Práctica 3

Mikel Barrenetxea y Amaia Lobato
Diseño de algoritmos

En esta práctica el objetivo es diseñar un juego donde el rival será la máquina. Tendremos que diseñar el algoritmo de tal manera que la computadora estudie qué movimiento le conviene hacer para ganar o impedir que nosotros ganemos.

Conecta 4

En nuestro caso, hemos elegido implementar el juego conecta 4. Se trata de un juego de mesa para dos jugadores. Para jugar se necesitan un tablero vertical (de 6 filas y 7 columnas) y 42 fichas (21 de un color y las 21 restantes de otro color).

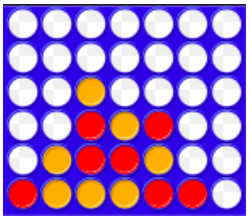

Figura 1: Tablero del juego con fichas amarillas y rojas

El objetivo del juego es alinear cuatro fichas del mismo color en horizontal, en vertical o en diagonal. Cada jugador tiene 21 fichas de un color. Por turnos, cada jugador mete una ficha en una de las columnas del tablero (siempre que dicha columna no esté llena) y la ficha cae al fondo de la columna. Gana el primero que consigue alinear 4 fichas de su color. En caso de que el tablero se llene y ni uno de los dos jugadores haya conseguido alinear 4 fichas, hay empate.

Conecta 4 es un juego de información perfecta, ya que los jugadores conocen todo lo que ha sucedido desde el comienzo del juego hasta que llega su turno.

Algunas de las estrategias para jugar a conecta 4 son las siguientes:

1. **Colocar fichas en la columna central del tablero**
   Esto da más oportunidades para hacer conexiones. Como la columna central tiene 3 columnas a su derecha y otras 3 a su izquierda, tener fichas en la columna central significa que se pueden hacer conexiones hacia cualquier dirección.
2. **Planear por adelantado qué movimientos hacer**
   Los movimientos en el juego conecta 4 fuerzan al jugador o a su oponente a hacer algún movimiento para evitar que el rival gane. Aprovecharse de este tipo de situaciones puede resultar muy útil.
3. **Evitar que el oponente haga una conexión de 3 fichas**
   Porque tal caso, si el oponente consigue colocar una ficha en cierta posición, el oponente ganará la partida.

4. **Jugar de manera activa**
   Aunque es importante jugar de manera defensiva y bloquear al oponente como ya se ha mencionado en el punto anterior (3.), para ganar el jugador tendrá que establecer conexiones entre sus fichas.

5. **Hacer un ataque multidireccional**
   Cuando se tienen alineadas tres fichas de tal manera que pueden expandirse a una alineación de cuatro fichas en diferentes direcciones, hay más posibilidades para ganar. (Ver Figura 2)

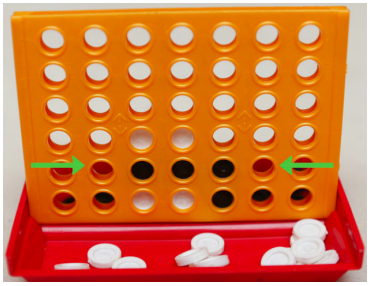

Figura 2: La conexión de las 3 fichas negras puede expandirse tanto por la izquierda como por la derecha.

6. **La trampa del 7**
   Se trata de un movimiento que consiste en posicionar las fichas de tal manera que formen un 7 (ver Figura 3). Es una buena estrategia, ya que permite alinear 4 fichas en diferentes direcciones.

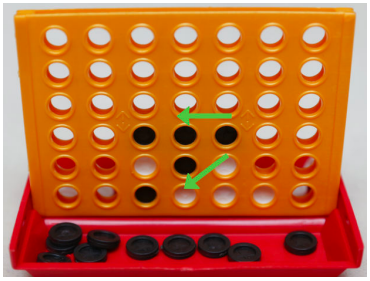

Figura 3: Tablero con fichas negras realizando la trampa del 7.

Por norma general, el primer jugador tiene más posibilidades de ganar si introduce la primera ficha en la columna central (como ya hemos mencionado en el primer punto, colocando fichas en la columna central hay más oportunidades para hacer conexiones). Si introduce la primera ficha en alguna de las columnas cercanas a la columna central, se puede llegar a un empate, mientras que si la mete en las más alejadas del centro su rival puede vencerle con mayor facilidad.

Algoritmo Minimax

Minimax es el método que usaremos para determinar cuál es el mejor de los movimientos que la computadora puede llevar a cabo en cada uno de sus respectivos turnos.

La idea principal de un algoritmo tipo minimax es muy sencilla. Se trata de que el ordenador escoja el mejor movimiento que puede hacer suponiendo que el contrincante escogerá el peor para él.

En la práctica, realizar la búsqueda completa en el árbol que expande el algoritmo minimax puede requerir de cantidades excesivas de tiempo y memoria. Para optimizar el problema, usaremos la poda del tipo alfa-beta. Con este tipo de poda evitaremos explorar nodos cuya evaluación final no va a poder superar los valores previamente obtenidos.

Para diseñar el algoritmo minimax hemos usado de referencia el que aparece en las transparencias

(tema: Búsqueda con adversario) y le hemos hecho unas pequeñas modificaciones para que funcione con nuestro juego. El resultado final es el siguiente:

Algoritmo 1: minimax

Entrada: nodo: Estado del tablero del juego actual, n: número de niveles a explorar, turno=turnomax
Salida: Nos devuelve el mejor movimiento que se puede hacer.

```
    Función minimax (nodo, n, turno)
    principio
 1: α ← −999999
 2: β ← 999999
 3: S ← sucesores(nodo)
 4: si (n = 0  ∨  S = ∅  ∨  juegoterminado(nodo)) entonces
 5:   Minimax ← evaluar(nodo)
 6:   movimiento ← nodo
 7: si no
 8:   si turno=turnomax entonces
 9:     mientras ¬parar hacer
10:       w ← seleccionar_sucesor(S)
11:       S ← S − {w}
12:       valores ← minimax(w,n-1,turnomin)
13:       si valor_minimax(valores)≥ α entonces
14:         α ← valor_minimax(valores)
15:         movimiento ← w
16:       fin si
17:       si (β ≤ α  ∨  w = ultimo(S)) entonces
18:         parar ← True
19:       fin si
20:     fin mientras
21:     Minimax ← α
22:   si no
23:     mientras ¬parar hacer
24:       w ← seleccionar_sucesor(S)
25:       S ← S − {w}
26:       valores ← minimax(w,n-1,turnomin)
27:       si valor_minimax(valores)≤ β entonces
28:         β ← valor_minimax(valores)
29:         movimiento ← w
30:       fin si
31:       si (β ≤ α  ∨  w = ultimo(S)) entonces
32:         parar ← True
33:       fin si
34:     fin mientras
35:     Minimax ← β
36:   fin si
37: fin si
38: devolver {Minimax, movimiento}
```

Dentro de la función minimax llamamos a otras funciones secundarias. Vamos a explicar brevemente lo que hacen estas funciones:

- *sucesores(nodo)*: Esta función devuelve una lista con los sucesores del nodo actual.
- *juegoterminado(nodo)*: Esta función devuelve *verdadero* cuando el juego ha terminado y *falso* en el caso contrario.

- *evaluar(nodo)*: Devuelve un valor asociado al nodo. Esta función la analizaremos con más detenimiento en el siguiente apartado.
- *selecionar_sucesor(S)*: Escoge el primer elemento de la lista S.
- *valor_minimax(valores)*: Escoge Minimax de la lista valores.
- *ultimo(S)*: Escoge el ultimo elemento de la lista S.

Algoritmos de evaluación

La función de evaluación de nodos juega un papel muy importante en el programa principal. Esta función se encarga de asignar un valor a cada nodo en relación a lo beneficiosa que es la jugada para la computadora.

La función recibe como argumento un nodo. Ese nodo representa una jugada posible y la función tiene que determinar cómo de beneficiosa es esa jugada para el ordenador. Para determinar el valor que corresponde a cada jugada, se puede diseñar la función de varias formas. Nosotros en esta práctica lo hemos hecho de dos maneras diferentes.

En la primera versión de la función evaluación, se calcula el valor de la jugada de la próxima manera:

1. Si en la jugada ya hay un cuatro en raya de la computadora, la función devuelve el valor 999999.
2. Si en cambio, en la jugada hay un cuatro en raya del oponente, la función devuelve el valor -999999.
3. Si no hay ningún cuatro en raya en la jugada, creamos un tablero (tab1) con las fichas que se han colocado hasta ahora dentro y los huecos vacíos los rellenamos con las fichas de la computadora.
4. Luego, se repite el anterior paso, pero esta vez llenando los huecos con las fichas del rival (tab2).
5. Se cuentan la cantidad de cuatro en líneas que tiene cada uno en el tablero llenado con sus fichas y se resta la cantidad de cuatro en líneas de la computadora con las del rival:

   (Cuatro en líneas de la computadora en tab1) − (Cuatro en líneas del contrincante en tab2)

El algoritmo de la primera versión de la función evaluación, que se encuentra en el programa *conecta4(v1).py*, es el siguiente:

Algoritmo 2: evaluar1

Entrada: nodo: El nodo representa una jugada. Es el tablero de juego actual con la jugada ejecutada.
Salida: Nos devuelve un valor asignado a la jugada en relación a lo beneficiosa que es la jugada para la computadora.

Función evaluar (nodo)
principio
1: contador4o $\leftarrow 0$
2: contador4x $\leftarrow 0$
3: **para** $i \in [1, 6)$ **hacer**
4: **para** $j \in [1, 7)$ **hacer**
5: **si** nodo[i][j]= "o" **entonces**
6: contador4o $\leftarrow$ contador4o $+ der(nodo, i, j, 4) + arr(nodo, i, j, 4) + diag\_der(nodo, i, j, 4) + diag\_der(nodo, i, j, 4)$
7: **si no**, **si** nodo[i][j]= "x" **entonces**
8: contador4x $\leftarrow$ contador4x $+ der(nodo, i, j, 4) + arr(nodo, i, j, 4) + diag\_der(nodo, i, j, 4) + diag\_der(nodo, i, j, 4)$
9: **fin si**
10: **fin para**
11: **fin para**
12: **si** contador4o> 0 **entonces**
13: **devolver** -999999
14: **si no**, **si** contador4X> 0 **entonces**
15: **devolver** 999999
16: **si no**
17: tab1 $\leftarrow$ llenar_o(nodo)
18: tab2 $\leftarrow$ llenar_x(nodo)
19: **para** $i \in [1, 6)$ **hacer**
20: **para** $j \in [1, 7)$ **hacer**
21: **si** nodo[i][j]= "o" **entonces**
22: contador4o $\leftarrow$ contador4o $+ der(tab1, i, j, 4) + arr(tab1, i, j, 4) + diag\_der(tab1, i, j, 4) + diag\_der(tab1, i, j, 4)$
23: **si no**, **si** nodo[i][j]= "x" **entonces**
24: contador4x $\leftarrow$ contador4x $+ der(tab2, i, j, 4) + arr(tab2, i, j, 4) + diag\_der(tab2, i, j, 4) + diag\_der(tab2, i, j, 4)$
25: **fin si**
26: **fin para**
27: **fin para**
28: **devolver** contador4x - contador4xo
29: **fin si**

Dentro de la función *evaluar1* tenemos cuatro funciones auxiliares muy importantes. Cada una de ellas comprueba a ver si hay n fichas del mismo color en una dirección concreta partiendo de la casilla (i,j):

- *der(nodo,i,j,n)*: Hacia la derecha.
- *arr(nodo,i,j,n)*: Hacia arriba.
- *diag_der(nodo,i,j,n)*: Subiendo en diagonal para la derecha.
- *diag_izq(nodo,i,j,n)*: Subiendo en diagonal hacia la izquierda.

En la segunda versión de la función evaluación, hemos contado la cantidad de fichas en línea que tiene cada jugador y hemos distinguido tres casos:

(I) Si en el tablero hay un cuatro en línea de la computadora la función de evaluación devuelve el valor 999999.

(II) Si en cambio, el cuatro en línea es del oponente, la función de evaluación devuelve el valor -999999.

(III) Si en el tablero no hay ningún cuatro en línea, la función cuenta todas las líneas de dos y tres fichas que tiene cada jugador y calcula el valor que devuelve con la siguiente fórmula:

$$100 * (\text{número de 3 en línea de la computadora}) + (\text{número de 2 en línea de la computadora})$$
$$-100 * (\text{número de 3 en línea del oponente}) - (\text{número de 2 en línea del oponente})$$

Por lo tanto, el algoritmo de la segunda versión de la función evaluación es este:

Algoritmo 3: evaluar2

Entrada: nodo: El nodo representa una jugada. Es el tablero de juego actual con la jugada ejecutada.
Salida: Nos devuelve un valor asignado a la jugada en relación a lo beneficiosa que es la jugada para la computadora.

Función evaluar (nodo)
principio
contador2o $\leftarrow$ 0
contador2x $\leftarrow$ 0
contador3o $\leftarrow$ 0
contador3x $\leftarrow$ 0
contador4o $\leftarrow$ 0
contador4x $\leftarrow$ 0
para $i \in [1, 6)$ **hacer**
  para $j \in [1, 7)$ **hacer**
    si nodo[i][j]= "o" **entonces**
      contador2o $\leftarrow$ contador2o + $der(nodo, i, j, 2)$ + $arr(nodo, i, j, 2)$ + $diag\_der(nodo, i, j, 2)$ + $diag\_der(nodo, i, j, 2)$
      contador3o $\leftarrow$ contador3o + $der(nodo, i, j, 3)$ + $arr(nodo, i, j, 3)$ + $diag\_der(nodo, i, j, 3)$ + $diag\_der(nodo, i, j, 3)$
      contador4o $\leftarrow$ contador4o + $der(nodo, i, j, 4)$ + $arr(nodo, i, j, 4)$ + $diag\_der(nodo, i, j, 4)$ + $diag\_der(nodo, i, j, 4)$
    si no, **si** nodo[i][j]= "x" **entonces**
      contador2x $\leftarrow$ contador2x + $der(nodo, i, j, 2)$ + $arr(nodo, i, j, 2)$ + $diag\_der(nodo, i, j, 2)$ + $diag\_der(nodo, i, j, 2)$
      contador3x $\leftarrow$ contador3x + $der(nodo, i, j, 3)$ + $arr(nodo, i, j, 3)$ + $diag\_der(nodo, i, j, 3)$ + $diag\_der(nodo, i, j, 3)$
      contador4x $\leftarrow$ contador4x + $der(nodo, i, j, 4)$ + $arr(nodo, i, j, 4)$ + $diag\_der(nodo, i, j, 4)$ + $diag\_der(nodo, i, j, 4)$
    fin si
  fin para
fin para
si contador4o$>$ 0 **entonces**
  devolver -999999
si no, **si** contador4x$>$ 0 **entonces**
  devolver 999999
si no
  devolver 100$*$contador3x + contador2x $-100*$contador3o $-$ contador2o
fin si

La función *evaluar2(nodo)* la usamos en el programa *conecta4(v2).py*. En el programa *conecta4(v3).py* usamos una variante mejorada de la función *evaluar2(nodo)*.

En la función *evaluar2(nodo)* tenemos en cuenta cualquier linea de dos y tres fichas que haya en el tablero, pero hay casos en los que algunas lineas de dos y tres fichas no se pueden convertir en lineas de 4, por lo tanto, no convendría contar esas lineas.

-	-	-	-	-	-	-
-	o	x	x	x	o	-

Figura 4: En este caso el tres en raya de "x" no puede convertirse en un cuatro en raya porque está rodeado de "o".

Para no tener en cuenta estos casos, hemos tenido que modificar las funciones *der(nodo,i,j,n)*, *arr(nodo,i,j,n)*, *diag_der(nodo,i,j,n)* y *diag_izq(nodo,i,j,n)*. La función *evaluar2(nodo)* sigue siendo igual que en la segunda versión. Esas pequeñas modificaciones suponen una mejora a la hora de evaluar cada jugada.

Análisis de complejidad computacional

En primer lugar analizaremos la complejidad temporal del algoritmo $minimax(nodo, n, turno)$. En este algoritmo hay llamadas a diferentes funciones: *sucesores*, *juegoterminado* y *evaluar*. Estas funciones que hemos mencionado tienen un coste temporal constante ($T(n) \in \Theta(1)$). Ya que los bucles que hay en dichas funciones (para cada ... hacer) se repiten un número finito de veces que no depende de n y además las operaciones que se realizan dentro de estos bucles tienen coste de una unidad temporal (asignaciones, operaciones aritméticas, comparaciones...). En el algoritmo $minimax(nodo, n, turno)$ se exploran n niveles del juego. Y como el jugador puede introducir la ficha en cualquiera de las 7 columnas del tablero (a menos que alguna esté llena), cada nodo tendrá k sucesores, tal que $k \in \{1, 2, ..., 7\}$. Como dentro de *minimax* hay una función de poda (poda alpha-beta), en ciertas ocasiones no se explorarán todos los sucesores de cada nodo ni todos los niveles del árbol. Por tanto, en el peor de los casos, todos los nodos que se exploren tendrán 7 sucesores y se explorarán todos los nodos de todos los niveles. En tal caso, tenemos que $T(n) \in O(7^n)$. En el mejor de los casos, en el primer sucesor el ordenador ganaría el juego y por tanto no se explorarían los demás nodos debido a la poda alpha-beta. En tal caso, tenemos que $T(n) \in \Omega(1)$.

En cuanto a la complejidad espacial, los elementos que se guardan en el algoritmo $minimax(nodo, n, turno)$: los sucesores del nodo, el nodo... son listas cuya longitud no depende de n, por tanto, solo queda mirar cuántas veces se llama recursivamente a la función *minimax*. Teniendo en cuenta el mejor y peor caso que ya hemos mencionado previamente, en el peor caso como la profundidad del árbol es de n niveles, tenemos que $S(n) \in O(n)$. En cambio, en el mejor caso, como el único nodo que se explora es el primer sucesor del primer nivel, tenemos que $S(n) \in \Omega(1)$.

Bibliografía

- https://es.wikipedia.org/wiki/Conecta_4
- https://en.wikipedia.org/wiki/Connect_Four
- http://es.wikihow.com/ganar-en-Conecta-4
- https://es.wikipedia.org/wiki/Minimax
- http://stackoverflow.com/questions/10985000/how-should-i-design-a-good-evaluation-function-for-connect-4
- http://roadtolarissa.com/connect-4-ai-how-it-works/
- http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf
- http://tromp.github.io/c4/c4.html

- https://en.wikibooks.org/wiki/LaTeX