UNIVERSIDAD SIMÓN BOLÍVAR

Departamento de Ciencia de la Información

CI-5437 Inteligencia Artificial

Profesor: Blai Bonet

PROYECTO 2

Elaborado Por:

Edward Fernández 10-11121

Stefani Castellanos 11-11394

Jirlfe Oropeza

Sartenejas, Noviembre 2016

PLANTEAMIENTO DEL PROBLEMA

Othello es un juego de mesa en el que dos jugadores (blancos y negros) colocan sus fichas en un tablero (usualmente 8x8) con la intención de cambiar de color las piezas del oponente y así, obtener una mayor cantidad de fichas en el tablero de su color. Para "voltear" alguna ficha del oponente, estas debe quedar atrapadas entre dos del color del jugador. Gana quien al final tenga más fichas de su color.

Este juego ha sido ampliamente estudiado en el campo de Inteligencia Artificial, es por esta razón que en este proyecto se desea implementar y evaluar los algoritmos de resolución de árboles de juego *minimax*, *negamax* sin poda y con poda alpha-beta, *scout* y *negascout*, para una versión reducida de Othello, con un tablero 6x6 y ofrecer resultados sobre el comportamiento de dichos algoritmos.

El tablero del juego está representado en 32 bits (cada posición es un bit) aprovechando el hecho de que las posiciones del centro del tablero siempre están ocupadas, ya que el juego inicia con estas llenas. Las posiciones del tablero están enumeradas de la siguiente manera:

4	5	6	7	8	9
10	11	12	13	14	15
16	17	0	1	18	19
20	21	2	3	22	23
24	25	26	27	28	29
30	31	32	33	34	35

En donde se completa con -1 las posiciones que no pertenecen al tablero. Adicionalmente se tienen los siguientes atributos:

- *t_* : para modelar los colores del centro del tablero, ya que estos pueden cambiar mas no desocuparse.
- free_: representa las posiciones libres del tablero. 0 si está desocupada, 1 si está ocupada.
- pos_: representa el color de la posición del tablero. 0 si es negra, 1 si es blanca.

Un estudio previo sobre este juego arrojó la solución para esta versión reducida, si ambos jugadores realizan las jugadas óptimas ganan los blancos (segundo jugador) por una diferencia de 4 fichas, es decir, 16 a 20. Esta observación nos permite conocer cuales son las jugadas en la variación principal y evaluar los algoritmos desde estos estados en los que sólo se recorre un sub-árbol del problema, ya que evaluarlos desde el inicio del juego llevaría demasiado tiempo.

ACTIVIDAD 1

Enunciado: *"Completar y probar la representación del juego."*

Para completar la representación del juego (Otelo 6x6), se procedió a trabajar en el archivo facilitado *'othello_cut.h'*. En este, se observó que las funciones *'move(bool color, int pos)'* y '*outflank(bool color, int pos)'* no realizaban el chequeo correspondiente a las diagonales.

La verificación de las diagonales se logró utilizando las matrices *'dia1[][7]'* y *'dia2[][7]'*.

ACTIVIDAD 2

Enunciado: "*Implementar los algoritmos:*

- minmax/maxmin doblemente recursivo para calcular el valor del juego
- versión negamax de minmax/maxmin
- versión negamax de minmax/maxmin con poda alpha-beta
- scout
- negascout = negamax con poda alpha-beta + scout"

Cada uno de los algoritmos antes mencionados se tuvo que adaptar al juego *'Otelo 6x6'*, dicha modificaciones corresponden a los turnos de los jugadores (blancos y negros) y el *'paso'* de un jugador que no puede realizar una jugada válida.

La implementación se encuentra en el archivo *'main.cc.'*

ACTIVIDAD 3

Enunciado: *"Los algoritmos deben ser evaluados sobre el juego de othello a lo largo de la variación principal. El valor del juego es -4 y, por lo tanto, todo tablero a lo largo de la variación principal debe evaluar a dicho valor.*

*La evaluación se hace empezando sobre el tablero terminal y subiendo sucesivamente sobre la variación principal. El *mejor* algoritmo será aquel que pueda llegar "más" lejos sobre la variación principal."*

RESULTADOS

A continuación, presentamos los resultados de las corridas de cada uno de los algoritmos usando un límite de 10 min, en un procesador Intel Core 2 Duo con 4GB de memoria RAM:

Al ser evaluado cada uno de los algoritmos implementados sobre el juego Otelo a lo largo de la variación principal se obtuvo como resultado el valor *'-4'*, logrando mostrar así que el ganador será el que posea las fichas blancas. Adicionalmente, se obtuvieron resultados con respecto a los nodos generados, nodos expandidos, el tiempo tomado en la profundidad correspondiente y en generar los nodos correspondientes. De estos resultados se puede concluir que:

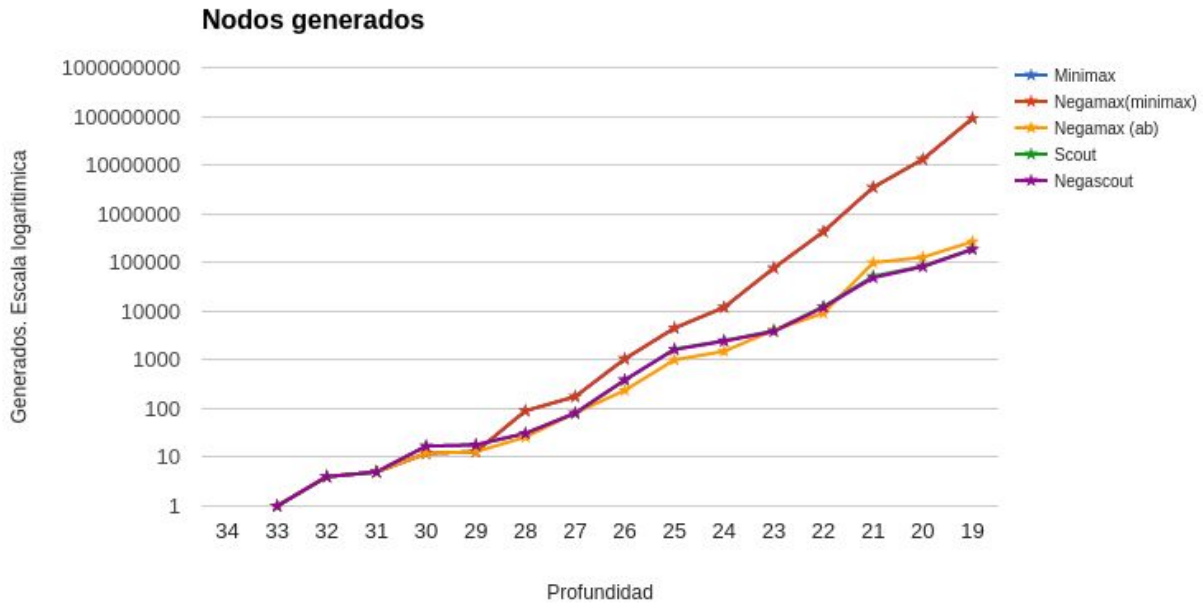
1. **Con respecto a los tiempos:**

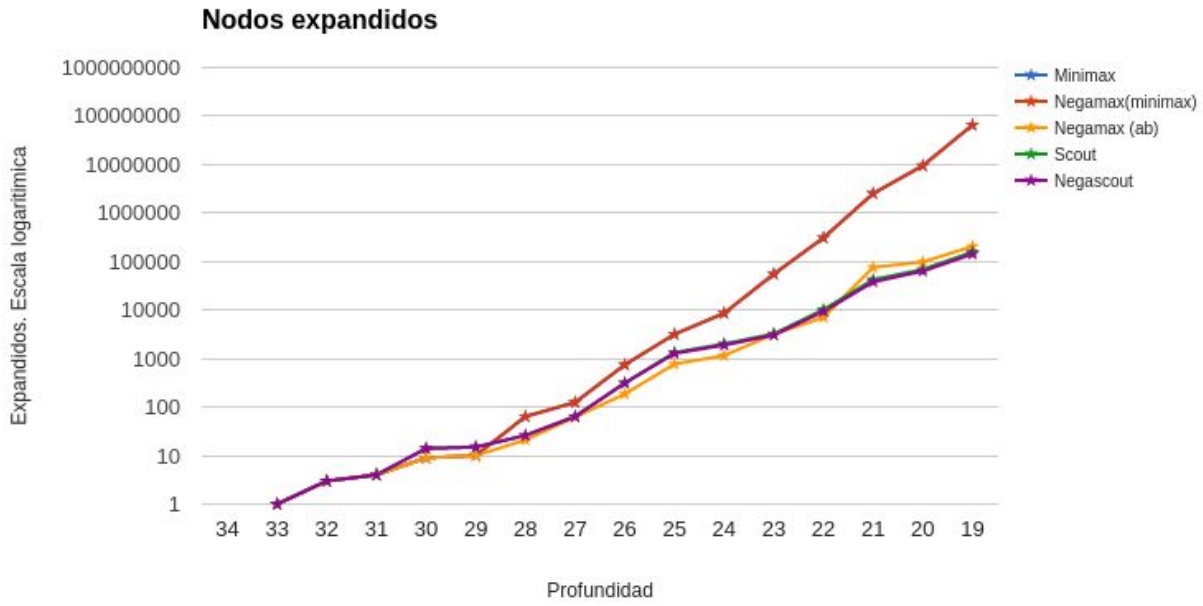
    - Los algoritmos *'Minmax'*, *'Scout'* y la versión *'Negamax'* de *'Minmax'* con poda alpha-beta reportaron, en su primera iteración, *'-nan'* en la cantidad de nodos generados por segundos. Esto se debe a que la función para calcular el tiempo perteneciente a la librería *'utils.h'* no es suficientemente precisa o la

computadora no logra “dar un *tic”* del reloj y en esta primera iteración no se generan ningún nodo, reportando 0/0, lo cual es *nan*.

- Para las siguientes iteraciones donde se obtiene tiempo infinito (inf), se debe a que los tiempos son muy insignificante y al hacer la división correspondiente se obtiene (número de nodos) / 0.
- Negascout tarda menos tiempo, en líneas generales, que el resto de los algoritmos. Minmax y Negamax sin poda utilizan mucho más tiempo que los demás debido a que explorar nodos innecesarios.

2. Con respecto a los nodos:





Nota: Minimax y Negamax tiene la misma cantidad de nodos generados puesto que son implementaciones diferentes del mismo algoritmo.

Resulta evidente que los algoritmos que no hacen poda de ningún tipo la cantidad de nodos que generan y exploran crecen exponencialmente en la profundidad, mientras que Negamax con poda alpha-beta reduce estos valores en gran medida. Los algoritmos de Scout y Negascout teóricamente exploran menos nodos, sin embargo, la poda que ellos realizan depende de cierta forma del orden de los hijos, pues espera que el mejor hijo de un estado en particular sea el primero generado y este no es siempre el caso, sin embargo, llegan a una mayor profundidad que cualquiera de los anteriores en los diez minutos.

De todos los algoritmos implementados y utilizados, se puede decir que el más exitoso fue ‘Negascout’ a pesar que ‘Scout’ y la versión ‘Negamax’ de ‘Minmax’ con poda alpha-beta llegaron hasta la misma profundidad de la variación principal (12).

Orden de éxito: Negascout, Scout, la versión ‘Negamax’ de ‘Minmax’ con poda alpha-beta, Minmax-Maxmin y la versión ‘Negamax’ de ‘Minmax’ sin poda.

Negascout tuvo mayor éxito sobre los otros algoritmos ya que este posee un factor de ramificación mucho menor debido a que él toma un valor ‘v’ como referencia y con este verifica si existe una manera más eficiente para conocer una segunda rama que me permita tener un valor mayor al ‘v’ dado, si no existe no es necesario realizar la búsqueda en dicho *branch*, por lo tanto no recorre nodos innecesarios, es decir, que no sea la jugada más óptima; Scout utiliza esta misma política, sin embargo Negascout también realiza poda alpha-beta lo que le permite explorar menos nodos además en el tiempo límite (10 minutos) logra explorar desde la profundidad 12 y Scout llega a la 13.

Luego le sigue el de ‘Alpha-beta pruning’ ya que sigue, básicamente, el comportamiento de búsqueda ‘inteligente’ utilizados por los humanos cuando el conocimiento del dominio no está involucrado. La técnica de poda alfa-beta trata de eliminar partes grandes del árbol, aplicándolo a un árbol Minimax estándar, de forma que se devuelva el mismo movimiento que devolvería este, gracias a que la poda de dichas ramas no influye en la decisión final.

ANEXOS

Minmax						
Profundidad	Jugador	Valor	Expandidos	Generados	Segundos	Generados/ segundos
34	White	-4	0	0	0.0000	-nan
33	Black	-4	1	1	0.0000	inf
32	White	-4	3	4	0.0000	inf
31	Black	-4	4	5	0.0000	inf
30	White	-4	9	12	0.0000	inf
29	Black	-4	10	13	0.0000	inf
28	White	-4	64	90	0.0000	inf
27	Black	-4	125	176	0.0000	inf
26	White	-4	744	1048	0.0033	3.19E+05
25	Black	-4	3168	4497	0.0038	1.17E+06
24	White	-4	8597	11977	0.0053	2.24E+06
23	Black	-4	55127	76825	0.0397	1.94E+06
22	White	-4	308479	428401	0.2167	1.98E+06
21	Black	-4	2525249	3478734	1.7525	1.99E+06
20	White	-4	9459570	13078932	6.5308	2.00E+06
19	Black	-4	65121519	90647894	50.3959	1.80E+06

Negamax(versión minimax)						
Profundidad	Jugador	Valor	Expandidos	Generados	Segundos	Generados/segundos
34	White	-4	0	0	0.0000	-nan
33	Black	-4	1	1	0.0000	inf
32	White	-4	3	4	0.0000	inf
31	Black	-4	4	5	0.0000	inf
30	White	-4	9	12	0.0000	inf
29	Black	-4	10	13	0.0000	inf
28	White	-4	64	90	0.0000	inf
27	Black	-4	125	176	0.0000	inf
26	White	-4	744	1048	0.0000	inf
25	Black	-4	3168	4497	0.0025	1.83E+06
24	White	-4	8597	11977	0.0046	2.59E+06
23	Black	-4	55127	76825	0.0377	2.04E+06
22	White	-4	308479	428401	0.2093	2.05E+06
21	Black	-4	2525249	3478734	1.7022	2.04E+06
20	White	-4	9459570	13078932	6.3345	2.06E+06
19	Black	-4	65121519	90647894	49.7084	1.82E+06

Negamax(versión alpha-beta)						
Profundidad	Jugador	Valor	Expandidos	Generados	Segundos	Generados/ segundos
34	White	-4	0	0	0.0000	-nan
33	Black	-4	1	1	0.0000	inf
32	White	-4	3	4	0.0000	inf
31	Black	-4	4	5	0.0000	inf
30	White	-4	9	12	0.0000	inf
29	Black	-4	10	13	0.0000	inf
28	White	-4	21	26	0.0000	inf
27	Black	-4	62	81	0.0000	inf
26	White	-4	186	237	0.0000	inf
25	Black	-4	769	1002	0.0000	inf
24	White	-4	1152	1501	0.0000	inf
23	Black	-4	3168	4067	0.0030	1.37E+06
22	White	-4	7031	9129	0.0044	2.06E+06
21	Black	-4	76021	98754	0.0545	1.81E+06
20	White	-4	98129	127643	0.0694	1.84E+06
19	Black	-4	205017	267603	0.1457	1.84E+06
18	White	-4	960343	1259429	0.6905	1.82E+06
17	Black	-4	1549785	2031923	1.1208	1.81E+06
16	White	-4	22325108	29501797	16.1737	1.82E+06
15	Black	-4	32949019	43574642	27.2571	1.60E+06
14	White	-4	82016158	107642870	70.4164	1.53E+06

13	Black	-4	315074162	415909955	273.3870	1.52E+06

Scout						
Profundidad	Jugador	Valor	Expandidos	Generados	Segundos	Generados/segundos
34	White	-4	0	0	0.0000	-nan
33	Black	-4	1	1	0.0000	inf
32	White	-4	3	4	0.0000	inf
31	Black	-4	4	5	0.0000	inf
30	White	-4	14	17	0.0000	inf
29	Black	-4	15	18	0.0000	inf
28	White	-4	26	31	0.0000	inf
27	Black	-4	64	80	0.0000	inf
26	White	-4	314	381	0.0000	inf
25	Black	-4	1334	1668	0.0000	inf
24	White	-4	2011	2471	0.0031	8.04E+05
23	Black	-4	3232	3950	0.0011	3.48E+06
22	White	-4	10214	12380	0.0075	1.64E+06
21	Black	-4	42358	51893	0.0299	1.74E+06
20	White	-4	68853	84201	0.0479	1.76E+06
19	Black	-4	157458	189156	0.1095	1.73E+06
18	White	-4	497954	607734	0.3498	1.74E+06
17	Black	-4	911296	1105247	0.6378	1.73E+06

16	White	-4	6096169	7673867	4.2534	1.80E+06
15	Black	-4	23572285	29625133	16.8908	1.75E+06
14	White	-4	57114374	72010379	46.0682	1.56E+06
13	Black	-4	211427675	269748618	178.4200	1.51E+06

Negascout						
Profundidad	Jugador	Valor	Expandidos	Generados	Segundos	Generados/ segundos
34	White	-4	0	0	0.0000	-nan
33	Black	-4	1	1	0.0000	inf
32	White	-4	3	4	0.0000	inf
31	Black	-4	4	5	0.0000	inf
30	White	-4	14	17	0.0000	inf
29	Black	-4	15	18	0.0000	inf
28	White	-4	26	31	0.0000	inf
27	Black	-4	64	81	0.0000	inf
26	White	-4	312	389	0.0000	inf
25	Black	-4	1275	1631	0.0000	inf
24	White	-4	1894	2421	0.0000	inf
23	Black	-4	3051	3843	0.0078	4.95E+05
22	White	-4	9329	11979	0.0068	1.76E+06
21	Black	-4	37988	48477	0.0276	1.76E+06
20	White	-4	63570	81631	0.0419	1.95E+06

19	Black	-4	142595	184144	0.1036	1.78E+06
18	White	-4	466161	605947	0.3417	1.77E+06
17	Black	-4	870050	1134288	0.6349	1.79E+06
16	White	-4	5518091	7221811	4.0165	1.80E+06
15	Black	-4	19705373	25831374	14.3319	1.80E+06
14	White	-4	47600678	62051335	34.3540	1.81E+06
13	Black	-4	185297022	242584981	130.5130	1.86E+06
12	White	-4	477003110	623011942	373.0960	1.67E+06