value = 300;
                    }
                }
                if (j + 1 < 7 && i + 1 < 6) {
                    if (m[i][j] == jugador && m[i + 1][j + 1] == jugador) {
                        if (value < 300)
                            value = 300;
                    }
                }
                if (j + 1 < 7 && i - 1 > -1) {
                    if (m[i][j] == jugador && m[i - 1][j + 1] == jugador) {
                        if (value < 300)
                            value = 300;
                    }
                }
                if (i + 1 < 6 && j - 1 > -1) {
                    if (m[i][j] == jugador && m[i + 1][j - 1] == jugador) {
                        if (value < 300)
                            value = 300;
                    }
                }
            }
```

```
        }
        return value;
    }
```

Si la jugada actual logro hacer ganar a CPU (computadora) la función de utilidad regresa 10000 siendo el máximo valor posible en el juego, si quien gana en la tirada es el rival (humano) retornará -10000 siendo el mínimo valor posible en el juego.
En caso contrario que ninguno haya ganado. La función heurística retornara el valor calculado de acuerdo a la siguiente evaluación
h(n) = costo(MAX) – costo(MIN)
Donde costo (MAX) es el **número total de líneas posibles que tiene el jugador MAX para ganar (CPU).**
Y costo (MIN) es el **número total de líneas posibles que tiene el jugador MIN para ganar (humano).**
n es el estado actual en el tablero.

Si el jugador max (CPU) tiene ventaja sobre min, el valor retornado es positivo puesto que el número de líneas será mayor a min.
Si el jugador min (humano) tiene ventaja sobre max, el valor retornado será negativo puesto que el número de línea será mayor a max
Con dicha evaluación si humano tiene ventaja en el tiro sobre cpu, la función valorMin tomara esa jugada y max se encargará de evitarla.



Habiando usado la podada alfa-beta como mejora del algoritmo minimax dentro del juego cuatro en raya y sumado a la heurística los tiempos de respuesta siendo CPU quien haga la primera tirada en el juego son los siguientes:

Profundidad	11	10	8	7	6	5	4
Tiempo	8min	2min	30seg	6seg	1seg	<1seg	<1seg

El resto de las profundidades la respuesta es menor a 1 segundo. A mayor profundidad la capacidad.