PRÁCTICA 4

INTELIGENCIA ARTIFICIAL

Javier Martinez Rubio javier.martinezrubio@estudiante.uam.es e357532
Jorge Santisteban Rivas jorge.santisteban@estudiante.uam.es e360104

1. Introducción

Conecta 4, también conocido como 4 en Línea, es un juego de estrategia abstracta donde los contrincantes disponen de información perfecta. Las reglas del Conecta 4 son:
- Dos jugadores introducen fichas en un tablero vertical formado por seis filas y siete columnas con el objetivo de alinear cuatro consecutivas de un mismo color.
- Cada jugador dispone de 21 fichas de un color, que nosotros representaremos con un `0` o un `1`.
- 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.
- Gana la partida el primero que consiga alinear cuatro fichas consecutivas de un mismo color en horizontal, vertical o diagonal.
- Si todas las columnas están llenas pero nadie ha hecho una fila, columna o diagonal válida, hay empate.

Por norma general, el primer jugador tiene más posibilidades de ganar si introduce la primera ficha en la columna central. Si lo hace en las contiguas se puede forzar un empate, mientras que si la mete en las más alejadas del centro su rival puede vencerle con mayor facilidad.

2. Resolución teórica

Conecta4 es un juego cuyo jugador "perfecto" ya fue resuelto en 1988 por Victor Allis, donde si ambos jugadores juegan la partida perfecta el primer jugador siempre gana si juega la columna central al principio y si elige otra columna el segundo jugador siempre puede forzar un empate. Basándonos en estos resultados teóricos hemos desarrollado 3 heurísticas basadas en el mismo algoritmo.

3. Heurísticas

Dado como funciona el algoritmo Negamax nuestro principal objetivo era desarrollar una heurística que analizará cualquier tablero y le diera una puntuación, independientemente quien fuera el jugador que realizase el movimiento, Negamax ya hará que los tableros del contricante tengan puntuación negativa. Para analizar un tablero hemos tenido en cuenta dos principales parámetros:

- **Fichas en la columna central:**

Para comprobar esto simplemente hemos realizado un bucle que vaya comprobando fila a fila si tenemos una ficha del usuario que juega en la tercera columna. Por cada ficha que encontremos sumaremos 3 puntos a la puntuacion final del tablero:

```
(loop for fila from 0 below (1- (tablero-alto tablero)) do
    (setf puntuacion
          (+ puntuacion
             (if (eql (obtener-ficha tablero 3 fila) ficha-actual)
                 3 0))))
```

- **Bloques ganadores:**

En este segundo apartado hemos definido el conjunto de bloque. Un **bloque** es todas las posibles combinaciones de 4 casillas donde un jugador podría definir un movimiento ganador. Nuestro algoritmo revisa todos estos bloques y según la disposición de fichas del bloque otorgaremos una determinada puntuación:

```
  ;Contamos el numero de bloques horizontales ganadores
  (loop for fila from 0 below (1- (tablero-alto tablero)) do
    (loop for columna from 0 to 3 do
      (let* ((b1 (obtener-ficha tablero columna fila))
             (b2 (obtener-ficha tablero (+ columna 1) fila))
             (b3 (obtener-ficha tablero (+ columna 2) fila))
             (b4 (obtener-ficha tablero (+ columna 3) fila)))
        (setf puntuacion
              (+ puntuacion
                 (evaluarBloque b1 b2 b3 b4 ficha-actual ficha-oponente))))))
   ;Contamos el numero de bloques horizontales ganadores
   (loop for columna from 0 below (1- (tablero-ancho tablero)) do
     (loop for fila from 0 to 2 do
       (let* ((b1 (obtener-ficha tablero columna fila))
              (b2 (obtener-ficha tablero columna (+ 1 fila)))
              (b3 (obtener-ficha tablero columna (+ 2 fila)))
              (b4 (obtener-ficha tablero columna (+ 3 fila))))
         (setf puntuacion
               (+ puntuacion
                  (evaluarBloque b1 b2 b3 b4 ficha-actual ficha-oponente))))))
    ;Contamos el numero de diagonales ascendentes
    (loop for fila from 0 to 2 do
      (loop for columna from 0 to 3 do
        (let* ((b1 (obtener-ficha tablero columna fila))
               (b2 (obtener-ficha tablero (+ 1 columna) (+ 1 fila)))
               (b3 (obtener-ficha tablero (+ 2 columna) (+ 2 fila)))
               (b4 (obtener-ficha tablero (+ 3 columna) (+ 3 fila))))
          (setf puntuacion
                (+ puntuacion
                   (evaluarBloque b1 b2 b3 b4 ficha-actual ficha-oponente))))))

    ;Contamos el numero de diagonales descendentes
    (loop for fila from 3 to 5 do
      (loop for columna from 0 to 3 do
        (let* ((b1 (obtener-ficha tablero columna fila))
               (b2 (obtener-ficha tablero (+ 1 columna) (- fila 1)))
               (b3 (obtener-ficha tablero (+ 2 columna) (- fila 2)))
               (b4 (obtener-ficha tablero (+ 3 columna) (- fila 3))))
          (setf puntuacion
                (+ puntuacion
                   (evaluarBloque b1 b2 b3 b4 ficha-actual ficha-oponente))))))
      puntuacion))))
```

En último lugar sólo nos queda definir como evaluamos los bloques. Este es el aspecto más significativo y en el cual difieren las tres heurísticas creadas. En primer lugar en un bloque debemos de contar las 3 posibles opciones para una casilla:

- Fichas nuestras
- Fichas rival
- Fichas vacías

Para ello simplemente utilizamos 3 contadores para los 3 tipos de fichas que podemos tener:

```
(let ((fichasNuestras 0)
       (fichasVacias 0)
       (fichasRival 0)
       (score 0))
```

```
    (setf fichasNuestras
          (+ fichasNuestras
             (cond ((eql b1 ficha-actual) 1)
                   (t 0))
             (cond ((eql b2 ficha-actual) 1)
                   (t 0))
             (cond ((eql b3 ficha-actual) 1)
                   (t 0))
             (cond ((eql b4 ficha-actual) 1)
                   (t 0))
             ))
     (setf fichasRival
           (+ fichasRival
              (cond ((eql b1 ficha-oponente) 1)
                    (t 0))
              (cond ((eql b2 ficha-oponente) 1)
                    (t 0))
              (cond ((eql b3 ficha-oponente) 1)
                    (t 0))
              (cond ((eql b4 ficha-oponente) 1)
                    (t 0))
              ))
      (setf fichasVacias
            (+ fichasVacias
               (cond ((NULL b1) 1)
                     (t 0))
               (cond ((NULL b2) 1)
                     (t 0))
               (cond ((NULL b3) 1)
                     (t 0))
               (cond ((NULL b4) 1)
                     (t 0))
               ))
```

Una vez contadas las casillas implementaremos 3 posibles estrategias, donde iremos revisando las siguientes situaciones y si se cumplen les asignaremos una determinada puntuación. Las posibles situaciones en un bloque son:

- Si tenemos 4 fichas nuestras
- Si tenemos 3 fichas nuestras y 1 vacía
- Si tenemos 2 fichas nuestras y 2 vacías
- Si el rival tiene 3 fichas y una vacía

En las 3 heurísticas enviadas iremos incrementando el valor de un bloque donde el usuario esté a una ficha de finalizar la partida, incrementando así la “agresividad” a la hora de jugar de este:

Heurística Nivel 1

```
      (+ score
               (cond ((= fichasNuestras 4) 100)
                     ((and (= fichasNuestras 3) (= fichasVacias 1)) 5)
                     ((and (= fichasNuestras 2) (= fichasVacias 2)) 2)
                     (t 0))
               (if (and (= fichasRival 3) (= fichasVacias 1))
                    -4 0)))
```

Heurística Nivel 2

```
(setf score
      (+ score
         (cond ((= fichasNuestras 4) 100)
               ((and (= fichasNuestras 3) (= fichasVacias 1)) 40)
               ((and (= fichasNuestras 2) (= fichasVacias 2)) 10)
               (t 0))
         (if (and (= fichasRival 3) (= fichasVacias 1))
              -10 0)))
```

Heurística Nivel 3

```
(setf score
      (+ score
         (cond ((= fichasNuestras 4) 100)
               ((and (= fichasNuestras 3) (= fichasVacias 1)) 60)
               ((and (= fichasNuestras 2) (= fichasVacias 2)) 10)
               (t 0))
         (if (and (= fichasRival 3) (= fichasVacias 1))
              -20 0)))
```

4. Conclusiones

Antes de afrontar como preparar una heurística nos planteamos el método que íbamos a seguir. En esta posición teníamos dos opciones: o aprender a jugar de una manera solvente a Conecta4 en un tiempo récord o investigar diferentes algoritmos ya creados (desde el 1988) y desarrollar nuestras heurísticas en base a esto. Finalmente nos decantamos y realizamos una búsqueda exhastiva de cuales eran las mejores estrategias para programar al mejor jugador posible. Si hubiéramos dispuesto de un poco más de tiempo nos habría gustado desarrollar dos conceptos más. En primer lugar un **programa de entrenamiento**, de manera que las puntuaciones que otorgamos en las distintas situaciones sean números aleatorios y generar un torneo con un alto número de participantes, que se enfrentarán entre ellos y que poco a poco obtuviéramos un mejor jugador, desarrollando así la idea de **aprendizaje automático**. La segunda era explorar la idea de **even y odd threats**, desarrollada por Allis en su tésis. Él diferencia las situaciones de una posible victoria en las que la última ficha se coloca en una fila par o en una fila impar. Tras intentar su implementación y desarrollar la heurística entregada finalmente decidimos no implementar esa idea, pero no descartamos intentarlo en un futuro.