



Universidad Autónoma Metropolitana

Unidad Iztapalapa

División de Ciencias Básicas e Ingeniería

Departamento de Ingeniería Eléctrica

Empleo de Algoritmos Genéticos en el juego de Othello

Tesis que presenta el alumno
Castillo Ibarra Salvador

94316764

Para la obtención del grado de
Licenciado en Computación

Asesor
John Goddard Close

Mayo 2002



Universidad Autónoma Metropolitana

Unidad Iztapalapa

División de Ciencias Básicas e Ingeniería

Departamento de Ingeniería Eléctrica

Empleo de Algoritmos Genéticos en el juego de Othello

Tesis que presenta el alumno
Castillo Ibarra Salvador

94316764

Para la obtención del grado de
Licenciado en Computación

Asesor

John Goddard Close

Mayo 2002

Indice

Introducción	5
Acerca de Othello	6
Notación	6
Las reglas del juego	7
El objetivo	7
La posición inicial	7
Haciendo un movimiento o jugada.....	8
Fin del juego	10
Otros trabajos sobre Othello	11
IAGO	11
BILL	11
Algoritmos genéticos	12
Othello7	13
¿Qué es?	13
¿Cómo trabaja?	13
Funciones de evaluación	13
El mejor movimiento.....	15
Algoritmos genéticos	15
Aspectos técnicos	18
Clases.....	18
Organización de archivos.....	20
Cómo agregar una red neuronal.....	21
Manual de usuario.....	22
Interfaz.....	22
Menú.....	23
Juego.....	23
Evolución.....	24
Archivos *.evl.....	25
Archivos *.tab	25
Resultados	27
Evolución	27
Juegos Humano vs CPU.....	27
Juegos CPU vs CPU.....	28
Conclusiones	29

Bibliografía.....	30
Sitios de interés.....	31

Introducción

Recuerdo que mi primer encuentro con Othello fue cuando lleve un curso de Inteligencia Artificial. Tuve que hacer un proyecto final y elegí hacer un programa que jugara Othello. Tuve suficiente tiempo para hacer dicho trabajo; pero como buen universitario lo deje al final. Por momentos pensé que no lo terminaría pero afortunadamente no fue este el desenlace: concluí dicho programa en el tiempo establecido y pude aprobar la materia.

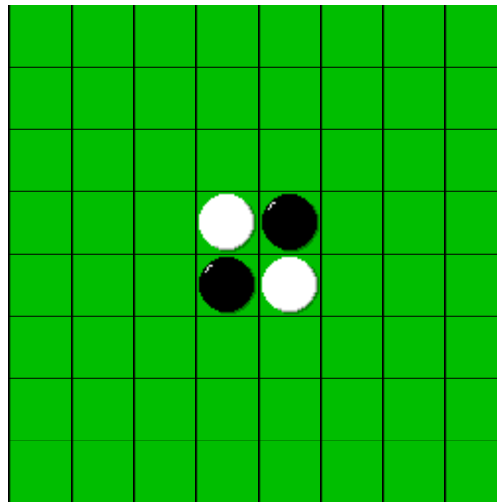
Pero ¿a qué viene esta historia?, bueno pues lo que quiero decir con esto es que Othello, como bien han sido mencionado en muchos artículos que se refieren a este ingenioso juego, es un juego muy sencillo de jugar: sus reglas son muy fáciles de aprender por lo tanto su implementación es relativamente sencilla. No por esto deja de ser atractivo como tema de investigación y prueba de ello es la gran cantidad de trabajos que uno puede encontrar en Internet.

Este trabajo se desarrollo pensando en encontrar la manera de enseñar a una computadora a jugar bien y verificar si la manera de jugar del ordenador era el esperado después del entrenamiento.

El método que se utilizó para crear Othello7 esta basado en algoritmos genéticos y funciones de evaluación que representen características importantes en un tablero de Othello. Los operadores genéticos empleados son mutación y elitismo que más adelante se detallan.

Acerca de Othello

Notación



El estado inicial

a1	b1	c1	d1	e1	f1	g1	h1
a2	b2	c2	d2	e2	f2	g2	h2
a3	b3	c3	d3	e3	f3	g3	h3
a4	b4	c4	d4	e4	f4	g4	h4
a5	b5	c5	d5	e5	f5	g5	h5
a6	b6	c6	d6	e6	f6	g6	h6
a7	b7	c7	d7	e7	f7	g7	h7
a8	b8	c8	d8	e8	f8	g8	h8

' Nomenclatura de los cuadros

a1	C					C	h1
C	X					X	C
C	X					X	C
a8	C					C	h8

| Cuadros Especiales

El tablero está dividido en ocho columnas y ocho renglones. Etiquetamos cada columna con una letra desde *a* hasta *h* de izquierda a derecha. Numeramos los renglones desde 1 hasta el 8 de arriba hacia abajo. Esto es opuesto a la convención usada en ajedrez. Así, como se muestra arriba, la esquina superior izquierda es llamada *a1*, y la inferior derecha es *h8*. Las piezas usadas en el juego son llamadas *discos*.

Algunos de los cuadros en el tablero son especialmente importantes. Los cuatro cuadros esquina (a1, h1, a8 y h8) son muy útiles tenerlos, puesto que éstos nunca se pueden recuperar; los que a partir de este momento nos referiremos a éstos únicamente como esquinas. Como consecuencia es peligroso hacer un movimiento en un cuadro siguiente a un cuadro esquina vacío a principios de el juego (tu oponente puede ser capaz de usar tu disco para hacer un movimiento en el cuadro esquina). Los cuadros b2, g2, b7 y g7 (empiezan una diagonal a partir de una esquina) son conocidos como *cuadros-X*. Los demás cuadros que están junto a las esquinas son llamados *cuadros-C*. Esto se muestra en el tercer diagrama de arriba.

Las reglas del juego

Othello es un juego de estrategia jugado por dos jugadores: Blanco y Negro. Este es jugado sobre un tablero de 8x8 (usualmente en color verde). Los dos jugadores colocan 64 discos cada uno de los cuales es negro de un lado y blanco del otro. Para conveniencia, cada jugador empieza con 32 discos pero no le pertenecen al jugador si su oponente termina sus discos, entonces esta obligado a darle los que necesite. Un disco es negro si el lado negro es visible y blanco si su cara es blanca esta arriba.

El objetivo

El ganador es el jugador que tenga mas discos de su color que su oponente al final del juego. Esto pasará cuando ninguno de los dos jugadores tenga un movimiento válido. Generalmente esto sucede cuando los 64 cuadros están ocupados.

La posición inicial

Al inicio del juego, dos discos negros son colocados en e4 y d5 y dos discos blancos están en d4 y e5 (ver figura 1).

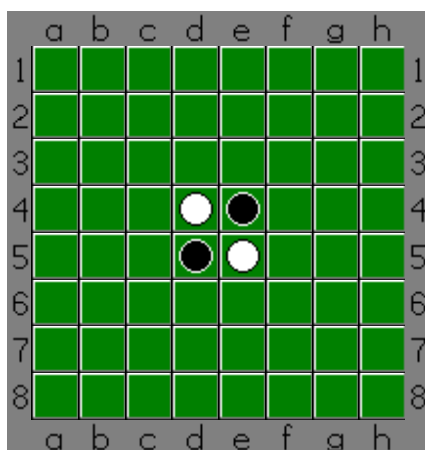


Figura 1: la posición inicial

Negro siempre empieza, y los dos jugadores subsecuentemente toman turnos para mover.

Haciendo un movimiento o jugada

En su turno, un jugador debe colocar un disco de su color en un cuadro vacío del tablero, adyacente a un disco del oponente. En adición, para poner su disco, debe flanquear uno o varios discos de su oponente entre el disco puesto y otro disco de su mismo color que ya este en el tablero. El jugador entonces voltea a su color todos los discos que fueron flanqueados. Los discos no son removidos del tablero ni tampoco movidos de un cuadro a otro.

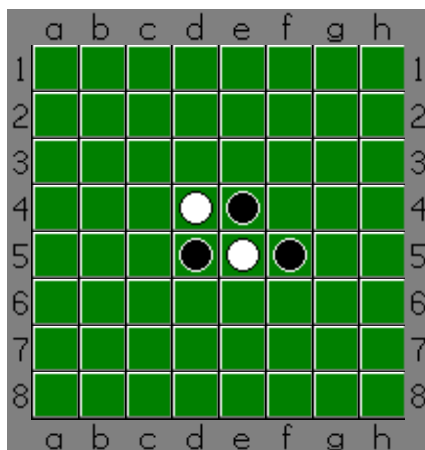


Fig. 2: Negro juega f5

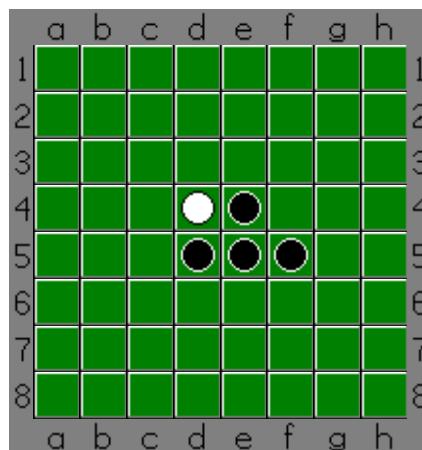


Fig. 3: y voltea e5!

El primer movimiento de negro puede ser, por ejemplo, f5 (ver figura 2). Jugando f5, el flanquea el disco blanco en e5 entre el disco jugado y otro disco negro que se encontraba en el tablero (aquí d5); el entonces voltea este disco (ver figura 3). Negro pudo también haber movido e6, c4 o d3. Como sea, estos cuatro movimientos de negro son perfectamente simétricos; negro no necesita tiempo en pensar acerca de su primer movimiento.

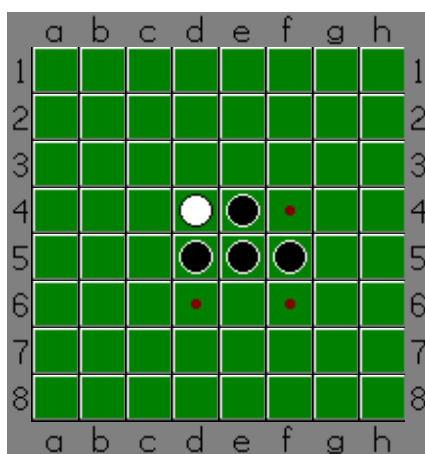


Fig. 4: Blanco f4, f6 o d6

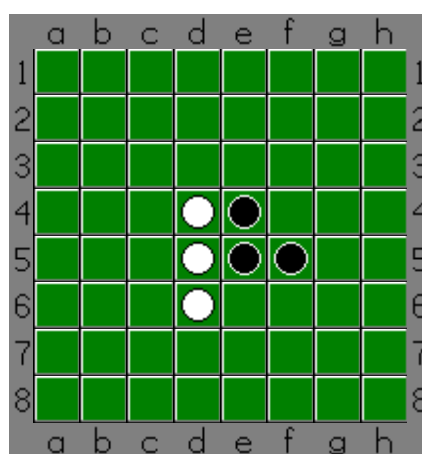


Fig. 5: Si blanco juega d6.

Ahora es el turno de que blanco mueva. El tiene tres posibles movimientos (ver figura 4). Cada posible movimiento voltea al menos un disco del oponente. Blanco puede jugar f4, f6 o d6. Notar que los discos pueden ser flanqueados en las ocho direcciones. Además, en cada dirección muchos discos pueden ser flanqueados (ver figuras 6 y 7). Todos esos discos flanqueados deben ser volteados.

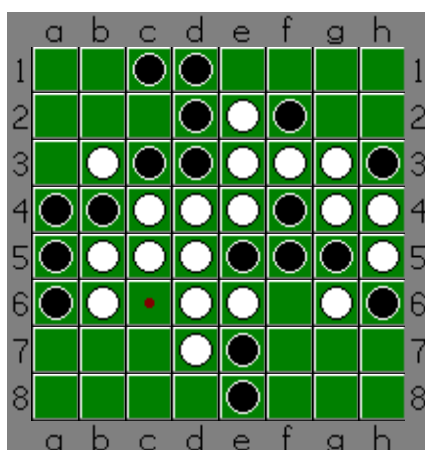


Fig. 6: Negro juega c6...

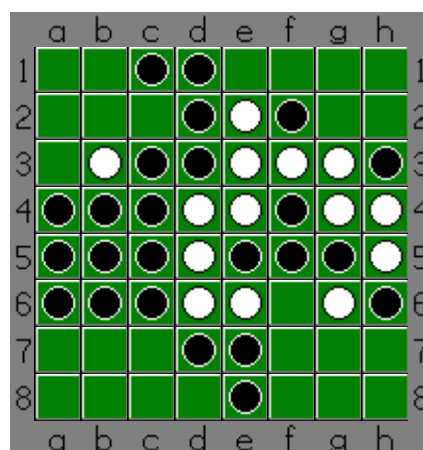


Fig. 7: resultando en esta posición

Negro jugó en c6. Voltea los discos en b6 (flanqueado por el disco en a6), b5 (flanqueado por a4), d7 (flanqueado por e8), c5 y c4 (flanqueado por c3). Nota que d6 y e6 no son volteados debido a que el cuadro f6 está vacío.

No hay reacción en cadena: los discos volteados no pueden ser usados para voltear otros discos en la misma jugada. De tal manera, en la figura 8, negro mueve en a5:

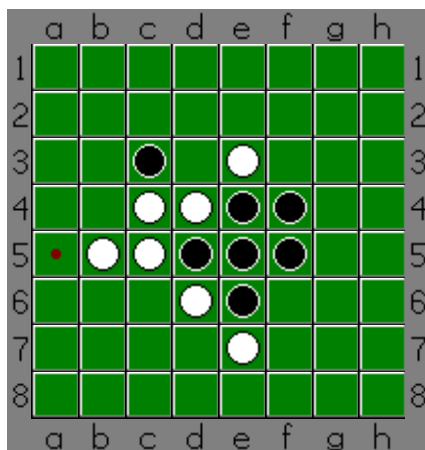


Fig. 8 : Negro juega a5...

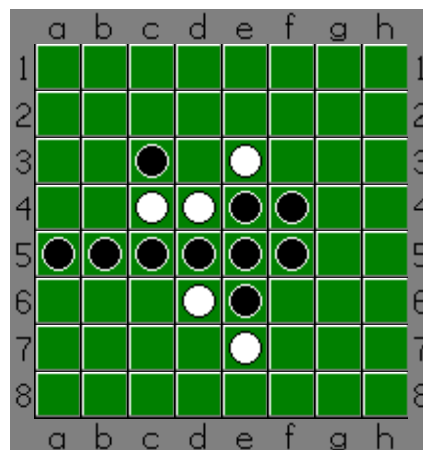


Fig. 9 : c4 permanece blanco.

Los discos en b5 y c5 son volteados puesto que están flanqueados. En este punto, aunque c4 esta flanqueado, no es volteado (ver figura 9). La razón de esto es que no esta flanqueado entre el disco jugado y otro disco.

Si, en tu turno, no puedes hacer un movimiento para voltear al menos un disco del oponente de acuerdo a estas reglas, tu debes pasar tu turno y tu oponente volverá a tirar. Pero si hay un movimiento válido, debes jugarlo.

Fin del juego

El juego termina cuando ninguno de los dos jugadores tiene un movimiento válido. Generalmente, esto sucede cuando los 64 cuadros son ocupados. Sin embargo, es posible que algunos cuadros permanezcan vacíos donde ninguno de los jugadores pueda tirar: por ejemplo, si todos los discos son del mismo color después de un turno, o en una posición como la de abajo (ver figura 10).

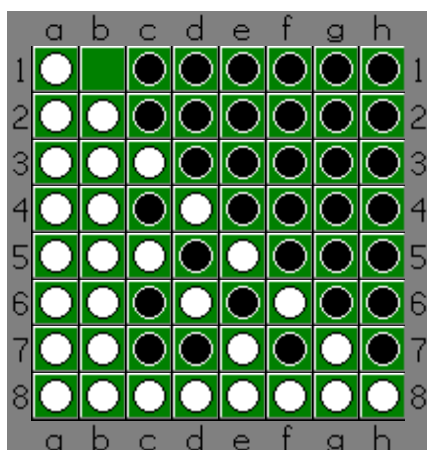


Fig. 10: El juego terminal!

Ninguno de los dos jugadores puede jugar en b1 puesto que no se pueden voltear discos. En este caso, contamos los discos para determinar el marcador final. Los cuadros vacíos son dados al ganador por convención. En este juego, blanco tiene 29 discos y negro tiene 34 discos, con un cuadro vacío. Entonces negro gana 35-29.

Otros trabajos sobre Othello

Este juego ha recibido gran atención dentro de la ciencia de la Computación por más de 10 años. A continuación se mencionan unos cuantos trabajos al respecto.

IAGO

Fue Paul Rosenbloom quién apunto que aunque el juego de Othello tiene un promedio 5 de bifurcación y una longitud limitada (menos de 64 movimientos) éste aún no puede ser solucionado exactamente y tiene un gran grado de complejidad para ser tratado como objeto de análisis científico. Rosembloom analizó dentro del juego Othello un par de estrategias principales (territorio estable y movilidad), cada una descomponible en sub-estrategias. Representaciones cuantitativas de esos conceptos fueron combinados con una sencilla función de evaluación. Junto con el algoritmo de búsqueda α - β , profundidad cambiante y orden al mover, esta función formo la base de el programa IAGO de Rosenbloom.

Aunque IAGO mostraba ser un muy *jugador* a nivel mundial tenia muchos inconvenientes. Primero, el conjunto de conceptos usado por el programa era mas bien limitado. Segundo, los conceptos, también llamadas características (*features*), fueron asumidas independientes y además combinadas en una función de evaluación lineal. Tercero, los coeficientes de la aplicación en la función de evaluación fueron seleccionados a mano, lo cual deja un significativo margen de error. Cuarto, IAGO usó una sencilla función de evaluación para todo el juego, aunque ahora es bien sabido que diferentes estrategias son necesarias para diferentes etapas del juego.

BILL

K. F. Lee y S. Majan se dirigieron a estos inconvenientes creando un programa nombrado BILL, que usó un conjunto de características (*features*) ligeramente extendido. Las representaciones de estas características fueron significativamente mejoradas a través de el uso de tablas pre-calculadas que permitieron a BILL reconocer cientos de miles de patrones en tiempo constante. Los autores aplicaron aprendizaje Bayesiano para combinar conceptos en la función de evaluación de BILL, que directamente estimaba la probabilidad de ganar. BILL aprendió muchas funciones de evaluación, una para cada movimiento entre la 25ª y la 48ª. Estas funciones de evaluación fueron entrenadas usando una gran base de datos de 3000 juegos creados por una versión anterior de el programa.

Estas propiedades y el mejoramiento en las técnicas de búsqueda y los tiempos de los algoritmos permitieron a BILL sobrepasar completamente a IAGO, pero como se menciona, BILL requiere de una gran base de datos de juegos para entrenar, y la calidad e la función de evaluación depende completamente de la calidad de esta base de datos. La función de evaluación de BILL es un polinomio cuadrático, que toma en cuenta linealmente una

interrelación entre las características (features); la pregunta de existencia de una mayor complejidad en la interacción de estas características permanece abierta.

Algoritmos genéticos

Un método diferente fue elaborado por D. Moriarty y R. Miikkulainen. Ellos evolucionaron una población de redes neuronales usando un algoritmo genético para evaluar posibles movimientos. Toda red ve la actual configuración del tablero como su entrada e indica la evaluación de éste (*fitness*) como su salida. En otras palabras en contra de buscar a través de posibles escenarios de juego para la mejor jugada, la red neuronal confía en el reconocimiento de patrones en la decisión de que movimiento será el más promisorio.

El punto interesante es que cada evolución de redes neuronales o criaturas fue necesario diferenciar entre todos los posibles movimientos, legales e ilegales, entonces el mejor movimiento legal fue elegido para continuar el juego. Los resultados mostraron que las redes con arquitectura fija no fueron capaces de aprender cualquier estrategia trivial, cuando las criaturas con una arquitectura de mutación ingeniaron elaborar el concepto de movilidad.

La debilidad de este método es que las criaturas necesitan ser evolucionadas contra otro jugador de Othello y si bien las criaturas finalmente son mejores que su oponente, la calidad de la red final es proporcional a la calidad de su oponente.

Othello7

¿Qué es?

Othello7 es una aplicación que se puede desglosar en dos partes:

- § Jugar Othello. Se tiene la opción de jugar contra el CPU o el CPU contra si mismo.
- § Partiendo de una población aleatoria de polinomios aplicando algoritmos genéticos se puede evolucionar dicha población.

Una característica que se buscó cubrir es que el programa fuera flexible para hacer experimentos y se pudieran agregar fácilmente diversos tipos de polinomios (adelante se hablara con mas detalle al respecto). Así mismo tiene capacidades como guardar juegos inconclusos y recuperarlos posteriormente (solo en modalidad *humano vs CPU*). Y evolucionar polinomios a partir de una evolución hecha anteriormente.

¿Cómo trabaja?

Funciones de evaluación

Son funciones que representan el *fitness* de un tablero (que tan bueno es el estado del tablero para el jugador). El dominio de dichas funciones es la configuración del tablero.

Ejemplo de función de evaluación:

$$f(X) = 10x_1 - 5x_2 + 3x_3$$

donde:

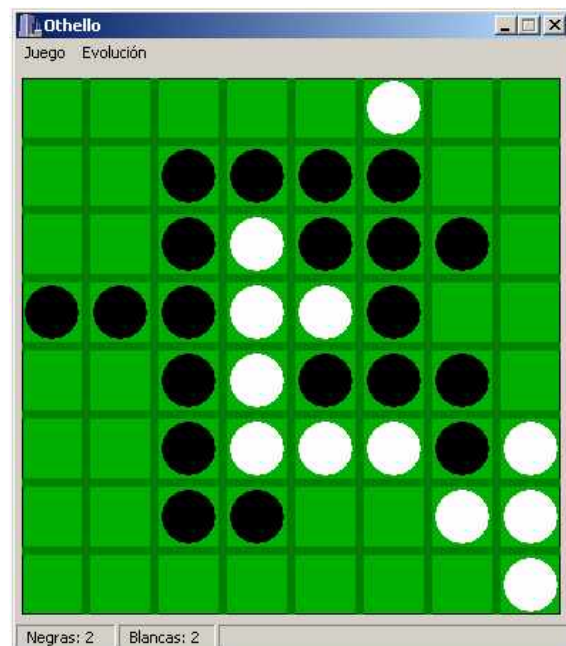
x_1 : es el número de fichas en las casillas de las esquinas.

x_2 : es el número de fichas adyacentes a las esquinas.

x_3 : número de fichas en las posiciones centrales del tablero (D4, E4, D5, E5, ver nomenclatura abajo).

por lo tanto si $f(X)$ se calcula para el jugador blanco de la figura de la derecha tenemos:

$$f(X) = 10(1) - 5(2) + 3(3)$$



El ejemplo de función de evaluación mostrado arriba no se utiliza en el juego. Existen dos tipos de polinomios que se utilizan en el programa:

Polinomio SCI. $f(X) = 50x_1 - 25x_2 + x_3$

x_1 : Es la diferencia de fichas colocadas (las del jugador menos las de su oponente) en las esquinas. Por ejemplo si el jugador tiene dos fichas en las esquinas y su oponente solamente una entonces $x_1 = 2 - 1 = 1$.

x_2 : Es la diferencia de fichas colocadas en las casillas adyacentes (las del jugador menos las de su oponente) a las esquinas.

x_3 : Es la diferencia de fichas colocadas en todo el tablero (las del jugador menos las de su oponente).

Polinomio JMA

$$f(X) = 500x_1 - 250x_2 - 150x_3 + 30x_4 + 16x_5 + 10x_6 + 2x_7 + x_8 + 30x_9 - 8x_{10}$$

Las variables x_1 a x_8 se obtienen directamente del tablero como la diferencia de las fichas colocadas en cada posición x_i (para $i = 1 \dots 8$) del jugador menos las del oponente.

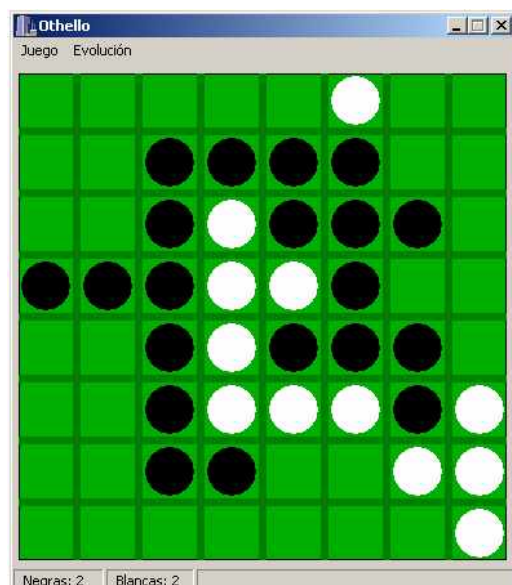
	1	2	3	4	5	6	7	8
A	x_1	x_3	x_4	x_6	x_6	x_4	x_3	x_1
B	x_3	x_2	0	0	0	0	x_2	x_3
C	x_4	0	x_8	x_7	x_7	x_8	0	x_4
D	x_6	0	x_7	x_5	x_5	x_7	0	x_6
E	x_6	0	x_7	x_5	x_5	x_7	0	x_6
F	x_4	0	x_8	x_7	x_7	x_8	0	x_4
G	x_3	x_2	0	0	0	0	x_2	x_3
H	x_1	x_3	x_4	x_6	x_6	x_4	x_3	x_1

x_9 Si una esquina esta ocupada por una ficha del jugador, a esta variable se suman el número de fichas contiguas que se encuentren sobre las casillas de los bordes del tablero que convergen en esta esquina. Y se restan en caso que se han fichas del oponente.

Para el caso del jugador blanco x_9 vale:

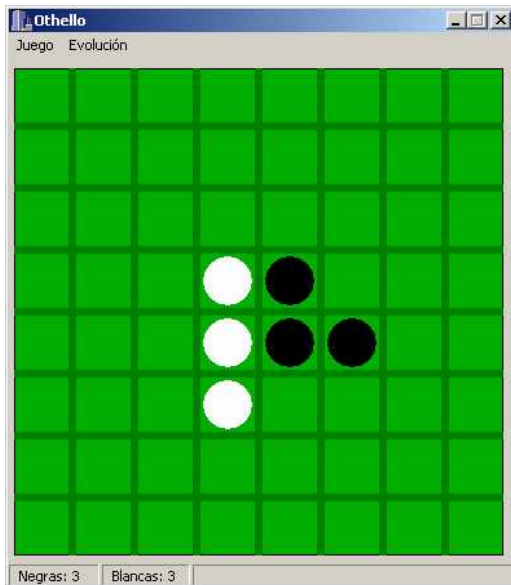
$$x_9 = 3 + 1 = 4$$

como tiene una ficha en una esquina (H8) se suman las fichas H8, H7 y H6 más H8 porque representa su lado que va de H8 a H1.



x_{10} : Representa la diferencia de *libertad* total entre el jugador y el oponente. Se obtiene de siguiente manera:

Por cada ficha del jugador en el tablero se suman las casillas vacías que sean adyacentes a dicha ficha (libertad del jugador) y también se calcula la libertad del oponente. La libertad total es la diferencia entre la libertad del jugador y la del oponente.



Para la configuración que se muestra en la figura de la izquierda x_{10} vale:

$$\begin{aligned} X_{10} &= (3 + 6 + 4) - (6 + 4 + 5) \\ &= 13 - 15 = -2 \end{aligned}$$

Suponiendo que el jugador es el color Negro y el oponente el color Blanco.

El mejor movimiento

Por medio del método minimax se crea un árbol en donde los nodos son tableros con un valor que se calcula por medio de las funciones de evaluación aplicadas a estos tableros. Es así como el CPU calcula o identifica cual es el mejor movimiento. La profundidad del árbol es fija, con un valor de 4. Esta operación se realiza en un juego contra un humano o en un juego CPU vs CPU.

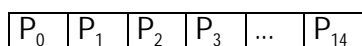
Algoritmos genéticos

Se utilizan para *evolucionar* poblaciones de polinomios y encontrar los coeficientes mejor evolucionados, es decir, con los que el polinomio gana más juegos. Por ejemplo para el polinomio de tipo SCI $f(X) = 50x_1 - 25x_2 + x_3$, se buscan mejores coeficientes que 50, 25 y 1. Los polinomios tienen un atributo llamado *fitness*, que en términos prácticos representa que tan bueno es el polinomio jugando Othello.

Inicialmente se tiene una población de 15 polinomios con coeficientes aleatorios. Estos 15 polinomios sufren una *mutación*: a cada uno de sus coeficientes se le suma una aproximación gaussiana (un número aleatorio); esto da como resultado 15 polinomios que son

llamados polinomios hijos de los polinomios mutados. Ahora se tiene una población de 30 polinomios.

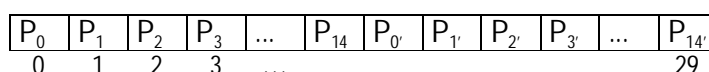
población inicial



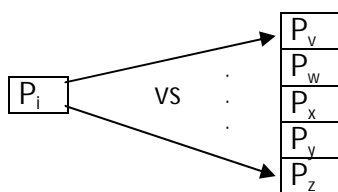
mutación



nueva población



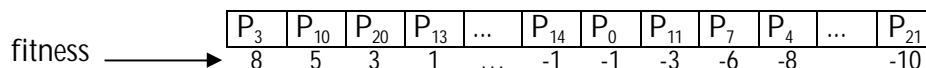
Cada uno de los 30 polinomios escoge 5 polinomios al azar de esta población contra los que disputa un juego de Othello. En cada juego entre polinomios el fitness de éstos se modifica de la siguiente manera: el polinomio ganador del juego se le suman una unidad a su fitness, al perdedor se le restan dos unidades y si hay empate el fitness no se afecta.



donde $i = \{0...29\}$

v, w, x, y, z son números aleatorios en el dominio $\{0...29\}$

Una vez que todos los polinomios (los 30) han jugado sus cinco encuentros. Se ordenan por su fitness, de mayor a menor.



En este punto se aplica otra vez el operador de mutación a los mejores 15 polinomios, sustituyéndose los 15 peores polinomios por los 15 que se acaban de crear por mutación.

P_3	P_{10}	P_{20}	P_{13}	...	P_{14}						
8	5	3	1	...	-1						



mutación

nueva población

P_3	P_{10}	P_{20}	P_{13}	...	P_{14}	$P_{3'}$	$P_{10'}$	$P_{20'}$	$P_{13'}$...	$P_{14'}$
8	5	3	1	...	-1	0	0	0	0	...	0

Por último se aplica un operador genético llamado *elitismo* que consiste en guardar el polinomio con mejor fitness (polinomio en la posición cero) en una posición 31ª, para el caso de que sean más de una generación se verifica el fitness del polinomio en la posición 31ª con el de la posición 0 y si es mayor su fitness de éste último, se guarda en la posición 31ª.

P_3	P_{10}	P_{20}	P_{13}	...	P_{14}	$P_{3'}$	$P_{10'}$	$P_{20'}$	$P_{13'}$...	$P_{14'}$	P_{10}
8	5	3	1	...	-1	0	0	0	0	...	0	5

← Polinomio 31



elitismo

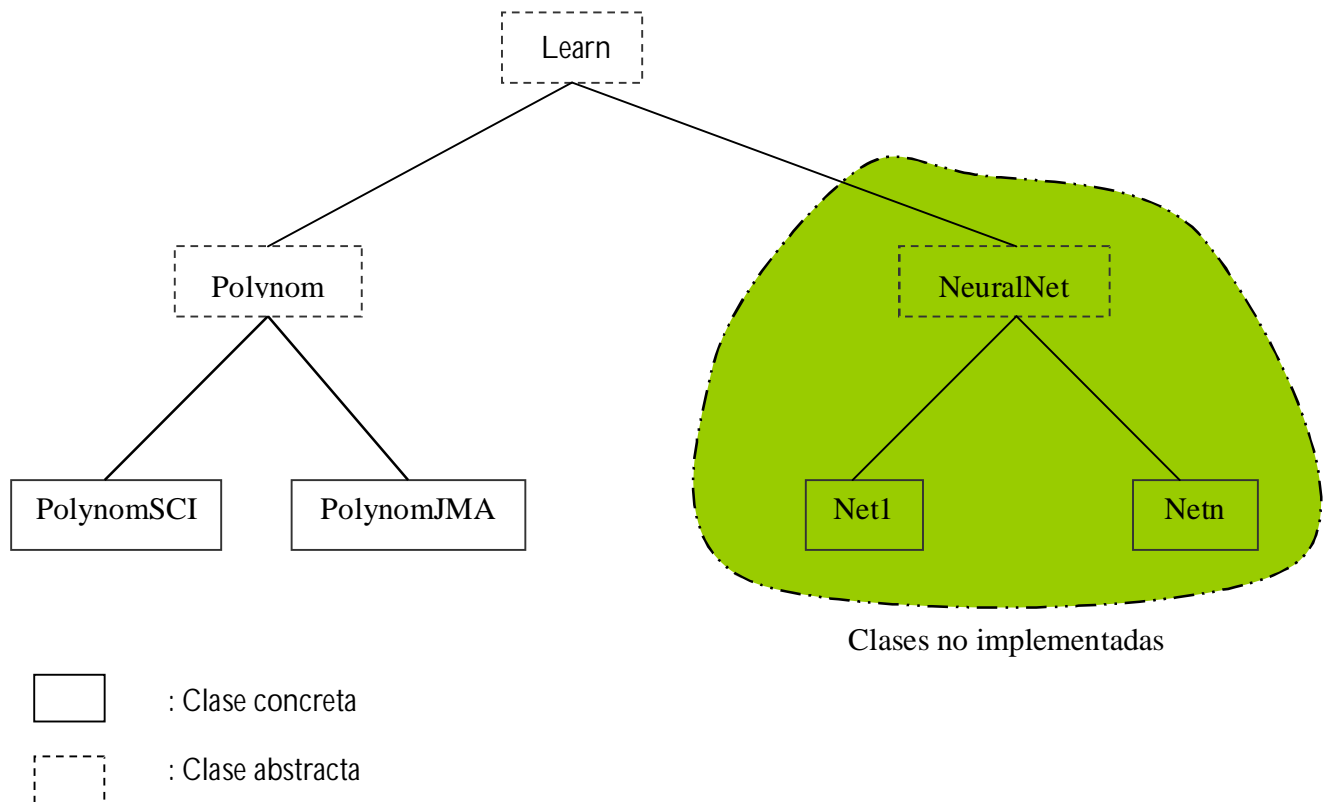
P_3	P_{10}	P_{20}	P_{13}	...	P_{14}	$P_{3'}$	$P_{10'}$	$P_{20'}$	$P_{13'}$...	$P_{14'}$	P_3
8	5	3	1	...	-1	0	0	0	0	...	0	8

A todo este conjunto de operaciones que se aplica a la población de polinomios se le llama generación. Con mayor número de generaciones se evolucionan mejor los polinomios.

Aspectos técnicos

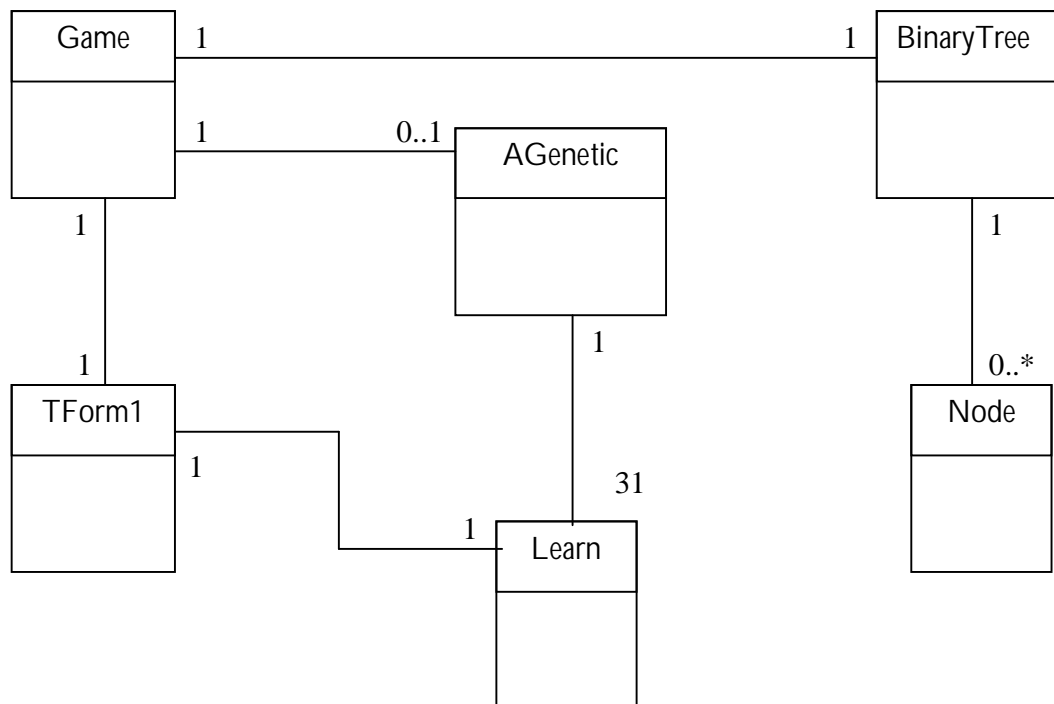
Clases

A continuación se presenta un bosquejo de cómo de las relaciones entre las clases utilizadas en el programa y más adelante se describen.



En el esquema de arriba se puede apreciar la jerarquía de clases de los objetos que se utilizan para el aprendizaje: polinomios o redes neuronales. La parte que respecta a redes neuronales no está implementada, en caso de hacerlo se debe crear una clase (que no necesariamente se llame NeuralNet) que derive de Learn. Por otra parte si se desea aumentar el tipo de polinomios se deben crear clases abstractas que deriven de Polynom.

En el siguiente diagrama se presenta de manera muy superficial las relaciones entre las principales clases del programa.



Tform1

Esta clase es la encargada de la interfaz con el usuario. En ella están contenidos los componentes visuales. Se encarga de pintar el tablero, las fichas, cargar archivos *.evl, *.tab, obtiene las coordenadas del puntero del ratón, etc.

Node

Esta clase genera objetos que representan nodos de un árbol *mínimax*. Sus atributos mas importantes son board (configuración de un tablero), value (valor de ese tablero).

BinaryTree

Representa un árbol binario minimax. Sus métodos son los propios de una estructura de datos de este tipo. Además cuenta con métodos como minimax par e impar.

Learn

Esta es una clase abstracta. Por lo tanto en el diagrama de arriba si se piensa en los objetos que genera son PolynomSCI, PolynomJMA. Representa el tipo de objeto de aprendizaje. Sus atributos dependen de los objetos que contenga (coeficientes de un polinomio, pesos de capas de una red, etc.), sin embargo un atributo común a todos sus objetos es el fitness. Tiene 3 métodos virtuales (por lo que es abstracta) que deben ser definidos por las clases que deriven de ésta y que sean clases concretas (la definición de estos

métodos depende del tipo de objeto que se cree). Para definir estos métodos en clases nuevas (clases hijas) ver la definición en PolynomSCI y PolynomJMA.

Game

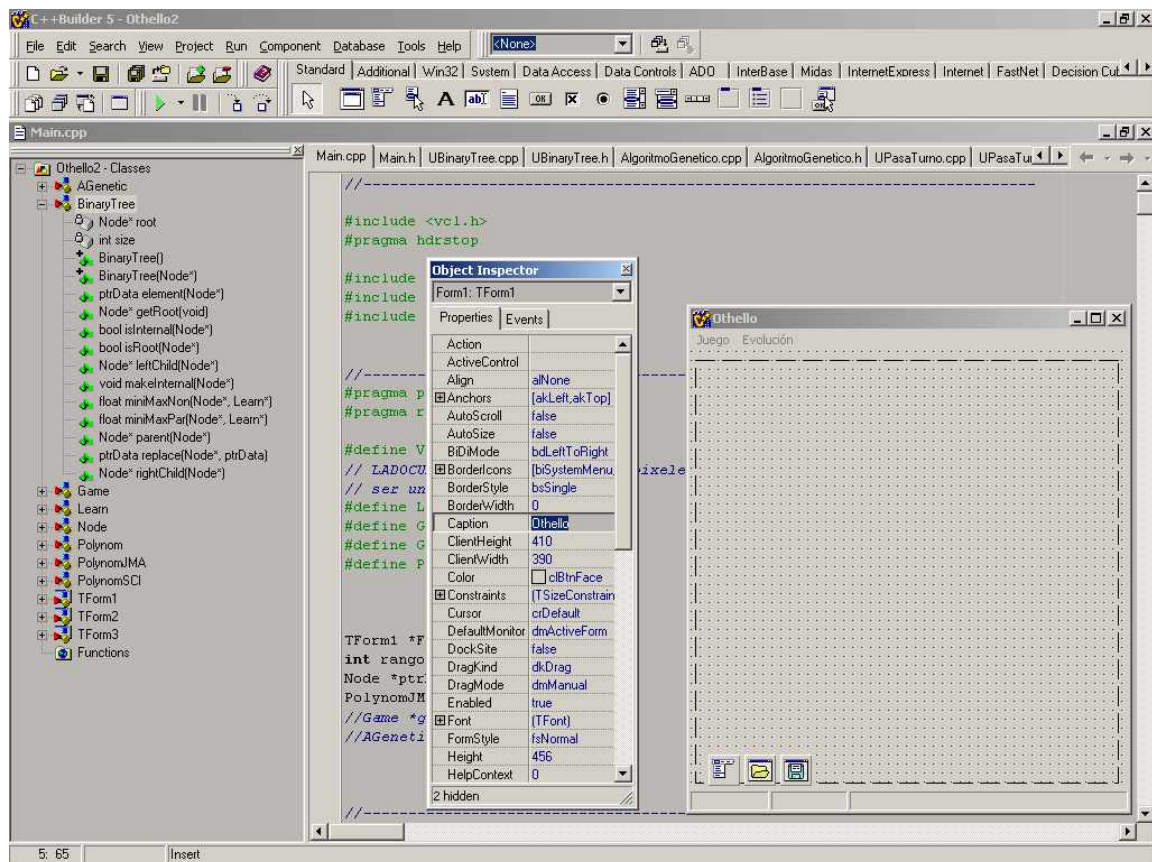
Representa el estado de un juego. Sus atributos principales son piezas por jugador, turnos perdidos por jugador, el turno actual, el árbol minimax. Con esta clase se puede hacer un juego entre dos elementos tipo Learn, es decir, que juegue el CPU contra si mismo; tiene más métodos que son muy fáciles de entender.

AGenetic

Esta clase es la abstracción de lo que sería una evolución. Por medio de sus atributos representa el número de generaciones, la población a evolucionar y sus métodos establecen los operadores genéticos.

Organización de archivos

La herramienta que se utilizó para desarrollar Othello7 fue C++ Builder versión 5.



Los archivos que se crearon son los siguientes:

Othello2.bpr: Es un archivo de proyecto (en C++ Builder).
El código se encuentra en:

Archivo	Clases
Main.cpp	TForm1
UBinaryTree.cpp	Learn, Polynom, PolynomSCI, PolynomJMA, Node, BinaryTree
AlgoritmoGenetico.cpp	Game, AGenetic
UPasaTurno.cpp	TForm2 (cuadro de diálogo)
UNumGens.cpp	TForm3 (cuadro de diálogo)

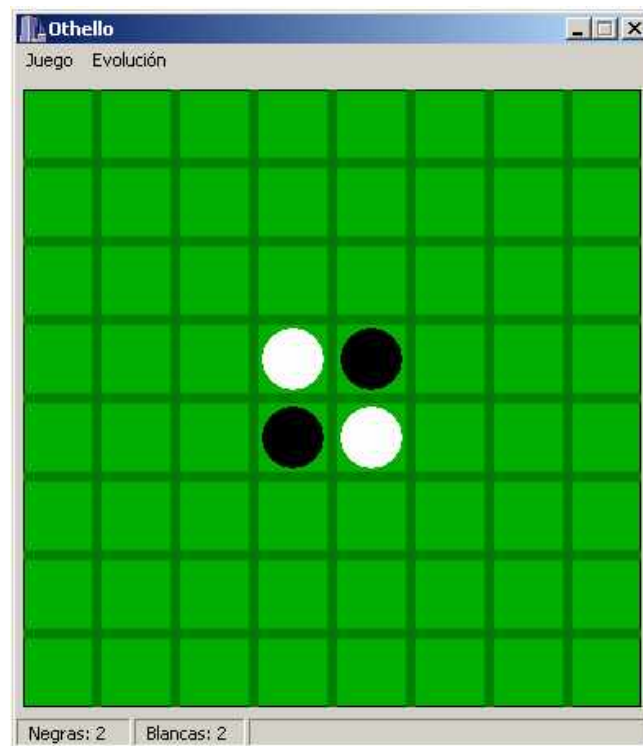
Cómo agregar una red neuronal

Esta es una idea de como sin entrar en detalles de como implementar una red en Othello7.

- § Crear una clase abstracta (o virtual) derivada de Learn para las redes neuronales en general (suponiendo que se quieran crear diferentes tipos) y que será análoga a Polynom a la que llamaremos NeuralNet.
- § NeuralNet debe definir las operaciones básicas de una red.
- § Crear clases concretas de las redes que se deseen implementar; dichas clases deben definir los métodos evalua() y generaHijo() (métodos abstractos de Learn) además de los suyos.
- § Se debe agregar código en los métodos HumanovsCPU1Click y CPU1vsCPU2Click de la clase TForm1.
- § Agregar en el menú Evolución->Nuevo el elemento Redes Neuronales.
- § Agregar el elemento Red1 (tipos que se deseen crear) en el menú Evolución->Nuevo->Redes Neuronales y su método asociado para generar la evolución, como en los polinomios SCI o JMA.

Manual de usuario

Interfaz



En la interfaz de la aplicación se identifican los siguientes elementos:

- § Un tablero de 8x8 casillas para jugar Othello.
- § Un menú con las opciones: Juego y Evolución que posteriormente se explicarán con detalle.
- § Una barra de estado en la parte inferior que se utiliza para mostrar el estado del juego.

Nota: A continuación se habla de *jugador* y *oponente*. El jugador es el primer contrincante que se elige ya sea Humano o CPU y el segundo es el contrincante número 2 que solo puede ser un CPU.

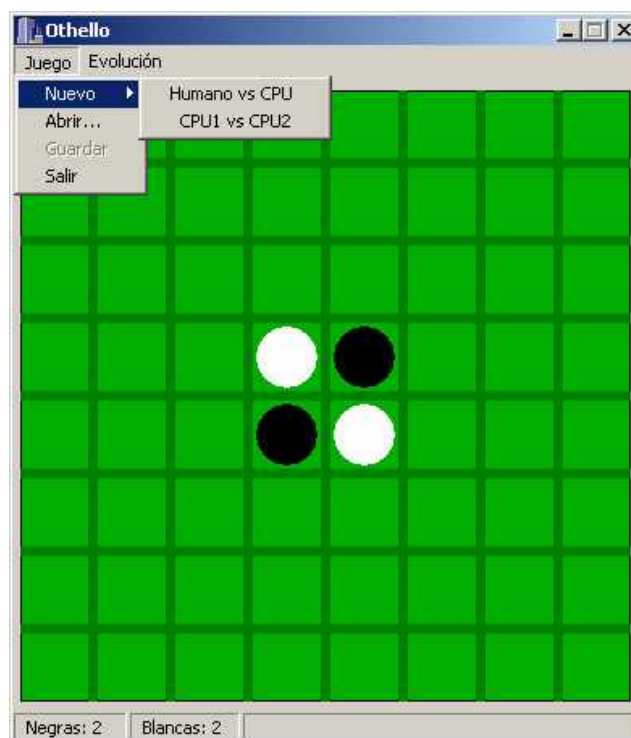
Menú

El menú esta compuesto por dos elementos Juego y Evolución.

Juego

Esta opción tiene que ver con jugar un nuevo juego de Othello, continuar un juego que previamente se almaceno, guardar el estado del juego actual (si hay) o salir del programa.

Nuevo. Se tienen dos alternativas *Humano vs CPU* o *CPU1 vs CPU2*.



En la opción *Humano vs CPU* una persona puede jugar contra el CPU: se elige el *jugador* que utilizará el CPU presionando esta opción y a continuación se elige un archivo de tipo *.evl (mas adelante se describe este tipo de archivos). Hecho lo anterior se puede empezar a jugar

Para la alternativa *CPU1 vs CPU2* se escoge un jugador para cada CPU presionando esta opción. A continuación se muestra el estado final del juego entre ambos jugadores elegidos.

Abrir. Esta opción sirve para cargar un juego o tablero previamente almacenado. Solo se puede jugar la modalidad *Humano vs CPU*. Cuando se presiona esta acción (Abrir...) se muestra

un cuadro de diálogo del cual se debe escoger un archivo de tipo *.tab. Una vez que se elige el archivo se escoge de un cuadro de diálogo un jugador (archivo *.evl) para el CPU.

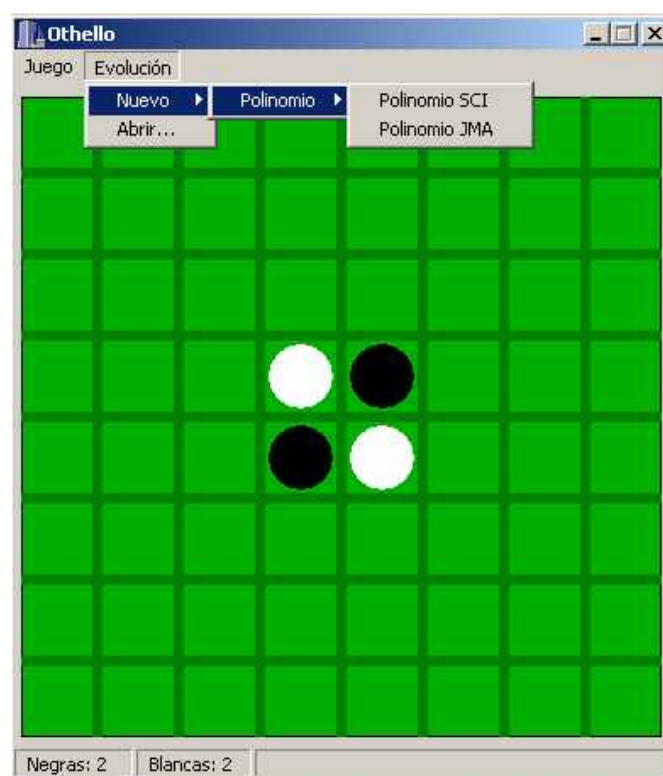
Nota: Cuando se juega en modo Humano vs CPU el primero jugara con las fichas de color negro y el CPU con las blancas.

Guardar. Si se tiene un juego activo y sin terminar esta opción estará disponible para poder guardar dicho estado del juego (tablero). Una vez elegida aparecerá un cuadro de diálogo para escoger un archivo *.tab en el cual guardar el estado (tablero).

Salir. Sirve para terminar la aplicación.

Evolución

En esta parte se puede evolucionar un polinomio desde cero o uno que ya exista de tipo SCI o JMA (que representan tipos de jugador).



Polinomio SCI o JMA. En cualquiera de los dos casos se crea una evolución de polinomios nueva según el tipo que se haya escogido. Una vez elegido el tipo de evolución se presenta un diálogo que pregunta el número de generaciones para la evolución y a continuación se debe dar

un nombre de archivo *.evl en el cual guardar dicha generación. Hecho esto comenzará la evolución que puede tardar varios minutos.

Archivos *.evl

Almacenan poblaciones de algoritmos evolucionados o una función de evaluación. A continuación se describen los formatos:

En la figura de la derecha se muestra una población evolucionada de polinomios de tipo SCI.

Primera línea:

1: tipo población (polinomio)

1: tipo polinomio (SCI)

Segunda línea

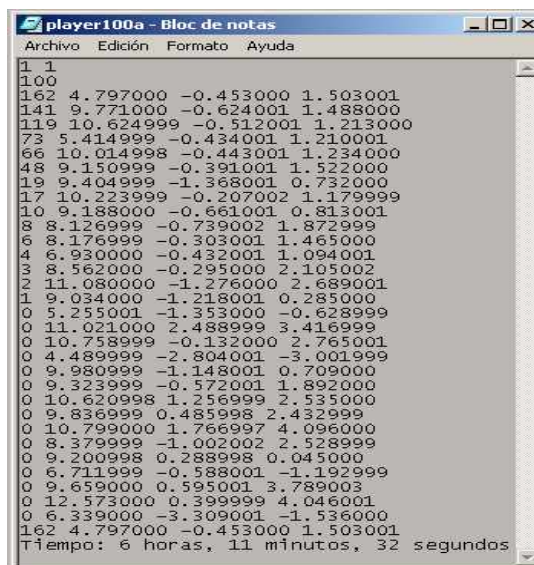
100: número de generaciones de esta evolución.

Siguientes 31 líneas

Cada línea es un polinomio, el primer valor es su fitness y los otros parámetros corresponden a sus coeficientes.

El último polinomio es el de mayor fitness. En la última línea aparece el tiempo en el que se desarrolló esta evolución.

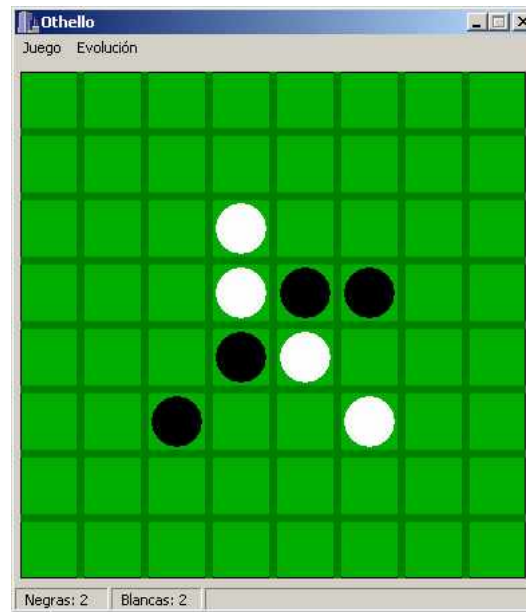
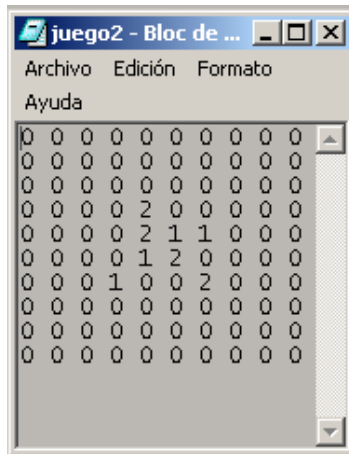
Nota: existen archivos *.evl que también contienen poblaciones de polinomios de tipo JMA, la única diferencia es que estos polinomios tienen 10 variables en lugar de 3.



```
1 1
100
162 4.797000 -0.453000 1.503001
141 9.771000 -0.624001 1.488000
119 10.624999 -0.512001 1.213000
73 5.414999 -0.434001 1.210001
66 10.014998 -0.443001 1.234000
48 9.150999 -0.391001 1.522000
19 9.404999 -1.368001 0.732000
17 10.223999 -0.207002 1.179999
10 9.188000 -0.861001 0.813001
8 8.126999 -0.739002 1.872999
6 8.176999 -0.303001 1.465000
4 6.930000 -0.432001 1.094001
3 8.562000 -0.295000 2.105002
2 11.080000 -1.276000 2.689001
1 9.034000 -1.218001 0.285000
0 5.255001 -1.353000 -0.628999
0 11.021000 2.488999 3.416999
0 10.758999 -0.132000 2.765001
0 4.489999 -2.804001 -3.001999
0 9.980999 -1.148001 0.709000
0 9.323999 -0.572001 1.892000
0 10.620998 1.256999 2.535000
0 9.836999 0.485998 2.432999
0 10.799000 1.766997 4.096000
0 8.379999 -1.002002 2.528999
0 9.200998 0.288998 0.045000
0 6.711999 -0.588001 -1.192999
0 9.659000 0.595001 3.789003
0 12.573000 0.399999 4.046001
0 6.339000 -3.309001 1.536000
162 4.797000 -0.453000 1.503001
Tiempo: 6 horas, 11 minutos, 32 segundos
```

Archivos *.tab

Contienen el estado de un juego no finalizado. Es un archivo que contiene 100 valores (0, 1 o 2), en un formato de 10 renglones por 10 columnas. El tablero de 8x8 casillas se obtiene quitando la primera y ultima columna así como también el primer y último renglón. El valor 0 significa que la casilla está vacía, el número 1 indica que el jugador tiene una ficha en esa posición y con el número 2 se marcan las fichas del oponente.



Resultados

Para los experimentos realizados con Othello7 se utilizó una PC con Pentium4 de 1.4GHz, 256 Mb en RAM.

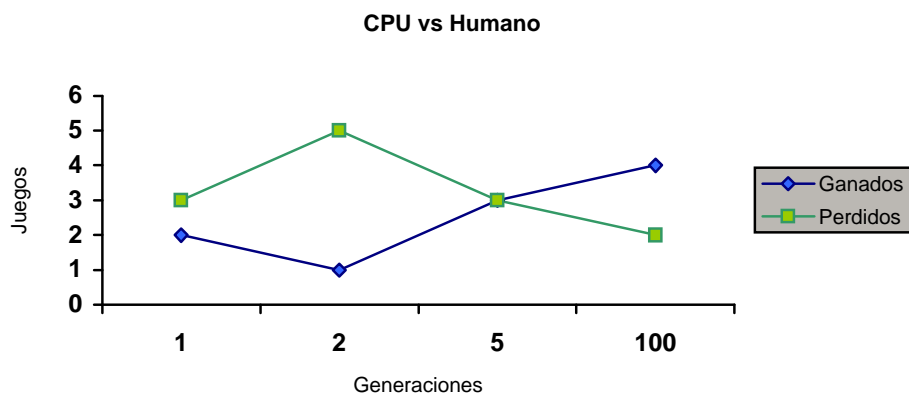
Evolución

Para polinomios de tipo SCI se evolucionaron poblaciones a 1, 5 y 100 generaciones.

Archivo	Generaciones	Fitness mejor polinomio	Tiempo evolución (hh:mm:ss)
player1aSCI.evl	1	3	00:03:05
player1bSCI.evl	1	5	00:03:39
player1cSCI.evl	1	6	00:03:26
player1dSCI.evl	1	3	00:03:21
player2aSCI.evl	2	9	00:06:44
player2bSCI.evl	2	9	00:07:11
player5aSCI.evl	5	36	00:17:27
player5bSCI.evl	5	11	00:15:30
player100aSCI.evl	100	162	06:11:32
player100bSCI.evl	100	141	06:18:29

Juegos Humano vs CPU

En la siguiente gráfica se presentan los resultados obtenidos de encuentros entre humanos y el CPU.



Juegos CPU vs CPU

Se realizaron juegos entre los siguientes archivos evolucionados:

Una generación

CPUa: player1aSCI.evl

CPUb: player1bSCI.evl

CPUc: player1cSCI.evl

CPUD: player1dSCI.evl

Dos generaciones

CPUe: player2aSCI.evl

CPUf: player2bSCI.evl

Cinco generaciones

CPUg: player5aSCI.evl

CPUh: player5bSCI.evl

Cien generaciones

CPUi: player100aSCI.evl

CPUj: player100bSCI.evl

CPU1/CPU2	CPUa	CPUb	CPUc	CPUD	CPUe	CPUf	CPUg	CPUh	CPUi	CPUj
CPUa	-	47/17	12/51	54/10	21/43	36/28	0/45	40/24	0/53	11/53
CPUb	63/1	-	24/40	60/4	21/43	36/28	0/45	40/24	0/48	11/53
CPUc	61/1	61/1	-	61/1	19/45	32/32	0/38	17/47	5/59	24/40
CPUD	36/28	36/28	16/48	-	21/43	36/28	0/45	49/15	0/53	20/44
CPUe	50/0	50/0	54/10	50/0	-	25/39	0/34	31/33	7/56	2/58
CPUf	17/47	24/40	20/44	25/39	37/27	-	0/45	47/17	55/8	16/48
CPUg	31/33	24/40	48/16	30/34	32/32	47/17	-	20/44	24/40	28/36
CPUh	0/40	0/40	31/33	43/21	47/17	0/40	28/36	-	36/28	18/46
CPUi	42/22	39/25	37/27	42/22	40/24	42/22	24/40	47/17	-	52/12
CPUj	4/30	4/30	54/10	4/30	54/10	50/0	58/5	49/15	41/23	-

En la tabla se muestran los resultados de los juegos en donde CPU1 es el primer jugador (fichas negras) y CPU2 el segundo (fichas blancas). Los resultados tienen la notación fichas CPU1\fichas CPU2. Las casillas con fondo gris son resultados que ganan los cpu's que juegan con las fichas negras, es decir cuando juegan como CPU1.

La archivo CPUi jugó un encuentro contra el programa Zebra ver. 1.0 para Pocket PC, únicamente en el nivel 1 (el más bajo).

Conclusiones

Los resultados muestran un regular grado de consistencia de las evoluciones con más generaciones con respecto a otras menos evolucionadas. Para estas poblaciones se pensó en pocos parámetros para evaluar el tablero (polinomios tipo SCI). Se dieron excepciones en encuentros entre jugadores avanzados y principiantes (en cuanto a generaciones) ganando éstos últimos.

Se notó que el grado de aprendizaje de los parámetros que se introdujeron en los polinomios de las poblaciones fue mayor en las evoluciones con mayor número de generaciones.

Al jugar una persona contra las evoluciones más avanzadas se puede decir que el programa aprendió a jugar a un nivel de principiante.

Esta aplicación además de ser un programa más que juega Othello, tiene varias posibilidades de crecimiento como son la posible incorporación de poblaciones de redes neuronales, de polinomios que evalúen otros factores de un tablero y evolucionarlos como los que ya se cuentan (SCI y JMA).

Bibliografía

El Lenguaje de Programación C++
Stroustrup Bjarne
Addison Wesley

Singer, Joshua A.
Combining reinforcement learning with genetic algorithms to produce an Othello-playing program
September 1, 2000

Moriarty, David & Miikkulainen, Risto
Evolving complex Othello strategies using marker-based genetic encoding of neural networks
Department of Computer Sciences
University of Texas at Austin, Austin

Leouski, Anton
Learning of position evaluation in the game of Othello
January 20, 1995
Department of Computer Sciences
University of Massachusetts

Alliot, Jean-Marc & Durand, Nicolas
A genetic algorithm to improve an Othello program
Ecole Nationale de l'Aviation Civile
Centre d'Etudes de la Navigation Aérienne

Leouski, Anton V. & Utgoff, Paul E.
What a neural network can learn about Othello
January 20, 1996
Department of Computer Sciences
University of Massachusetts

Sitios de interés

<http://www.satirist.org/learn-game/systems/othello/>

<http://www.nada.kth.se/~gunnar/zebra.html>

<http://www.sop.inria.fr/cafe/Olivier.Arsac/darwersi/>

<http://www.cam.org/~bigjeff/Hannibal.html#what>

<http://www.neci.nj.nec.com/homepages/mic/log.html>

<http://www.cs.ualberta.ca/~games/keyano/>

<http://perso.wanadoo.fr/brunodlb/ushomepg.htm>

<http://www.mathjendl.org/reversi.html>

<http://federation.othello.free.fr/41.htm>

<http://home.swipnet.se/~w-50714/othello/tutorial/intro.htm>