Universidad Autónoma de Baja California

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

Ing. en Computación

Laboratorio de Inteligencia Artificial

Práctica No. 4

Algoritmo Minimax con mejora Alfa-Beta

Objetivo: Implementar el algoritmo minimax en un juego de 2 contrincantes para poder vencerlo o jugar a su ritmo.

Introducción:

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.

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.

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.

```
public int miniMax(int m[][]) {
    int mejor_mov = -1;
    int max, max_actual;
    max = Integer.MIN_VALUE; //-infinito será el max por el momento
    for(int j=0;j<Ventana.COL;j++) {
        if(Tablero.disPonible(m, j)) {                          //1)
            Tablero.insertarFigura(m, j, CPU);                  //2)
            int temp = Tablero.getY()/Ventana.SIZE;             //3)
max_actual = valorMin(m,0,Integer.MIN_VALUE,Integer.MAX_VALUE); //4)
            m[temp][j]=0;                                       //5)
            if(max_actual > max) {                              //6)
                max = max_actual;
                mejor_mov = j;
            }
        }
    }
    return mejor_mov;                                           //7)
}
```

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.

```
public int valorMax(int m[][], int prof, int alfa, int beta) {
    int isFin = Tablero.isOver(m);
    if(isFin==WIN ) {                                           //1)
        return heuristica(m);
    } else {
        if(isFin == TIE) {                                      //2)
            return heuristica(m);
```

```
        } else {
            if(prof>MAX) {                                          //3)
                return heuristica(m);
            } else {
                for(int j=0;j<Ventana.COL;j++) {
                    if(Tablero.disPonible(m, j)) {                  //4)
                        Tablero.insertarFigura(m, j, CPU);
int temp = Tablero.getY()/Ventana.SIZE;
                    alfa = max(alfa,valorMin(m,prof+1,alfa,beta)); //5)
                        m[temp][j]=0;
                        if(alfa>=beta)                              //6)
                            return beta;
                    }
                }
                return alfa;
            }
        }
    }
}
```

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 **elMinimo** 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.

```
public int valorMin(int m[][], int prof, int alfa, int beta) {
    int isFin = Tablero.isOver(m);
    if(isFin==WIN ) {
        return heuristica(m);
    } else {
        if(isFin == TIE) {
            return heuristica(m);
        } else {
            if(prof>MAX) {
                return heuristica(m);
            } else {
                for(int j=0;j<Ventana.COL;j++) {
                    if(Tablero.disPonible(m, j)) {
                        Tablero.insertarFigura(m, j, H);
                        int temp = Tablero.getY()/Ventana.SIZE;
beta = min(beta,valorMax(m,prof+1,alfa,beta));                      //1)
                        m[temp][j]=0;
```

```
                    if(alfa>=beta)                          //2)
                        return alfa;
                }
            }
            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.

Función de utilidad (heurística)

```
public int heuristica(int m[][]) {
    int costo=0;
    if(Tablero.isOver(m)==WIN) {
        if(Tablero.getPlayer()==CPU)
            return 10000;
        else
            return -10000;
    }
    costo = costoU(m,CPU,H) - costoU(m,H,CPU);  //1)
    return costo;
}
```

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.

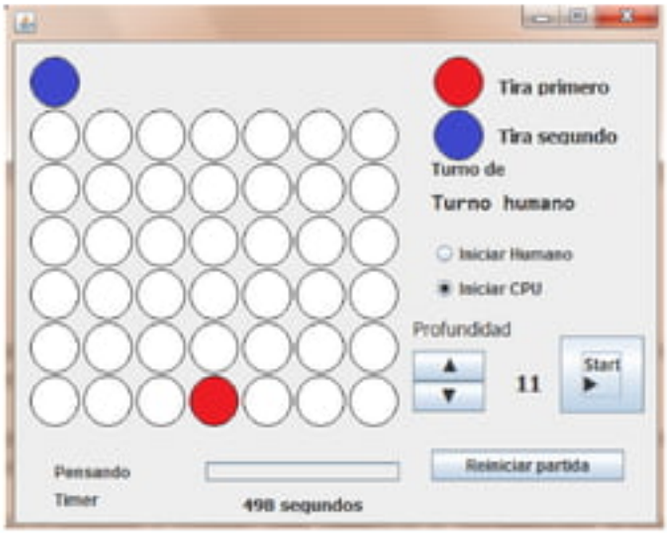
Si el jugador max (CPU) tiene ventaja sobre min, el valor retornado es positivo puesto que el número de líneas será mayor a min.

Si el jugador min (humano) tiene ventaja sobre max, el valor retornado será negativo puesto que el número de línea será mayor a max

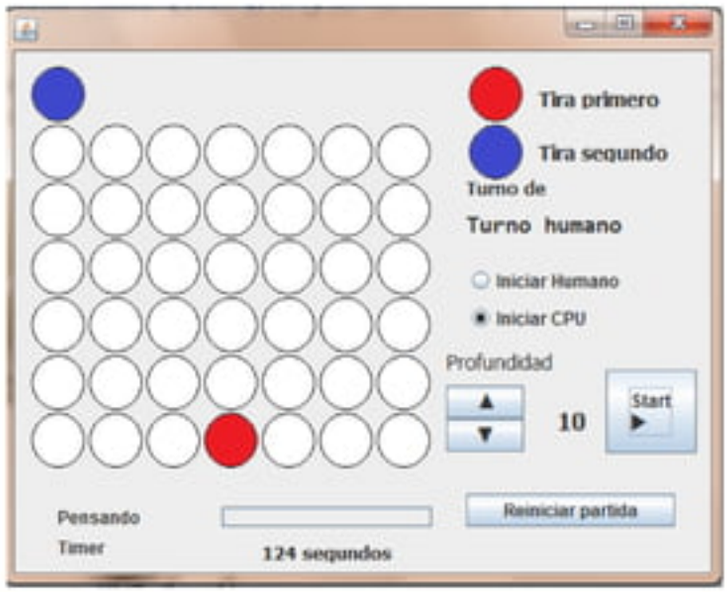
Con dicha evaluación si humano tiene ventaja en el tiro sobre cpu, la función valorMin tomara esa jugada y max se encargará de evitarla.

Habiando usado la podada alfa-beta como mejora del algoritmo minimax dentro del juego conecta 4 y sumado a la heurística los tiempos de respuesta siendo CPU quien haga la primera tirada en el juego son los siguientes:

Profundidad = 11
8.3 minutos

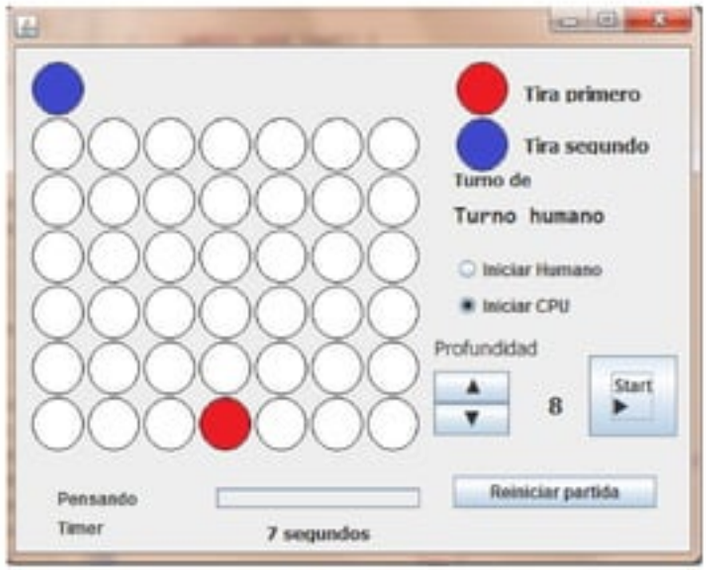


Profundidad = 10
2.06 minutos



Profundidad = 9
31 segundos

Profundidad = 8
7 segundos



El resto de las profundidades la respuesta es menor a 1 segundo. A mayor profundidad la capacidad de ganar para la computadora es mayor pero el tiempo de respuesta aumenta.

Conclusiones

El algoritmo minimax depende mucho de la función de utilidad para tomar la decisión correcta, por lo que debe ser estudiado el juego para conseguir una buena heurística.

A mayor profundidad ser a más dificil derrotar a la computadora, pero eso trae un costo importante en tiempo y memoria, por lo que se requieren aplicar mejoras a dicho algoritmo como lo es alfa-beta. Si se desea conseguir una maquina invencible será necesario aplicar otras mejoras como negamax , scout, etc. Para lograr explorar una profundidad mayor. Aunque entre más turnos existan en el juego menor se vuelve el tiempo de respuesta, el algoritmo forma la estrategia desde el inicio del juego.

Referencias

http://www.lsi.upc.edu/~bejar/ia/material/trabajos/Algoritmos_Juegos.pdf