SISTEMAS INTELIGENTES

Practica Minimax

Rafael Soria Diez

Objetivo

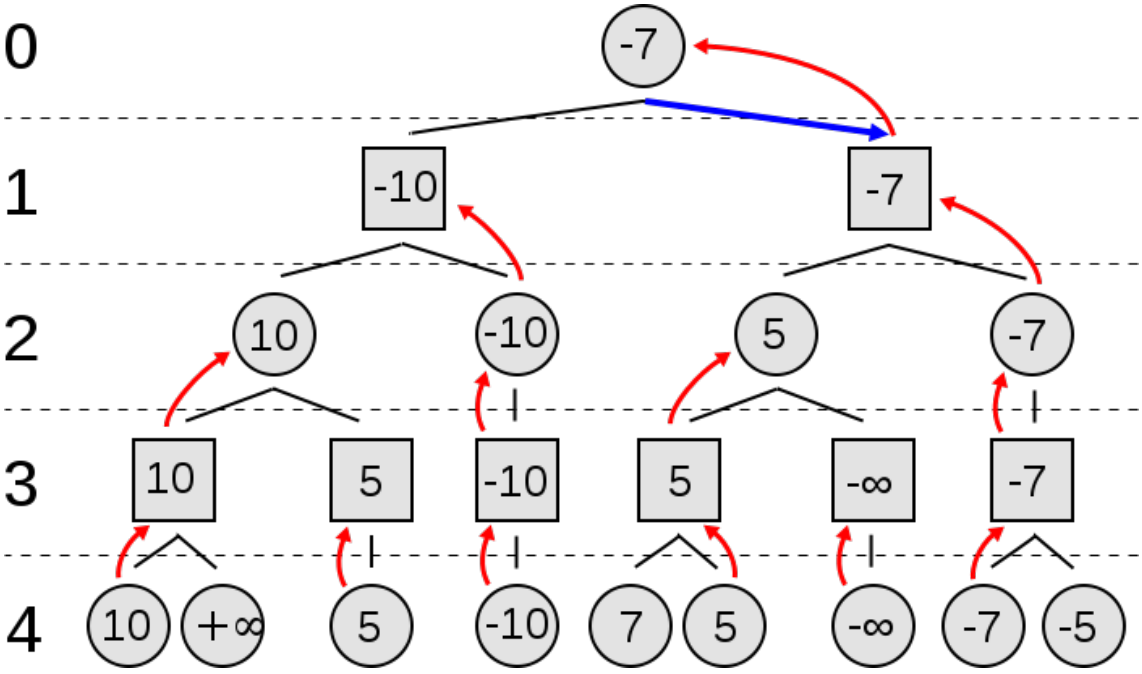
El objetivo de esta práctica es implementar inteligencia al juego conocido como conecta-4 para que sea capaz de competir contra un humano a través del algoritmo minimax y una heurística apropiada.

Minimax

El algoritmo de minimax en simples palabras consiste en la elección del mejor movimiento para el computador, suponiendo que el contrincante escogerá uno que lo pueda perjudicar, para escoger la mejor opción este algoritmo realiza un árbol de búsqueda con todos los posibles movimientos, luego recorre todo el árbol de soluciones del juego a partir de un estado dado, es decir, según las casillas que ya han sido rellenadas.

Los pasos a realizar son los siguientes:

- Generación del árbol de juego. Se generaran todos los nodos hasta llegar a un estado normal.
- Calculo de los valores de la función de utilidad para cada nodo terminal
- Calcular el valor de los nodos superiores a partir del valor de los inferiores. Según si el nivel es MAX o MIN se elegirán los valores mínimos y máximos representados los movimientos del jugador y del oponente, de ahí el nombre de minimax.
- Elegir la jugada valorando los valores que han llegado a nivel superior.



El pseudo-codigo es el siguiente:

```
algoritmo minimax. V(N)
Entrada: nodo N
Salida: valor de dicho nodo
Si N es nodo terminal entonces devolver f(N)
sino
    generar todos los sucesores de N: N1, N2, ..., Nbc

    Si N es MAX entonces devolver max(V(N1), V(N2), ..., V(Nb)) fsi

    Si N es MIN entonces devolver min(V(N1), V(N2), ..., V(Nb)) fsi

fsi
falgoritmo
```

Donde F es la heurística para implementar el sistema de puntuación del algoritmo minimax.

Mi implementación en java es la siguiente:

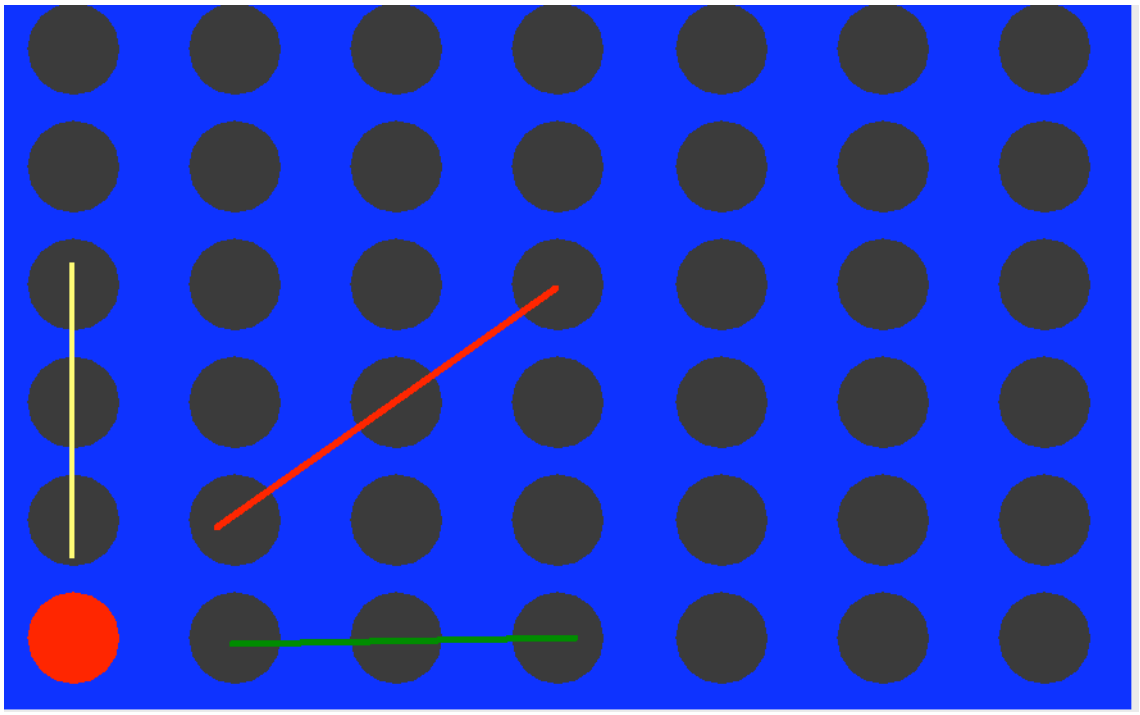
```java
public int V(Tablero tablero1, boolean maxmin, int jugador, int nivelActual){
    int columna,puntuacion,Primernodo,auxPuntuacion;

    columna = puntuacion =  Primernodo = auxPuntuacion = 0;

    if (tablero1.tableroLleno()|| tablero1.cuatroEnRaya() != 0 || nivelActual == NIVEL_DEFECTO) {
        return F(tablero1, m_jugador);

    }else{
        for(int i = 0; i <m_tablero.numColumnas() ; i++){
            //creo la copia del tablero
            Tablero auxTablero = new Tablero(tablero1);
            //miro a ver si se puede poner
            if(auxTablero.ponerFicha(i, jugador) == 0){
                if(maxmin){

                    auxPuntuacion =  V( auxTablero, !maxmin, jugador-1,nivelActual+1);
                    //miro si es la primera vez o si la puntuacion es mejor
                    if(auxPuntuacion > puntuacion || Primernodo == 0){
                         puntuacion = auxPuntuacion;
                         columna= i;
                         Primernodo++;
                    }

                }else{

                    auxPuntuacion =  V( auxTablero, !maxmin, jugador+1,nivelActual+1);

                     if(auxPuntuacion < puntuacion || Primernodo == 0 ){
                        puntuacion = auxPuntuacion;
                        columna = i;
                        Primernodo++;
                    }

                }
            }

        }
        if(nivelActual == 0)return columna;
        return puntuacion;

    }
}
```

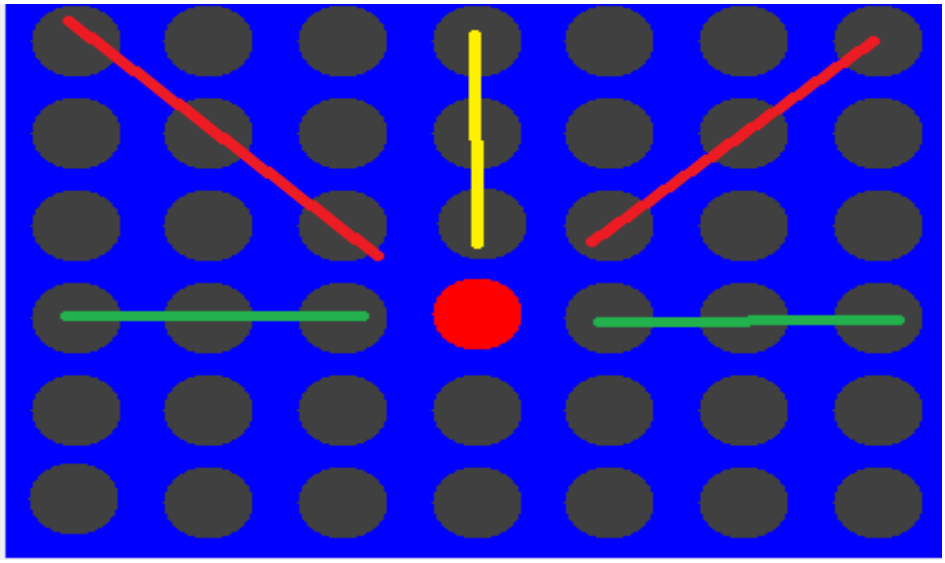
Heurística

Nuestra heurística se encuentra en la función F:

```
public  int F(Tablero tableroCopia,int jugador){
    int puntuacion = 0;

    for(int i = 0; i < tableroCopia.numFilas();i++){
       for(int j = 0; j < tableroCopia.numColumnas();j++){
          // mientras que la casilla no este vacia
          if(tableroCopia.obtenerCasilla(i,j ) != 0 ){
              puntuacion += comprobarHorizontal(i,j,tableroCopia,jugador);
              puntuacion += comprobarVertical(i,j,tableroCopia,jugador);
              puntuacion += comprobarDiagonal(i,j,tableroCopia,jugador);

          }
       }
    }

    return puntuacion;

}
```

En primer lugar recorre todo el tablero horizontalmente, y cuando acaba la fila se sube a la siguiente. De tal forma que mira todas las casillas de la siguiente manera:





Al colocar una ficha, miraremos entre las direcciones horizontal, vertical y diagonal, dependiendo de la puntuación obtenida de la suma de todas las direcciones se colocará en una columna u otra. El sistema de puntuación de las distintas direcciones es común, la cual se mide a partir de la siguiente función.

En esta función vemos que depende de las casillas que tenga fichas ocupadas y/o vacías tiene una puntuación u otra, siendo más favorable la que está a punto de hacer el 4 en raya.

```
public int puntuacion(int ficha,int vacio){
    //le pongo un valor por defecto
    //por si no entra en ninguna de las condiciones de abajo
    int result = 100;

    //caso que mira que hay 4 en raya
    if(ficha == 3)  result =  2*10000000;
    //caso de 3 en raya y uno vacio
    if(ficha == 2 && vacio == 1) result = 10000000;
    else if(ficha == 1 && vacio == 2) result =  10*600;
    //por si encuentra dos adyacentes y de la posicion del tablero
    else if(ficha == 2 && vacio == 0) result = 10*400;
    else if(vacio == 3) result =  10*200;
    return result;
}
```

Vertical

```
public  int comprobarVertical(int i,int j,Tablero tableroCopia,int jugador){
    //mira a ver si no ha encontrado ninguna ficha
    boolean salir = false;
    int result,casillaActual,jugadorCasilla,auxI,ficha,vacio;
    ficha = vacio = result = casillaActual = jugadorCasilla = 0;

    //obtendremos de quien es la casilla inicial
    jugadorCasilla = tableroCopia.obtenerCasilla(i, j);

    // miro arriba
    auxI = i;
    for(int k = 0; k < 3 && !salir;k++){
        auxI++;
        if(auxI >= 0 && auxI <= 5 && j>= 0 && j <=6){
            //miro a quien pertenece la casilla
            casillaActual = tableroCopia.obtenerCasilla(auxI, j);
            if(casillaActual == 0) vacio++;
            else if (casillaActual == jugadorCasilla) ficha++;
            else if (casillaActual != jugadorCasilla)salir = true;
        }else salir = true;

    }
    result = puntuacion(ficha,vacio);

    //si es el jugador contrario a la jugada
    if(jugadorCasilla != jugador) result *= -1;
    return result;
}
```

En este método en primer lugar miramos a quien pertenece la casilla inicial en la que estamos por lo tanto empezamos en la situación de uno en raya , una vez obtenido al jugador que pertenece la casilla, miramos tres posiciones hacia “arriba” con la restricción de que si encuentra un jugador contrario, y si se sale del tablero para de observar.

Una vez obtenida la situación le ponemos a la variable result una puntación de la función puntuación descrita anteriormente.

Por último, si la casilla que encontramos no pertenece al jugador que le pertenece la jugada se le multiplica la puntuación obtenida por -1.

Horizontal

```
public  int comprobarHorizontal(int i,int j,Tablero tableroCopia,int jugador){
    //mira a ver si no ha encontrado ninguna ficha
    boolean salir = false;
    int result,casillaActual,jugadorCasilla,auxJ,vacio,ficha;
    vacio = ficha = result = casillaActual = jugadorCasilla = 0;

    //obtendremos de quien es la casilla inicial
    jugadorCasilla = tableroCopia.obtenerCasilla(i, j);

    auxJ = j;
    for(int k = 0; k < 3 &&  !salir ;k++){
       auxJ++;
        if(i >= 0 && i <= 5 && auxJ >= 0 && auxJ <=6){
            casillaActual = tableroCopia.obtenerCasilla(i, auxJ);
            if(casillaActual == 0) vacio++;
            else if (casillaActual == jugadorCasilla) ficha++;
            else if (casillaActual != jugadorCasilla)salir = true;
        }else salir = true;

    }
    //la puntuacion obtenida por la derecha
    result = puntuacion(ficha,vacio);
```

La implementación es igual que la función vertical con la diferencia que miro hacia la “derecha”. Una vez obtenida las condiciones de las casillas, asignamos a la variable result, una serie de puntuación con la función descrita anteriormente. Una vez mirado la parte derecha, procedo a hacer lo mismo con la parte izquierda.

```
    //miro la izquierda
    auxJ = j;
    vacio = ficha = 0;
    for(int k = 0; k < 3 &&  !salir ;k++){
        auxJ--;//aumentamos aqui la posicion
        //obtengo la casilla
        if(i >= 0 && i <= 5 && auxJ >= 0 && auxJ <=6){
            casillaActual = tableroCopia.obtenerCasilla(i, auxJ);
            if(casillaActual == 0) vacio++;
            else if (casillaActual == jugadorCasilla) ficha++;
            else if (casillaActual != jugadorCasilla)salir = true;
        }else salir = true;

    }

    int izquierda = puntuacion(ficha,vacio);
    result += izquierda;

    if(jugadorCasilla != jugador) result *= -1;
    return result;
```

Diagonal

La función diagonal sigue el mismo procedimiento que el horizontal

```
public  int comprobarDiagonal(int i,int j,Tablero tableroCopia,int jugador){
    //mira a ver si no ha encontrado ninguna ficha
    boolean salir = false;
    int result,casillaActual,jugadorCasilla,auxJ,auxI;
    int vacio,ficha;

    vacio = ficha = result = casillaActual = jugadorCasilla = 0;

   //diagonal-derecha-abajo
    jugadorCasilla = tableroCopia.obtenerCasilla(i, j);
    auxJ = j;
    auxI = i;
    for(int k = 0; k < 3 &&  !salir && j-3 >= 0 ;k++){
         auxI++;
         auxJ--;
         if(auxI >= 0 && auxI <= 5 && auxJ >= 0 && auxJ <=6){
            casillaActual = tableroCopia.obtenerCasilla(auxI, auxJ);
            if(casillaActual == 0) vacio++;
            else if (casillaActual == jugadorCasilla) ficha++;
            else if (casillaActual != jugadorCasilla)salir = true;
        }else salir = true;

     }
     result = puntuacion(ficha,vacio);
```

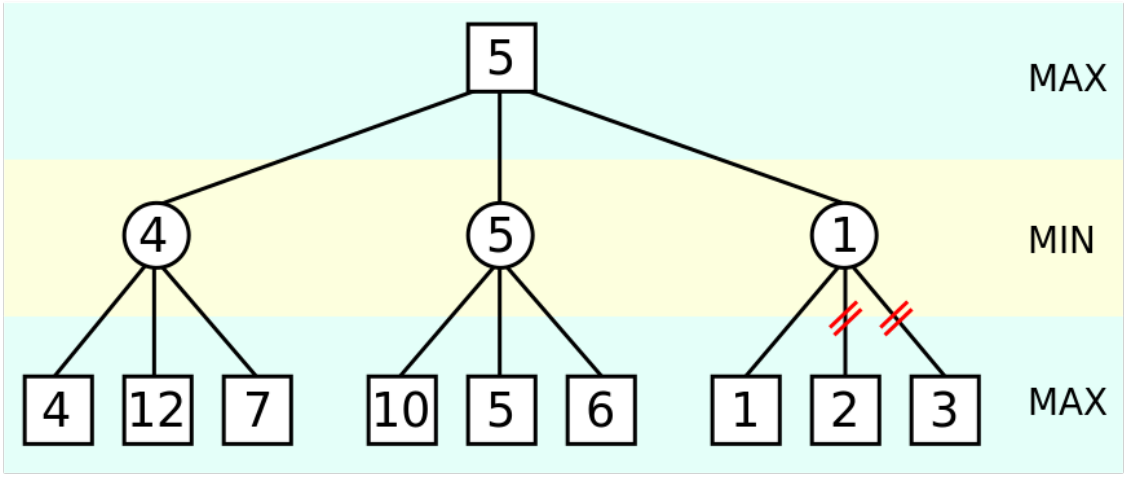
```
 ficha = vacio = 0;
 auxJ = j;
 auxI = i;
 for(int k = 0; k < 3 && !salir && j+3 >= tableroCopia.numColumnas() ;k++){
     auxI++;
     auxJ++;
     if(auxI >= 0 && auxI <= 5 && auxJ >= 0 && auxJ <=6){
        casillaActual = tableroCopia.obtenerCasilla(auxI, auxJ);
        if(casillaActual == 0) vacio++;
        else if (casillaActual == jugadorCasilla) ficha++;
        else if (casillaActual != jugadorCasilla)salir = true;
    }else salir = true;

 }
 int diagonalDerecha = puntuacion(ficha,vacio);
 result += diagonalDerecha;

if(jugadorCasilla != jugador) result *= -1;
return result;
```

Poda Alfa y Beta

Es una técnica de búsqueda que reduce el número de nodos evaluados en un árbol de juego, como el algoritmo de minimax explicado anteriormente. La función principal de esta poda es simplificar el coste temporal que tiene el algoritmo de minimax.

La poda alfa-beta toma dicho nombre de la utilización de dos parámetros que describen los limites sobre los valores hacia atrás que aparecen a lo largo de cada camino.

- α es el valor de la mejor opción hasta el momento del camino para MAX, esto implicara por lo tanto la elección del valor más alto.
- B es el valor de la mejor opción hasta el momento a lo largo del camino para MIN, esto implicara por lo tanto la elección del valor más bajo.



El pseucodigo es el siguiente:

```
función alfa-beta(nodo //en nuestro caso el tablero, profundidad, α, β, jugador)
    si nodo es un nodo terminal o profundidad = 0
        devolver el valor heurístico del nodo
    si jugador1
        para cada hijo de nodo
            α := max(α, alfa-beta(hijo, profundidad-1, α, β, jugador2))
            si β≤α
                romper (* poda β *)
        devolver α
    si no
        para cada hijo de nodo
            β := min(β, alfa-beta(hijo, profundidad-1, α, β, jugador1))
            si β≤α
                romper (* poda α *)
        devolver β
```

La implantación del código es la siguiente:

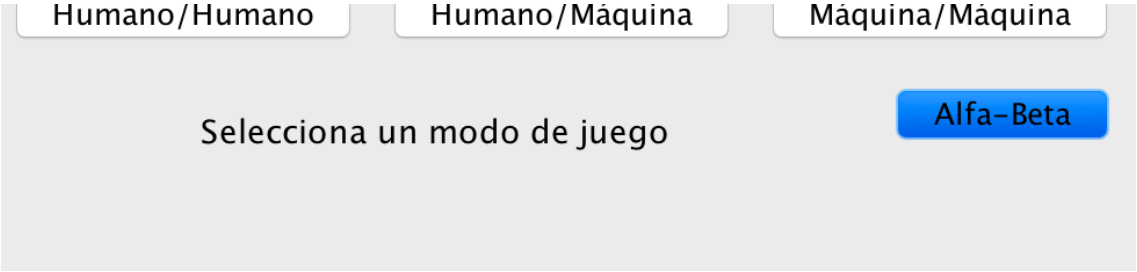
```
public int VAlfayBeta(Tablero tablero1, boolean maxmin, int jugador, int nivelActual,int alpha, int beta){
    int columna,puntuacion,primerNodo,auxPuntuacion;
    columna = puntuacion =  primerNodo = auxPuntuacion = 0;

    if (tablero1.tableroLleno()|| tablero1.cuatroEnRaya() != 0 || nivelActual == NIVEL_DEFECTO) {
        return F(tablero1, m_jugador);

    }else{
        for(int i = 0; i <m_tablero.numColumnas() ; i++){
            //creo la copia del tablero
            Tablero auxTablero = new Tablero(tablero1);
            //miro a ver si se puede poner
            if(auxTablero.ponerFicha(i, jugador) == 0){
                // si es max
                if(maxmin){

                    alpha =  VAlfayBeta( auxTablero, !maxmin, jugador-1,nivelActual+1,alpha,beta);
                    if(alpha > puntuacion || primerNodo == 0){
                         puntuacion = alpha;
                         columna= i;
                         primerNodo++;
                    }
                    if(alpha >= beta) return beta;
                }else{
                    beta =  VAlfayBeta( auxTablero, !maxmin, jugador+1,nivelActual+1,alpha,beta);
                    if(beta < puntuacion || primerNodo == 0 ){
                        puntuacion = beta;
                        columna = i;
                        primerNodo++;
                    }
                     if(alpha >= beta) return alpha;
                }
            }
        }
        if(nivelActual == 0)return columna;
        return puntuacion;
```

Podemos ver que el método es a simple vista igual, con la diferencia de la condiciones de que si alfa es mayor de beta, devuelve ya directamente alfa o beta dependiendo de si es MAX o MIN.

La manera para eligir si queremos ejecutar el juego con la podo o no es la siguiente:

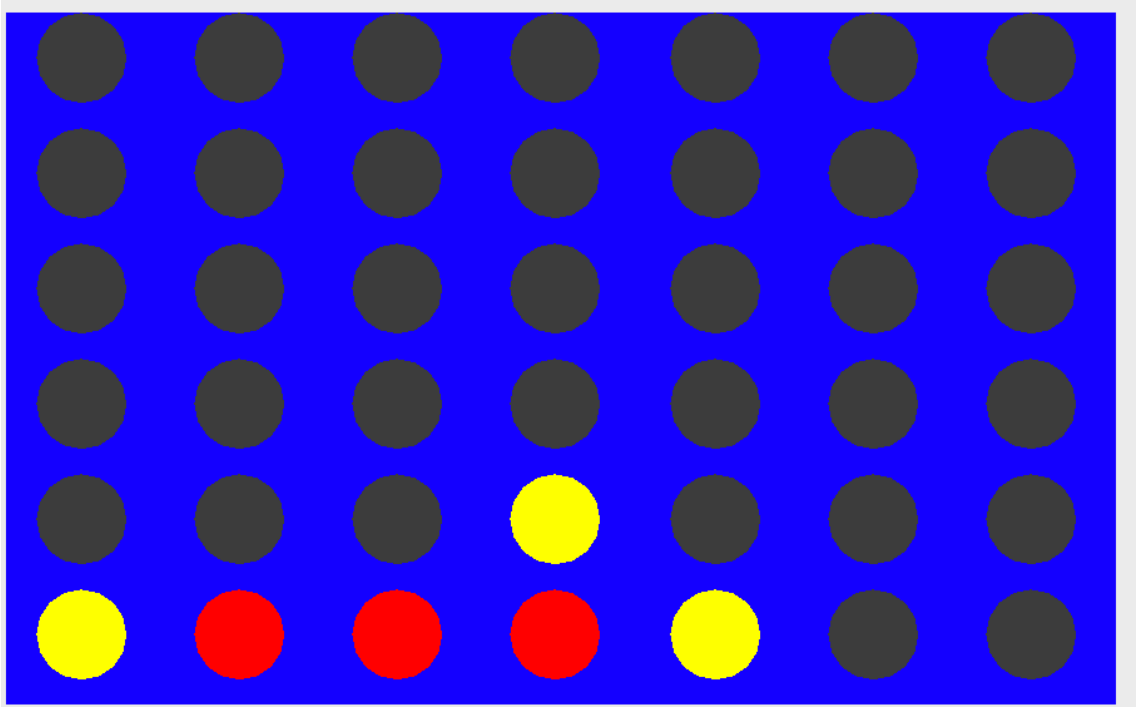
```
$
//aqui se elige si quieres poda
if(Interfaz.alfaBeta){
    columnaFinal = VAlfayBeta(tablero,maxMin,m_jugador,nivelActual,Integer.MIN_VALUE, Integer.MAX_VALUE);
}else{
    columnaFinal = V(tablero,maxMin,m_jugador,nivelActual);
}
```

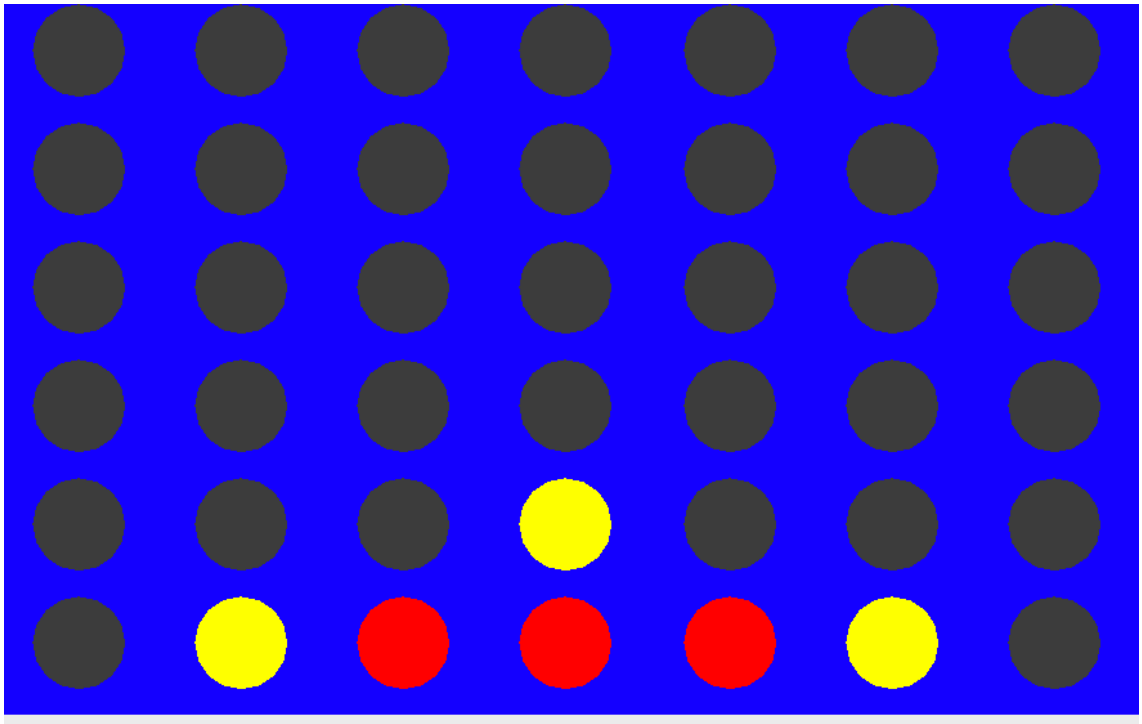
A través de un botón de la interfaz elegimos poder activarla o no:



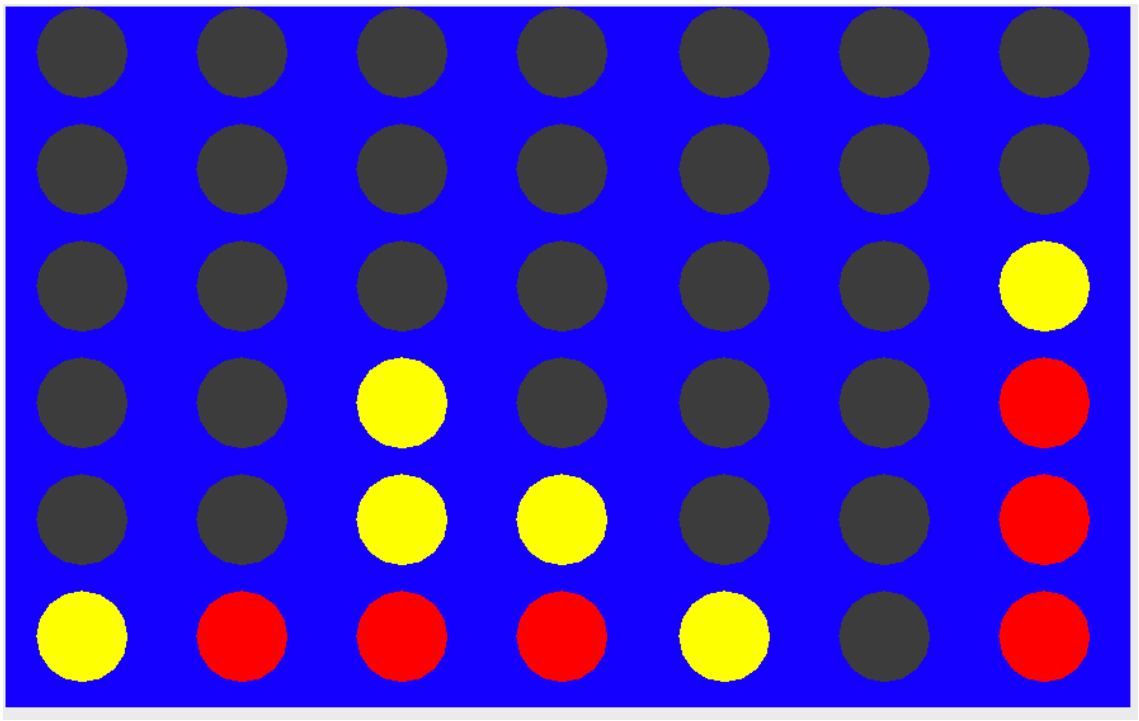
Pruebas

Esta situación vemos cómo consigue cerrar horizontalmente y verticalmente:

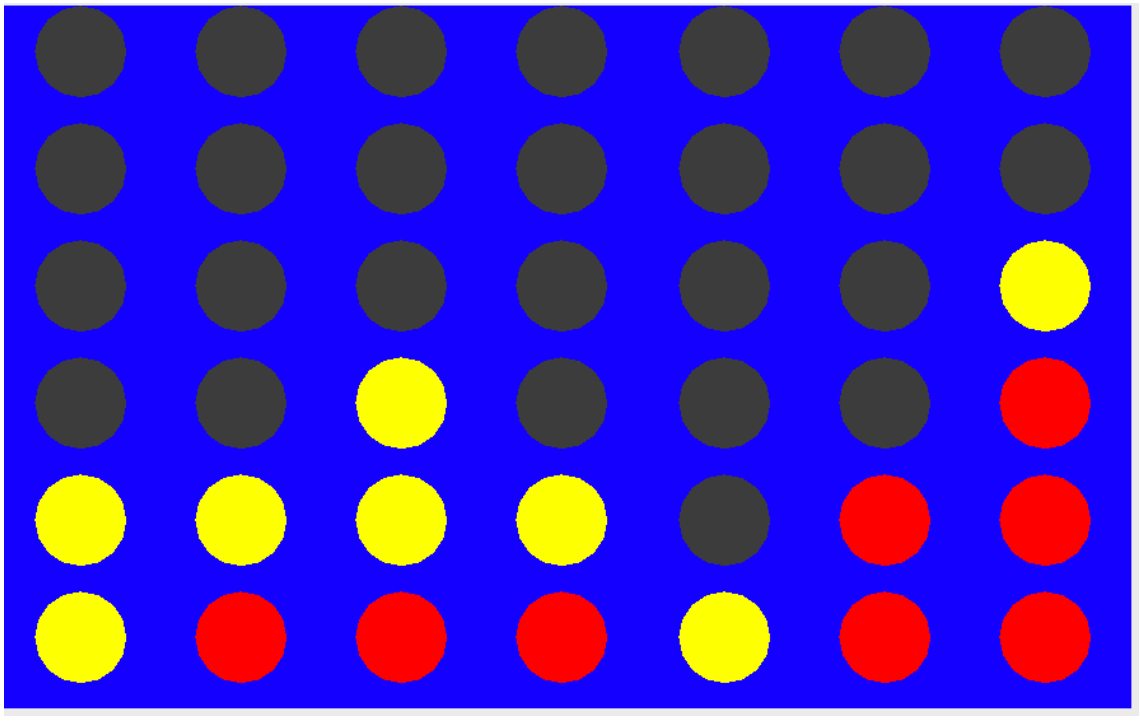




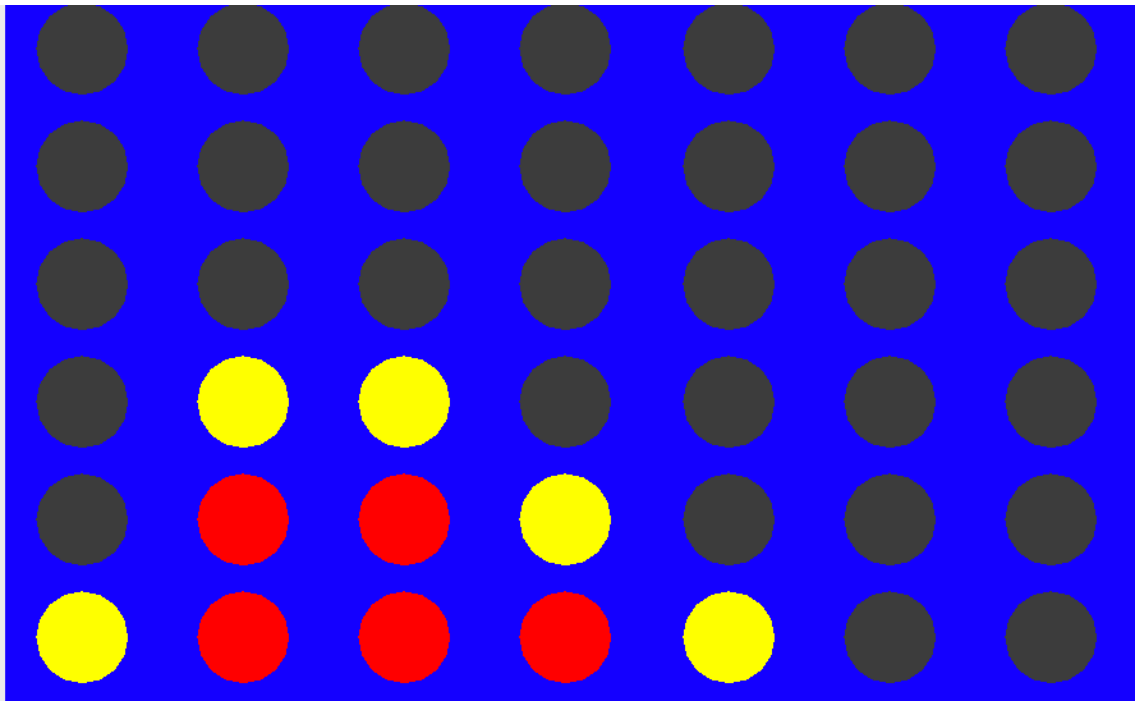
En este caso vemos como cierra oportunidad de hacer 4 en raya.

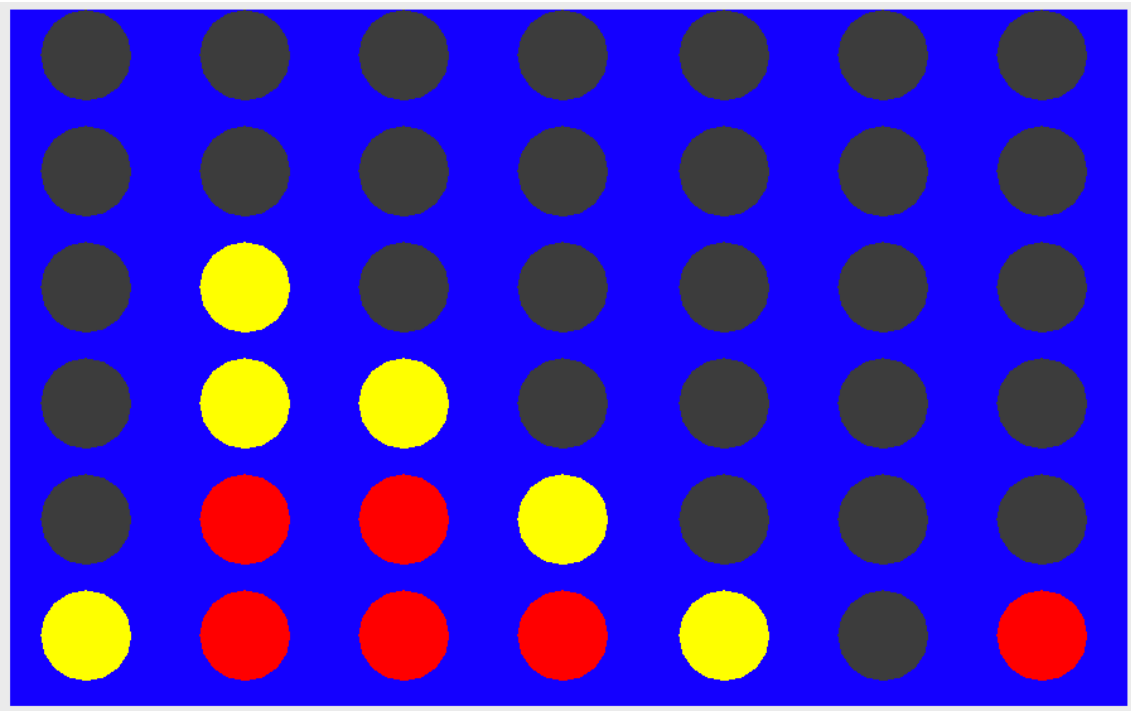


Si continuamos un poco jugando, vemos como es capaz de hacer un 4 en raya horizontalmente:

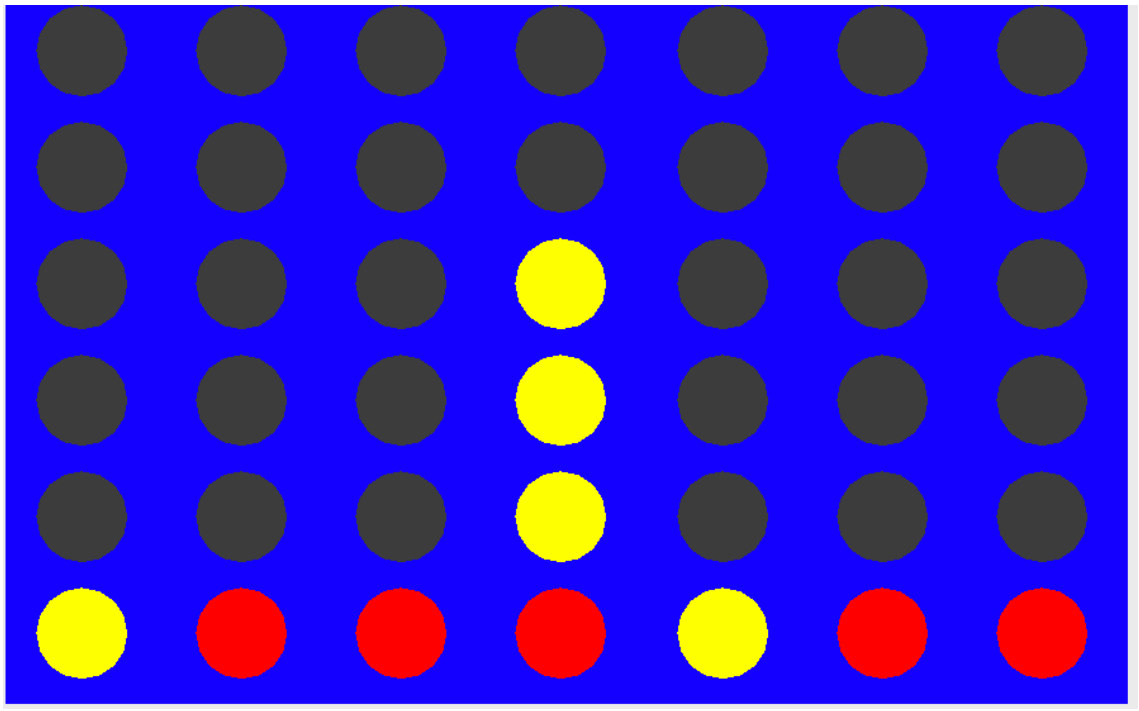


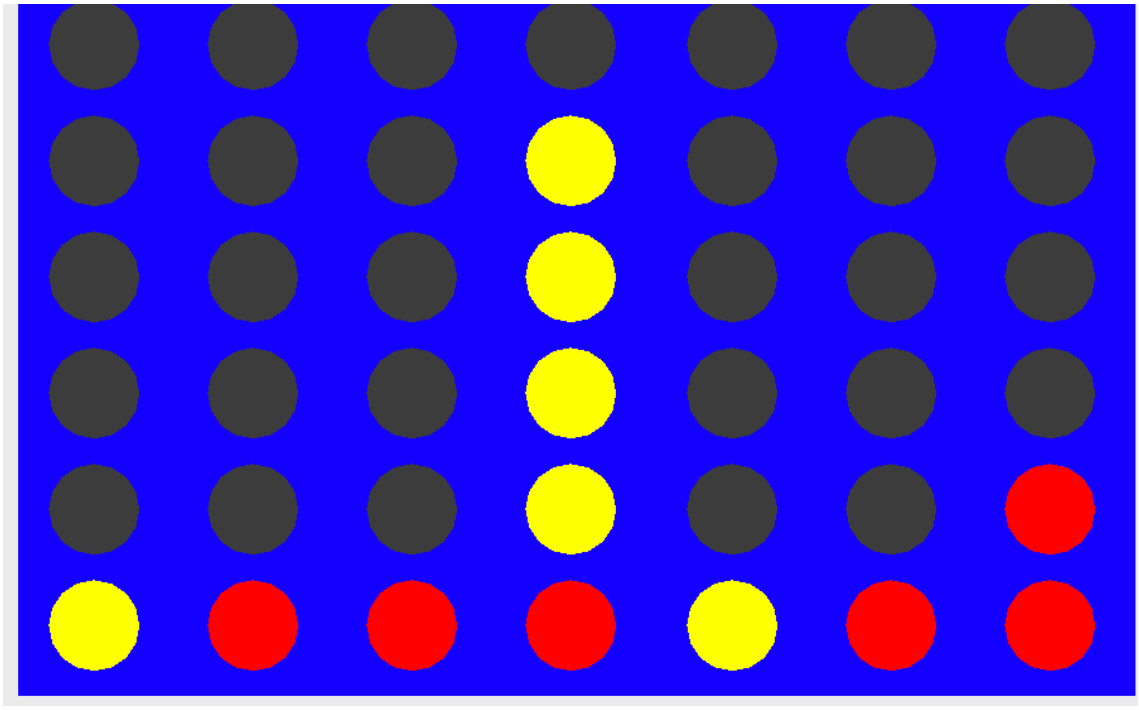
En la siguiente situación vemos como el jugador 2 está a punto de hacer un 4 en raya diagonalmente , si le “dejo” ganar, vemos como pone la casilla correspondiente para ganar.



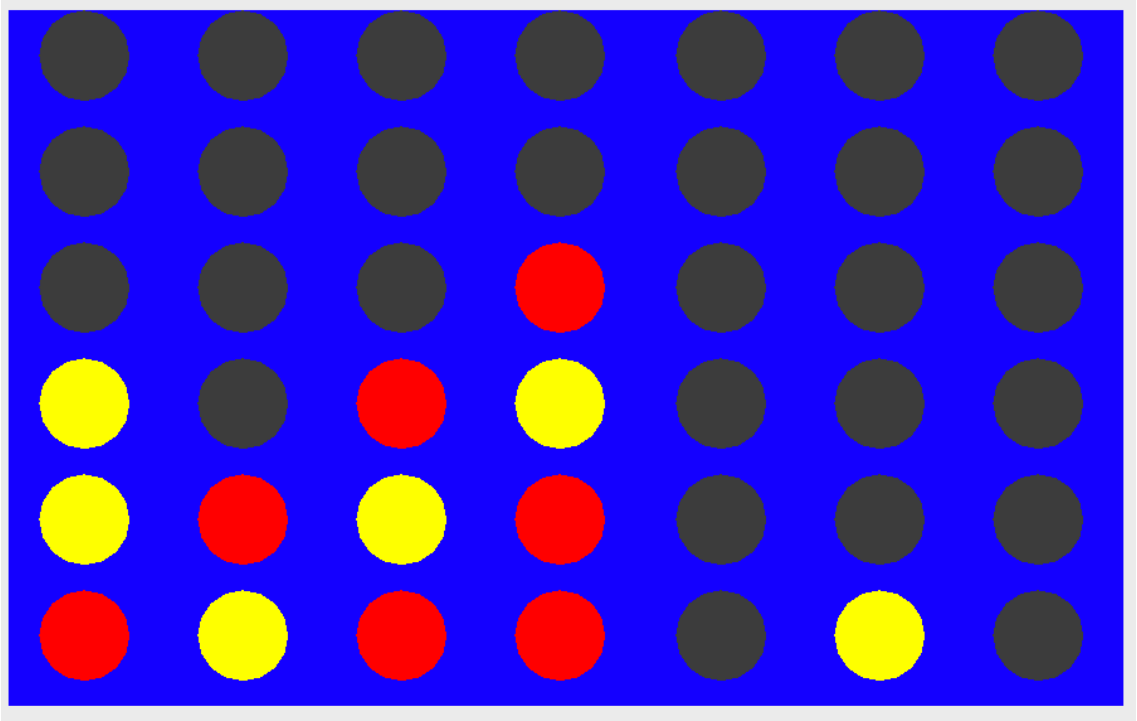


En esta situación vemos cómo puede ganar verticalmente:





Sin embargo hay algunos casos que no consigue ganar o intentar parar el 4 en raya como el siguiente:



Conclusión

En mi opinión el método minimax es una opción muy interesante para adentrarse en la inteligencia artificial sobre todo en el ámbito de juegos.

La función de minimax en si no tiene una gran dificultad elevada, ya que en mi caso la dificultad recae en pensar en una buena heurística para implementar a la maquina una buena estrategia para intentar ganar.

Además la poda alfa-beta es una poda sencillita que permite comprobar lo importante que es reducir el coste temporal en los algoritmo cuyo coste supera o es igual a exponencial.