UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

FACULTAD DE CIENCIAS EXACTAS

INGENIERIA DE SISTEMAS

Trabajo práctico especial
Análisis y Diseño de Algoritmos II

Juego de Inteligencia Artificial Othello

ALUMNOS: BONGIORNO, Emilio
SANTAMARINA, Esteban

Docentes: Profesor asociado: DEA Liliana Favre
Profesor adjunto: Ing. Laura Felice

1 Resumen:

Este trabajo fue desarrollado como proyecto final para la materia “Análisis y Diseño de Algoritmos II” correspondiente a la carrera de Ingeniería de Sistemas que rige en la Facultad de Ciencias Exactas de la UNICEN. El objetivo del mismo consistió en la elaboración del juego Othello en el que puedan participar dos jugadores, y donde también exista la posibilidad de jugar contra el ordenador en diferentes dificultades. Para ello fue necesario desarrollar un algoritmo basado en la estrategia Minimax que permita obtener movimientos inteligentes para un jugador automático en un tiempo computacional razonable.

2 Introducción:

El juego Othello consiste en un tablero cuadrado de 64 casillas y la misma cantidad de fichas con dos caras: una negra y otra blanca. Al inicio de la partida se sitúan cuatro fichas en el centro del tablero como se muestra en la imagen 1.

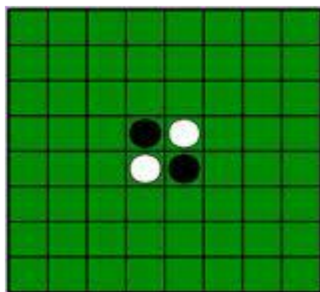

Imagen 1: Comienzo del juego

Comenzando por el jugador que utiliza las fichas de color negro, los jugadores deben colocar en sus respectivos turnos una ficha de su color en las posiciones válidas. Estas corresponden a los casilleros vacíos en los que dicho color pueda “comer” por lo menos una ficha del enemigo. La forma de capturar las fichas es colocando una propia de tal manera que queden situadas fichas del enemigo en forma horizontal, vertical o diagonal entre la ficha recién colocada y otra propia que ya esté situada en el tablero. En este caso, estas fichas que quedaron “encerradas” se dan vuelta pasando a ser ahora del color del jugador que realizó el movimiento.
Como sólo se permite poner fichas en las posiciones que no han sido ocupadas, cada partida consta de un máximo de 60 turnos (64 casillas del tablero menos las 4 ocupadas al inicio). En caso que un jugador no pueda capturar ninguna ficha, debe ceder el turno al oponente. El juego llega a su fin cuando:
1- Son ocupadas las sesenta y cuatro casillas del tablero por fichas.
2- Ninguno de los dos jugadores puede realizar una jugada y ambos deben pasar su turno.
En ambos casos la partida da por finalizada, y resulta ganador el jugador que posea más fichas con su color sobre el tablero. En caso que los dos jugadores tengan la misma cantidad de fichas, se da como resultado un empate.

Para desarrollar una plataforma que permita jugar con dos jugadores humanos simplemente se necesita mantener un control de los turnos y de las jugadas válidas para

cada uno. Pero el problema comienza cuando uno de los jugadores es la computadora, y esta debe elegir de forma inteligente un movimiento válido cada vez que es su turno. Entonces surge la necesidad de implementar un algoritmo basado en Minimax, en el cual se analizan los futuros movimientos posibles tanto del jugador manual como el automático, para poder decidir que jugada le traerá mayor beneficio en un largo plazo.

3 Algoritmo Minimax:

3.1 Estrategia Minimax

En general, Minimax es un método de decisión para minimizar la pérdida máxima esperada en juegos multijugador. Consiste en elegir el movimiento que más beneficia a uno mismo, suponiendo que el adversario también va a escoger también el más conveniente para él.
El método se divide en tres pasos:
1- Generar el árbol de juego alternando movimientos de ambos jugadores y asignar a cada movimiento el valor apropiado.
2- Calcular la función de utilidad de cada nodo final, recorriendo recursivamente los nodos hasta el nodo inicial.
3- Elegir como jugada a realizar aquel primer movimiento que conduce al nodo final con mayor función de utilidad.

3.2 Minimax en Othello

En la implementación general del método Minimax se pueden definir constantes MAX que representa al jugador automático y MIN al contrario. Particularmente, en Othello es posible manejar los turnos de cada jugador según el color de sus fichas. En caso que toque jugar a las blancas será el turno de 1, y en caso que toque jugar a las negras será el turno de 2, alternando dichos números en cada jugada. De esta forma, es posible crear un árbol en el cual cada nodo tendrá una configuración del tablero distinta, y donde sus hijos serán todas las posibles jugadas que puede realizar el jugador que tenga el turno. La imagen 2 muestra un ejemplo de una parte del árbol generado en el comienzo de una partida.

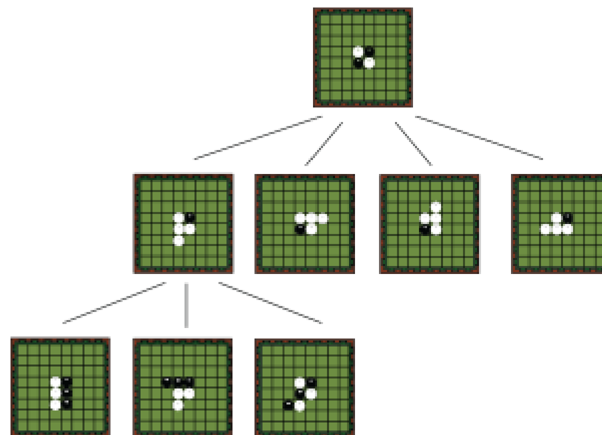

Imagen 2: Ejemplo de árbol de juego

Para valorar los distintos movimientos se tienen en cuenta la cantidad de fichas de cada color que se encuentran en el tablero. Cada ficha blanca representa una unidad negativa mientras que cada ficha negra se valora positivamente. Al final de las comprobaciones, la mejor posición para colocar la ficha del jugador automático será aquella que a la larga de la mayor puntuación en caso que juegue con las negras, o la menor puntuación en caso que juegue con las blancas.
Al expandir el árbol con una profundidad considerable, no sólo se puede conseguir el movimiento que traerá un mayor beneficio, sino que también se considera aquel que le traerá más prejuicios al contrincante. Con este método es posible saber la estrategia que va a seguir el oponente, de forma que sepamos qué ocurría si se realiza un movimiento en una casilla del tablero. A esto se lo llama **empatía**, uno de los aspectos más difíciles de controlar por un jugador humano, ya que no es fácil predecir todas las posibles combinaciones que se pueden dar, y menos cuando se intenta observar mayor cantidad de turnos. Sin embargo, el control de dicho aspecto es más sencillo para un jugador automático, ya que una computadora tiene muchas más facilidades para este cálculo.

3.3 Pseudocódigo de Minimax

```
Minimax(color, prof, punt_elegida, mov_elegido, tablero, p_max)
Si prof == p_max
        punt_elegida = ganancia(tablero)
Sino
        generar_movimientos(color)
         Si no_movimientos
              punt_elegida = ganancia(tablero)
         Sino
              Mejor_punt = infinito;
              Mientras haya movimientos
                  aux = nuevo_tablero
                  Realizar_movimiento(color,movimiento, aux)
                   Minimax(enemigo,prof+1,puntuacion,movimiento,aux,p_max)
                  Si color == 1
                       Si puntuacion < mejor_punt
                          mejor_punt = puntuacion
                          mejor_mov = movimiento
                   Sino
                       Si puntuacion > mejor_punt
                          mejor_punt = puntuacion
                          mejor_mov = movimiento
             Terminar Mientras
        mov_elegido = movimiento
        punt_elegida = puntuación
```

Como es un algoritmo recursivo, debe contar con una condición de corte. En este caso, la recursión finaliza cuando no hay más movimientos por realizar, o cuando se llega a la profundidad máxima establecida (p_max).
En ambos casos, se modifica la variable punt_elejida que se pasa como parámetro, la cual contendrá la "ganancia" en el tablero para la disposición de las fichas en ese momento. Esta función de utilidad se calcula como se indicó previamente, sumando las fichas del jugador automático y restando las del jugador manual.
Si no se llega a una hoja, se generan todos los hijos del nodo, que serán aquellas configuraciones donde el color al que le toca jugar pueda colocar su ficha y realizar la

jugada. Por cada una de las posibles jugadas, se realiza el movimiento en un tablero auxiliar, y se invoca nuevamente a la función Minimax con el color del adversario. Este llamado modificará las variables puntuación y movimiento, las cuales tendrán almacenadas la puntuación que mayor beneficio le dio al adversario, y el movimiento que llevó a obtener dicha puntuación.
Por cada una de las ramas que se explore desde un nodo, se compara el puntaje obtenido con la mejor puntuación obtenida hasta el momento. En caso de estar en el turno de las blancas, si la puntuación es menor a la más pequeña obtenida hasta el momento, esta pasa a ser la nueva puntuación más pequeña, y en caso de ser el turno de las negras, se comporta análogamente guardando la puntuación más grande en caso que el nuevo puntaje obtenido sea mayor. Lo que se hace es **"Minimizar"** cuando deben jugar las blancas, quedándose con el resultado más pequeño, y **"Maximizar"** cuando el turno es de las negras, eligiendo como jugada la que le de un mayor resultado.
La imagen 3 ilustra lo recientemente explicado, con una profundidad de hasta cuatro movimientos.

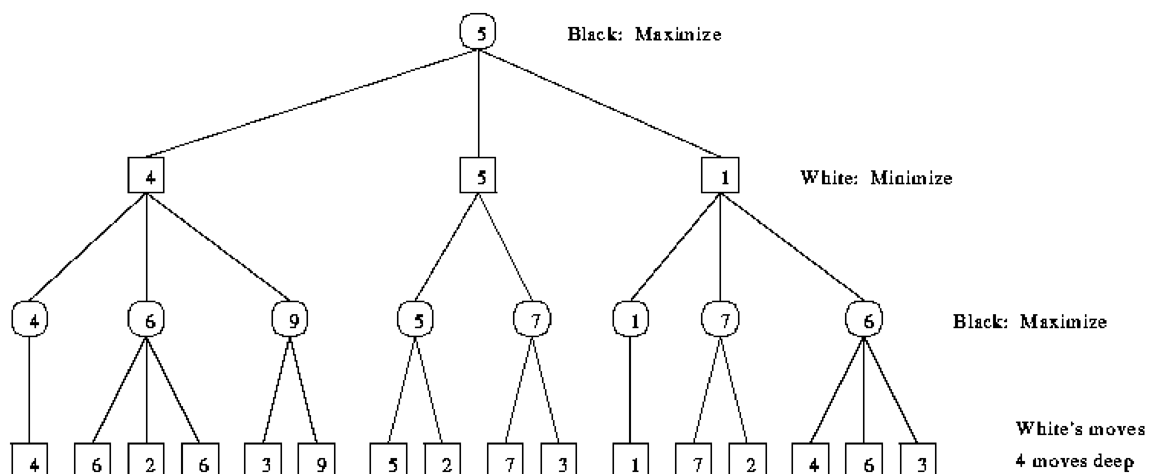

Imagen 3: Ejemplo de Minimax, las negras maximizan y las blancas minimizan

La variable de "mejor puntaje" debe comenzar valiendo una cifra que represente el infinito, es decir, un número que la función de utilidad no pueda alcanzar en ningún momento. Esto es necesario ya que no siempre hay dos o más movimientos posibles para elegir, y en caso de que sea sólo una la posibilidad, el algoritmo debe tenerlo en cuenta. De esta forma se consigue que la comparación de las variables "puntuacion" y "mejor_punt" siempre sea verdadera en la primera iteración, ya que "puntuacion" estará inicializada en un valor muy pequeño en caso que sea el turno de las blancas, y en un valor muy grande en caso que le toque jugar a las negras.

En la implementación, en caso que no existan movimientos posibles para un color, se corrobora si es posible realizar una jugada con el otro color, ya que si un jugador no puede jugar, de todas formas puede existir la posibilidad de que el oponente si pueda. En caso que exista una jugada para el adversario, el procedimiento dentro del recorrido de los hijos es el mismo, sólo que esta vez se realiza el movimiento con el color "enemigo" en dicho paso, y se invoca a la función recursiva con el mismo color que se recibió como parámetro.
Con esta inclusión se puede llegar más profundo en los casos donde un jugador deba ceder el turno pero el otro pueda seguir jugando, y a su vez mejora la decisión del jugador automático principalmente en las últimas jugadas, ya que en esas instancias es más común tener que ceder el turno.

Esta implementación no fue incluida en el pseudocódigo para que sea más comprensible su interpretación.

4 Funciones de utilidad:

Anteriormente se mencionó que para calcular la valoración de una configuración del tablero se sumaban las fichas negras y se restaban las blancas. Esta es una de las funciones de utilidad, cuyo pseudocódigo puede resumirse como el siguiente:

```
ganancia1(Tablero)
resultado = 0;
Por cada casillero de Tablero
        Si casillero == negro
                resultado = resultado+1
        Sino Si casillero == blanco
                resultado = resultado-1
Devolver resultado
```

Ya se mencionó que el jugador automático implementa la empatía mediante las comprobaciones recursivas de turnos sucesivos, pero no es la única consideración a tener en cuenta.
El comportamiento del algoritmo con esta función de utilidad es el de intentar capturar la mayor cantidad de fichas posibles, teniendo en cuenta también de perder la menor cantidad de fichas propias.
El grado de empatía que se aplica en este caso es suficiente para que un jugador novato pueda vencer al ordenador si se aplica una profundidad no muy grande en el algoritmo Minimax. Sin embargo, al expandir el árbol a profundidades mayores que seis (tres turnos para cada oponente), la computadora puede tardar un tiempo en dar una respuesta, una espera que no es muy deseable para el humano en un juego. Por ello resultó necesario implementar una función de utilidad más eficiente que consiga resultados inteligentes con profundidades no muy grandes.

Para esta nueva función se tienen en cuenta otras estrategias del juego que van más allá de la cantidad de fichas con las que se termina en las hojas del árbol.
Una de ellas es la **posesión de las esquinas**. Estas casillas son las más importantes, ya que una vez que se coloque una ficha en una de esas posiciones, no puede ser capturada. A demás, permiten que se puedan situar fichas del mismo color alrededor por los laterales consiguiendo que tampoco puedan ser capturadas, ya que tienen un lado completamente protegido. Si bien priorizar mucho la captura de una esquina puede llegar a traer prejuicios, es una casilla que debe tener mucha mayor consideración que las demás.
Esto lleva al jugador a ser precavido con las casillas limítrofes a la esquina. En particular, las casillas adyacentes en diagonal, denominadas **Casillas X**, son las más peligrosas de todas. Colocar una ficha propia en cualquiera de esas posiciones le da una gran probabilidad al rival de apoderarse de la esquina. Las dos casillas en los laterales de las esquinas, llamadas **Casillas C**, también son consideradas peligrosas aunque en menor medida debido a que pueden servir como punto de apoyo por ubicarse en los laterales.
Las casillas limítrofes a las Casillas C en los laterales en cambio, suelen traer mayores beneficios, ya que pueden provocar que el adversario coloque una ficha en una Casilla

C que luego permita apoderarse de la esquina. A demás, el hecho de ubicarse en el lateral permite que sea una ficha más difícil de capturar. Es por esto que también colocar una ficha en las casillas adyacentes a estas sobre el lateral (los cuatro pares de casillas en el medio del tablero sobre los bordes) es a su vez beneficioso aunque en menor medida.
Con estas consideraciones fue posible implementar una nueva función de utilidad, la cual aumenta considerablemente la complejidad de vencer al ordenador que la versión donde sólo se consideran las fichas giradas y no el lugar donde queden situadas. El pseudocódigo de la misma es el siguiente:

```
ganancia2(Tablero)
int resultado = 0;
Por cada casillero de Tablero
        Si casillero == negro
                Si casillero == esquina
                        resultado = resultado+50
                Sino Si casillero == casilla X
                        resultado = resultado-5
                Sino Si casillero == casilla C
                        resultado == resultado-1
                Sino Si casillero == adyacente a casilla C
                        resultado == resultado+5
                Sino Si casillero == casilla lateral al medio
                        resultado == resultado+2
                Sino
                        resultado == resultado+1
        Sino Si casillero == blanco
                Si casillero == esquina
                        resultado = resultado-50
                Sino Si casillero == casilla X
                        resultado = resultado+5
                Sino Si casillero == casilla C
                        resultado == resultado+1
                Sino Si casillero == adyacente a casilla C
                        resultado == resultado-5
                Sino Si casillero == casilla lateral al medio
                        resultado == resultado-2
                Sino
                        resultado == resultado-1
Devolver resultado
```

La función respetará los valores de las fichas con los factores ilustrados según se muestra en la imagen 4. En esta, los valores negativos quieren decir que tener una ficha del color propio en ese casillero es algo no conveniente para el jugador, mientras que tener una ficha en una esquina representa tener cincuenta fichas en su poder.

50	-1	5	2	2	1	5	50
-1	-5	1	1	1	1	-5	-1
5	1	1	1	1	1	1	5
2	1	1	1	1	1	1	2
2	1	1	1	1	1	1	2
5	1	1	1	1	1	1	5
-1	-5	1	1	1	1	-5	-1
50	-1	5	2	2	5	-1	50

Imagen 4: valuación de las fichas según sus posiciones en el tablero

5 Implementación

5.1 Diseño del programa

Se ha diseñado un diagrama de clases como se muestra en el anexo con el fin de ilustrar el uso de las mismas. A continuación se explican las funcionalidades de cada una de ellas:

<u>Clase Othello:</u>
Es el juego en cuestión. Las estructuras que incluye son un Tablero, un Juego, un Jugador Automático y dos variables en las que se almacena el color del usuario y del ordenador. Esta clase se encarga de organizar los turnos, indicar cuando finaliza la partida, y llevar a cabo el juego ya sea en modo multijugador y contra la computadora. A demás, se incluye todo lo que sea referente a la interfaz del tablero y los contadores de fichas. Brinda también las conexiones donde interactúa el usuario al cliquear en los botones, verificando que la acción sea válida y llevándola a cabo.

<u>Clase Tablero:</u>
Representa al tablero del juego Othello. La estructura que se implementó para almacenar la información es una matriz de 8 filas y 8 columnas. Los posibles datos en

ella son 0, 1 y 2, que representan que en la casilla no hay ninguna ficha, que hay una ficha blanca y que hay una ficha negra respectivamente.
Posee una serie de métodos de consulta sobre la configuración del tablero. Ellas consisten en verificar si existe alguna jugada posible para un determinado color, comprobar si dado un color y una posición es posible comer alguna ficha, calcular la cantidad de fichas de un color en el tablero, y consultar que hay sobre un casillero.
Para verificar si se come alguna ficha se debe recorrer el tablero desde la posición dada hacia los ocho sentidos posibles, lo que acota linealmente la función. En el caso de verificar si existe alguna jugada posible, este costo se multiplica por la cantidad de casillas en el tablero, ya que en el peor de los casos debe recorrer el tablero completo. Calcular la cantidad de fichas de un color está acotado al número de casillas total, mientras que consultar que hay en una casilla tiene un costo constante.
A su vez, se crearon métodos que consisten en modificar el tablero, ya sea realizando una jugada válida o simplemente colocando una ficha individual. El primero se encarga de, dada una posición vacía, comer las fichas contrarias en todas las direcciones posibles, mientras que el segundo modifica una casilla sin importar el resto del tablero.
Finalmente, la última función que posee este TDA es la de resetear el tablero, en caso de que se quiera iniciar una nueva partida.

Clase Juego: En ella se almacena, modifica y reinicia quien tiene el turno, y decide si se juega contra el ordenador o en modo multijugador. Contiene métodos muy simples que tienen que ver con la gestión de los turnos mediante el transcurso del juego. Posee 3 funciones observadoras con complejidades constantes ya que solo se encargan de obtener el turno actual, observar al turno enemigo y verificar si el jugador automático está activo.
Además, se implementaron tres funciones modificadoras, las cuales se ocupan de alterar el turno actual, activar al jugador automático o en caso de reiniciar el juego, resetear el turno al inicial.

Clase JugAutomatico: ésta clase se encarga de implementar mediante un conjunto de funciones al jugador automático. Permite obtener una posición válida e inteligente para un jugador tras la ejecución del algoritmo Minimax explicado en el punto 3. A su vez almacena en una variable la dificultad del juego que se está ejecutando.

Clase Opciones: ésta clase fue agregada con el fin de mejorar la calidad del programa, brindándole al usuario la posibilidad de elegir el color de fichas con el que quiere jugar contra el ordenador, y también permitirle cambiar el nombre de “blancas” o “negras” por el deseado.

Clase Contenido: la clase contenido se encuentra en el panel de ayuda y hace referencia a la explicación acerca del juego, aclarando cuál es la función de cada botón y las distintas partes que posee la interfaz. Al igual que la clase Opciones, surge con la necesidad de mejorar la calidad del juego, ayudando al usuario a adaptarse fácilmente al entorno.

Clase AcercaDe: incluye el diseño de una ventana que contiene información acerca del programa, como la fecha de realización del proyecto, la versión y las personas que intervinieron en el desarrollo del mismo.

Clase Ayuda: consiste en una ventana que muestra por pantalla las reglas del juego y la forma de jugarlo. Dicha clase es necesaria para interpretar correctamente las reglas del juego, y que el usuario no confunda con ciertas similitudes que tiene con el "Reversi".

5.2 Interfaz del programa

Como Othello se trata de un juego, se consideró necesaria una interfaz agradable que le brinde facilidad al usuario para jugar. Por lo tanto, se pensó el tablero con botones "cliqueables" en las casillas para que se pueda realizar una jugada simplemente haciendo clic sobre el cuadrado y sin necesidad de escribirla. También se dispusieron dos contadores en cada costado que indican la cantidad de fichas que posee cada jugador y que se actualizan cada vez que se realiza una jugada. Debajo del tablero se dispuso un cuadro que indica constantemente a quién le toca jugar. Al costado de este se encuentra un botón que sólo es visible en el modo "Vs Ordenador" y que al ser cliqueado realiza la jugada del rival automáticamente. Esa última implementación fue necesaria ya que si la computadora realizaría la jugada automáticamente, el usuario no podría ver cual fue el movimiento aplicado debido a la velocidad en que se ejecuta la acción.
En la parte superior de la pantalla se muestran dos menús. El primero permite elegir el tipo de juego, reiniciar la partida y configurar las opciones del juego, mientras que el segundo permite consultar información sobre el juego y sus reglas.

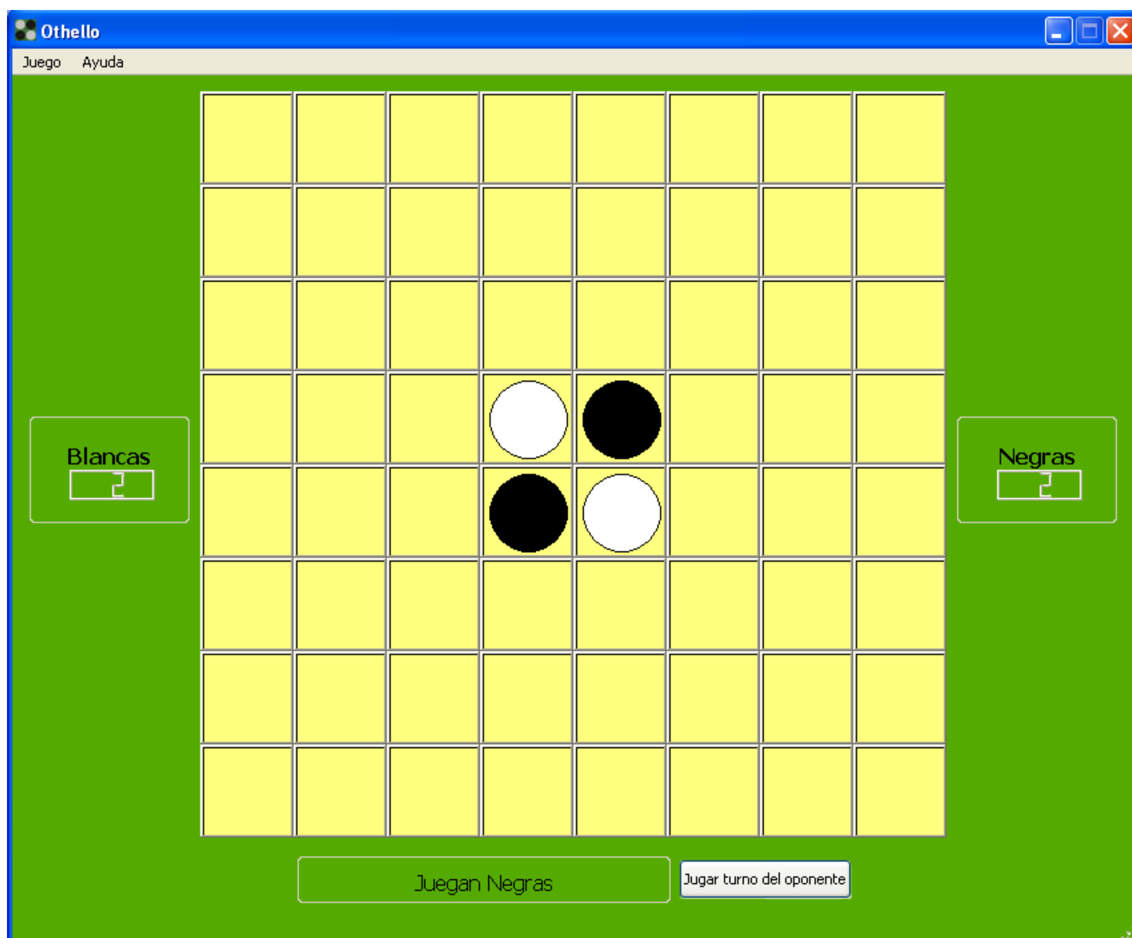

Imagen 5: Ventana principal del juego

Para elegir el color de fichas que se desean utilizar para el modo de juego “Vs ordenador” y cambiar los nombres de los jugadores se diseño una ventana a parte, con el fin de no acumular mucha información sobre la ventana donde se encuentra el tablero de juego.

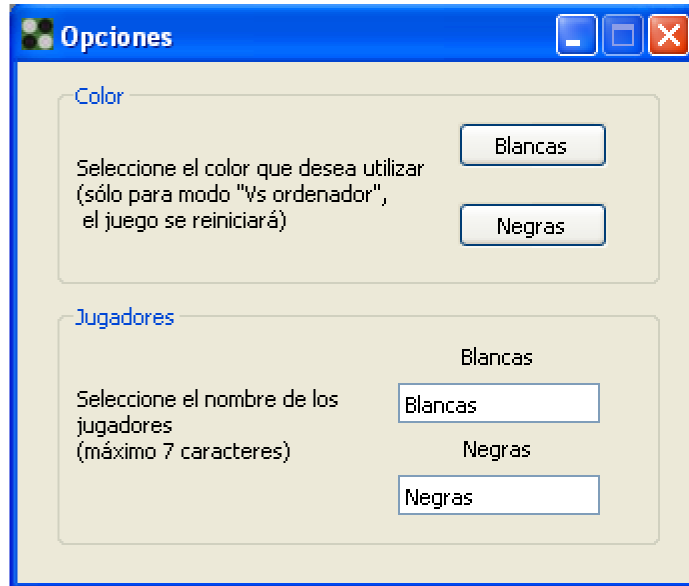

Imagen 6: Opciones de juego

5.3 Detalles de la implementación

El programa está desarrollado en el Lenguaje de Programación C++. El entorno de desarrollo integrado (IDE) utilizado es QT Creador de Trolltech. Se eligió este entorno debido a que brinda muchas facilidades para la creación de una interfaz gráfica (en lugar del CodeBlocks utilizado en la cursada) algo que se buscó desde el principio debido al poco conocimiento en el tema. La plataforma sobre la que se trabajó es Windows XP.

6 Complejidad Temporal:

6.1 Algoritmo minimax:

Al igual que en la búsqueda exhaustiva por profundidad, tiempo de la búsqueda se encuentra determinado por el tamaño del espacio de búsqueda, dado por:

$$M^N$$

Donde M es la movilidad y representa el factor de ramificación, mientras que N es el número de movimientos y representa la profundidad del árbol.
La movilidad es la cantidad de movimientos que un jugador puede hacer en su turno. Esta varía a medida que avanza el juego, aumentando de forma acusada al principio, durante el primer tercio de la partida, luego pasa por un máximo de movilidad de 12

movimientos (aproximadamente por el turno número 22), y finalmente decae más suavemente hasta que la movilidad es nula y finaliza la partida.
Se estima que el promedio de movilidad en una partida completa de Othello es de 8. A demás se sabe que la máxima cantidad de movimientos en el peor de los casos es de 60 (hay 64 casilleros en el tablero y se comienzan con 4 fichas ya colocadas).
Por lo tanto, el tamaño del árbol del juego completo es de aproximadamente 8^{60}, lo que equivale a 10^{54} nodos.
Expandir este árbol entero es muy costoso computacionalmente, por lo que se emplearon profundidades máximas de tres y seis en el juego desarrollado. Para las dificultades "Fácil" e "Intermedio" el espacio de búsqueda se reduce a 8^3 aproximadamente, mientras que en "Difícil" el árbol tiene un tamaño promedio de 8^6.
A su vez hay que considerar que para cada nodo expandido se crea un nuevo tablero, lo que tiene una complejidad lineal de O(n) donde n es la cantidad de casilleros del tablero. Esto se puede evitar creando métodos que permitan deshacer la última jugada realizada, de modo que se pueda volver al estado anterior una vez que se vuelve de la recursividad.
También se invoca a una función que realiza la jugada, pero cuyo costo también es acotado linealmente.

6.2 Posible mejora en la complejidad

Una de las mejoras para este proyecto que vale la pena remarcar es la Poda Alfa-Beta. Esta técnica tiene como fin reducir el número de nodos evaluados en un árbol de juego en un algoritmo Minimax. Toma dicho nombre de la utilización de dos parámetros que describen los límites sobre los valores hacia atrás que aparecen a lo largo de cada camino. Estos parámetros se refieren a:
Alfa: es el valor de la mejor opción hasta el momento a lo largo del camino para las negras, esto implicará por lo tanto la elección del valor más alto.
Beta: es el valor de la mejor opción hasta el momento a lo largo del camino para las blancas, lo que implica la elección del valor más bajo.
El valor de los parámetros se va actualizando a medida que se recorre el árbol. El método realizará la poda de las ramas restantes cuando el valor actual que se está examinando sea peor que el valor actual de Alfa o Beta para Negras o Blancas respectivamente.
Como el orden en que se examinan los sucesores en Othello es aleatorio, el número aproximado de nodos generados luego de aplicar la Poda Alfa-Beta se reduciría a $O(M^{3N/4})$.

7 Conclusión:

Se desarrolló una plataforma del juego Othello que permite jugar en modo multijugador. A su vez se implementó un algoritmo que permite obtener jugadas inteligentes en tiempo real, lo que posibilitó crear un modo de juego donde el usuario tiene como rival a la computadora. Este crea un árbol de juego con las posibles jugadas futuras que pueden desarrollarse en la partida, y se queda con el primer movimiento del cual al final se llega a un mayor beneficio, suponiendo que el adversario también realiza jugadas inteligentemente, denominando a esta estrategia "empatía".
El algoritmo obtiene jugadas válidas según la dificultad que se elija. Mientras más profundo se explore en el árbol de juego, se obtendrán resultados mejores, pero esto conlleva a un mayor costo computacional. Para la implementación de las diferentes

dificultades de juego se empleó: en “fácil” una profundidad de hasta tres jugadas futuras, donde sólo se consideran las fichas que se capturan, en “intermedio” se utiliza la misma profundidad pero con una función de utilidad que sigue una estrategia de juego en la cual las esquinas del tablero predominan en importancia, y finalmente en “difícil” le extendió la profundidad hasta seis jugadas futuras y a su vez se tiene en cuenta la estrategia recientemente mencionada.
El programa fue probado con profundidades mayores, y en las etapas de juego donde la movilidad es muy grande, el tiempo de ejecución se torna prolongado, algo no deseado para un usuario. De todas formas, se considera que en el nivel “difícil” la complejidad de vencer al ordenador es suficiente para que un jugador novato no pueda acercarse a la victoria, y para que un jugador avanzado tenga que emplear algunas estrategias si desea ganar la partida.

En lo personal, resultó interesante el desarrollo de un juego que permita la interacción entre el usuario y la computadora. A demás, el hecho de tener que probar la funcionalidad y eficiencia del algoritmo con un juego llevó a pensar nuevas estrategias para el mismo y tornó más entretenido el trabajo.
En el desarrollo del mismo se aprendió a diseñar una interfaz gráfica con las facilidades que brindan algunos entornos, algo que resulta de un amplio interés personal para los autores. A demás, el estudio profundo del algoritmo Minimax permitió ampliar los conocimientos sobre el mismo y comprender la importancia que tiene para el desarrollo de los juegos por turnos.

8 Referencias:

8.1 Webs:

http://www.it.uc3m.es/jvillena/irc/practicas/07-08/Othello.pdf

http://www.pressibus.org/ataxx/autre/minimax/paper.html

http://web.iti.upv.es/~evidal/students/sin/tema05/aima_c05_es2p.pdf

http://es.wikipedia.org/wiki/Otelo_en_computadoras#Otelo_8_x_8

https://sites.google.com/site/elreversista/preguntas-y-curiosidades-sobre-el-reversi

8.2 Imágenes:

http://www.grupoalquerque.es/ferias/2006/europa/othelo.gif

http://olab.is.s.u-tokyo.ac.jp/~kamil.rocki/tree.png