OTHELLO

Laura Elizabeth Florian Cruz
Fredy Edgar Carranza Athó
Escuela Académico Profesional de Informática, Universidad Nacional de Trujillo

RESUMEN

Othello es un juego de tablero en el que dos jugadores hacen movimientos con el objetivo de ganar con el número mayor de fichas sobre el tablero, o dejar a su oponente sin movimientos. El proyecto se baso en implementar y comprender el funcionamiento del algoritmo alfa-beta, así como desarrollar la función heurística para obtener la mejor jugada en el juego del Othello. La búsqueda tradicional se basa en contar el número de fichas para la elección el nodo de búsqueda con mayor valor; nuestra función considera además del número de fichas, la cantidad e movimientos y el número de fichas estables. Los resultados de respuesta del juego son óptimos, en su mayoría deja al oponente sin que pueda realizar ningún movimiento.

Palabras Claves: Othello, Inteligencia Artificial, Poda Alfa Beta, Heurística, Movilidad, Estabilidad.

1. INTRODUCCION

El othello es un juego de tablero en el que dos jugadores van colocando fichas alternativamente en lugares donde existe un movimiento que permite capturar piezas del adversario para cambiarlas al color que el pertenece, con el objetivo de ganar con el numero mayor de fichas sobre el tablero o dejar a su oponente sin movimientos
La mayoría de los juegos de mesa y una gran cantidad de problemas informáticos pueden resolverse mediante una adecuado modelamiento en estados y aplicar un algoritmo de búsqueda en esos estados, examinando así por anticipado, explorando en el grafo generado, para determinar cual es la mejor opción.

Este algoritmo no podría buscar entre todos los estados generados ya que demanda mucho tiempo y consume mayores recursos de memoria. Uno de los métodos mas comunes en el tema de juegos es el algoritmo de Mínimax, pero este padece de características mencionadas anteriormente; frente a ello se ha tomado en cuenta al algoritmo Poda Alfa-Beta que nos permite eliminar nodos innecesarios, valiéndose de una función heurística.

La función heurística permitirá asignarle un valor de utilidad a los nodos hijos para predecir cuan buenos o útiles son dichos nodos.

2. CLASES Y ESTRUCTURAS

Para el desarrollo de Othello hemos requerido diferentes tipos de estructuras como de clases.

Estructuras

Arreglo Entero Tablero[64]. Este arreglo de enteros representará a nuestro tablero de juegos, y contendrá 64 posiciones representando a todos los casilleros de un tablero común.

En dicha estructura la forma de representar a las fichas será de la siguiente manera. Si es un uno, representará a una ficha de color rojo, de ser cero a una ficha azul y de ser menos uno el casillero está vacío.

Clases

Clase Nodo. Representará al nodo que será manipulado en la pila, permitiendo la construcción del árbol del juego. Sus atributos representan a un dato que contendrá, así como un puntero que se dirigirá al nodo anterior y siguiente.

Clase NodoBusqueda. Permitirá ser el un nodo de propósito específico para una búsqueda, ya que tiene atributos como profundidad y costo.

Clase NodoOthello. Hijo de NodoBusqueda. Representará al nodo clave para la generación del árbol de juego. Objetos de esta clase son los que irán contenidos en el atributo dx de la clase Nodo.

Clase Pila. Facilitará la manipulación de los nodos generados, haciendo simple el proceso de poner y quitar nodos. Contendrá un método especial que permitirá eliminar todos los descendientes de un nodo específico.

Clase Jugada. Representará a una potencial jugada, conteniendo como atributos principales el número del casillero donde se puede colocar una ficha, y además un arreglo con las fichas que cambiarían de color, de realizarse dicho movimiento.

Clase ArregloJugada. Permitirá una fácil manipulación de todas las potenciales jugadas en un determinado momento del juego. Si este arreglo está vacío significa que no se puede realizar ningún movimiento.

Clase Ficha. Representará a una ficha de juego, indicando su número de casillero y su color.

Clase Juego. Será la clase donde se coloquen como métodos todos aquellos procedimientos que hagan que Othello se desarrolle de manera automática. Es decir, contiene los algoritmos que permiten determinar las jugadas válidas, generar los nodos hijos, calcular la utilidad y realizar la poda Alfa – Beta. Consideraremos siempre a la computadora como el jugador MAX y al humano como MIN, sea cual fuere el color que hayan elegido.

Clase PanelOthello. Facilitará la manipulación de las interfaces y la interacción a través de ellas.

Clase Othello. Será la clase principal donde se ejecutará el método Main que permitirá lanzar el programa.

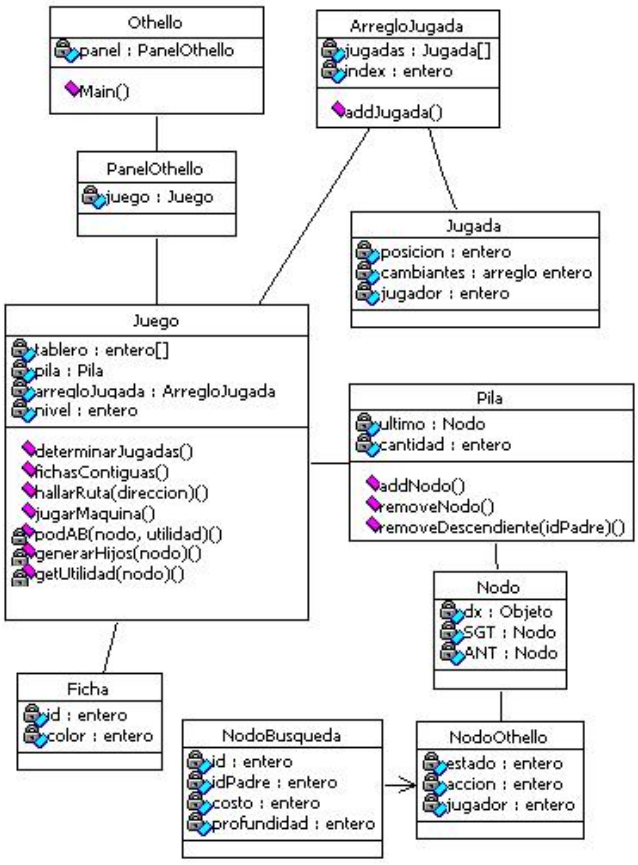

Fig. 1. ***Diagrama de clases***

3. PROCEDIMIENTOS

Para determinar jugadas validas

Una jugada valida se define por los siguientes pasos :

Recorrer el tablero y escoger las posiciones validas

Una posición valida se define como un casillero libre donde a través de algunos sus ocho vecinos a través de una ruta en línea recta se logra llegar a una ficha de color de juego actual.; con la restricción de que la ruta debe estar conformada de fichas de color contrario. [2]

Determinar Fichas cambiantes

Una vez escogidas las posiciones validas se encontraran cual de las rutas cumplen con la condición de que contengan las fichas de color diferente y terminen en una ficha de color igual al del turno.

Las direcciones consideradas para determinar las fichas cambiantes están asociadas a los siguientes valores enteros porque permite una mejor manipulación del tablero

	Arriba	Abajo
Vertical	-8	+8
Horizontal	-1	+1
Diagonal derecha	-7	+9
Diagonal izquierda	-9	+7

Tabla. 1. ***Valores para las direcciones del tablero***

El procedimiento que haya las jugadas validas encuentra las posiciones validas y las fichas cambiantes de dichas posiciones, como se observa en el siguiente algoritmo:

```
determinarJugadasValidas
ENTRADA tablero, color
SALIDA jugadasValidas
Para   i=0 hasta 64
      Si tablero[i] esta vacio
            unoVecino=determinarCambiantes(tablero, i,-8,color)
            dosVecino= determinarCambiantes (tablero, i,-7,color)
            tresVecino= determinarCambiantes (tablero, i,1,color)
            cuatroVecino= determinarCambiantes (tablero, i,9,color)
            cincoVecino= determinarCambiantes (tablero, i,8,color)
            seisVecino= determinarCambiantes (tablero, i,7,color)
            sieteVecino= determinarCambiantes (tablero, i,-1,color)
            ochoVecino= determinarCambiantes (tablero, i,-9,color)
            vecinos= unoVecino + dosVecino+ tresVecino + cuatroVecino
            + cincoVecino + seisVecino + sieteVecino +  ochoVecino
            si longitud de vecinos es diferente de 0
                  jugada [j] = vecinos + i
                  j = j+1
            fin si
```

```
        fin si
fin para
devolver jugada
```

determinarCambiantes

```
ENTRADA tablero, color, posición, dirección
SALIDA ruta
Si casillero de la posición p esta vacío tiene un vecino de dirección d de diferente color c
        Calcular las fichas contiguas en la dirección d y agregar a ruta
        Si al final de la ruta no hay ficha de color c
                Devolver vacío
        Sino
                Devolver ruta
        Fin si
Fin si
Devolver vacío
```

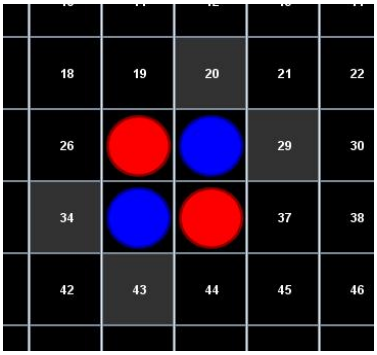

Fig. 2. ***Casilleros 20,29,34,43 (pintados de gris) son las posiciones validas***

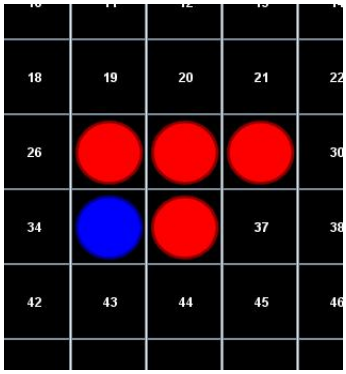

Fig. 3. ***Casilleros 27, 28 (con fichas rojo) son fichas cambiantes por efecto de seleccionar el casillero 29***

Para encontrar la función de utilidad

Para hallar la función de utilidad de una jugada se tiene en cuenta tres factores: la cantidad de fichas ocupadas por el jugador, la movilidad y la estabilidad, que se muestran a continuación:

Numero de fichas en el tablero

Se halla contando el número de fichas en el tablero de un color dado. Esto se efectúa con la finalidad de encontrar la mayor diferencia de fichas. Par ello damos el siguiente algoritmo:

determinarNumeroFichas

```
ENTRADA  tablero, color c
SALIDA numeroFichas de igual color c
Para i = 0 hasta 64
        Si tablero[i] tiene el color c
                numeroFichas =  numeroFichas + 1
        fin si
Fin para
```

Numero de movimientos posibles

La movilidad se refiere a los movimientos potenciales que podría realizar un jugador para un determinado estado del tablero así como los movimientos que su oponente pueda realizar después de su jugada.[3]

El objeto es encontrar la jugada que deje al contrincante con el menor número de movimientos, ello permitirá que la computadora gane la partida sin dejar opción a que el oponente juegue. Lo hacemos a través del siguiente algoritmo, que determina el número de movimientos posibles de un determinado.

determinarNumeroMovimientos

```
ENTRADA  tablero
SALIDA numeroMovimientosMAX , numeroMovimientosMIN
numeroMovimientos de MAX es igual al numeroMovimientos posibles de MAX en el estado actual del tablero
Por cada movimiento m  posible de MAX
        numeroMovimientos de MIN es igual a la sumatoria de numeroMovimientos posibles de MIN en el estado que genero m
Fin
```

Numero de fichas estables

La estabilidad está determinada por la posibilidad de que una ficha cambie su color en las siguientes jugadas. La posición que indica a la ficha más estable es la posición de las esquinas y junto con ellas en un menor grado de estabilidad las posiciones de los bordes del tablero.[3]

Se considera al tablero con valoraciones en cada casilla de la matriz, donde las esquinas se valoran con 120 puntos, y la función protege laterales da valores -40 o -60 a las casillas que ponen en peligro las esquinas o los laterales del tablero.

120	*-40*	*20*	*5*	*5*	*20*	*-40*	*120*
-40	*-60*	*-5*	*-5*	*-5*	*-5*	*-60*	*-40*
20	*-5*	*15*	*3*	*3*	*15*	*-5*	*20*
5	*-5,*	*3*	*3*	*3*	*3*	*-5,*	*5*
5	*-5,*	*3*	*3*	*3*	*3*	*-5,*	*5*
20	*-5*	*15*	*3*	*3*	*15*	*-5*	*20*
-40	*-60*	*-5*	*-5*	*-5*	*-5*	*-60*	*-40*
120	*-40*	*20*	*5*	*5*	*20*	*-40*	*120*

Tabla. 2. ***Pesos en el tablero***

calcularFichasEstables
ENTRADA tablero, color c
SALIDA estabilidad de igual color c
Para i = 0 hasta 64
Si tablero[i] tiene el color c
estabilidad = estabilidad + pesos[i]
fin si
Fin para

3.2.4. Determinar la utilidad de un nodo

Considerando el número de fichas, la movilidad y la estabilidad, se describe la utilidad como la suma de la diferencia de cantidad de fichas en el tablero del jugador máquina y la del oponente; la diferencia de la movilidad del jugador y del oponente y la diferencia de la estabilidad entre los mismos. Cada factor tiene una ponderación que va de acuerdo a la importancia de su función.[1]

funcionUtilidad=
((nroFichasMax - nroFichasMin) *1) +
((nroMovimientosMax-nroMovimientosMin)*10) +
((nroFichasEstablesMax-nroFichasEstablesMin)*50)

Algoritmo de poda Alfa-Beta

Este algoritmo será aplicado después de que haya jugado el oponente de MAX. Se determinarán las posibles jugadas que pueda hacer MAX en ese momento. A cada una de ellas se le aplicará el algoritmo poda Alfa-Beta, para determinar su valor de utilidad. Se escogerá el estado que tenga mayor utilidad.

Por cada estado posible de MAX, se comenzarán a generar los hijos de dicho estado a través de la aplicación de las acciones que no son más que las jugadas válidas. Al haber generado estos hijos se deberá a llegar a un límite máximo que será indicado por el nivel del juego. Al llegar a este nivel, el algoritmo cesará de generar hijos y considerará a estos estados como nodos hoja. Estos nodos hoja, serán considerados para evaluarlos y a través de la función heurística explicada, determinar su valor de utilidad.

podaAB
ENTRADA nodoOthello n ,utilidadPadre
SALIDA utilidad de n
Si el jugador es MAX utilidad = -99999
sino utilidad = +99999
fin si
si profundidad = nivel devolver la utilidad de n
sino
generar hijos de n
para cada hijo h de n
utilidad de h = podaAB(ultimo hijo en pila generado, utilidad
si jugador n es MAX
si utilidad de h > utilidad
utilidad = utilidadH;
si utilidad > utilidadP
return utilidad;
fin si
fin si
si jugador n es MIN
si utilidad de h > utilidad
utilidad = utilidadH;
si utilidad < utilidadP
return utilidad;
fin si
Fin si
Fin si
Fin para
Fin si
Retornar utilidad

4. CONCLUSIONES

- El othello es un juego interesante en el que no solo se gana con el mayor número de fichas sobre el tablero sino también dejando al ponente sin mas jugadas a su favor.

- Aplicar un algoritmo de búsqueda demanda tiempo y consume mucho recursos, en cambio el algoritmo de Poda Alfa-Beta minimiza el valor de generara estados innecesarios ahorrando así los recursos de la maquina.

- El algoritmo de Poda Alfa-Beta se vale de una función heurística que nos ayudara a decidir nodos que nos permitan dejar ala oponente en fracaso.

- La mejor opción para determinar la heurística es considerando la movilidad y la estabilidad y no con el mayor de número de fichas.

5. REFERENCIAS

[1] M. Pagola Barrio, “Resultados y Resumen del Campeonato de Otelo,” *www.ayc.unavarra.es/miguel.pagola/Resumen%20Campeonato%20Otelo.pdf*, Universidad Pública de Navarra, España, pp. 1-10, Noviembre 2006

[2] LeagueUSA, “The Game Of Othello” http://www.site-constructor.com/othello/index.html 2001

[3] Mikael Møller Rasmussen, “Othello” http://www.tjhsst.edu/~rlatimer/assignments2004/othello.html