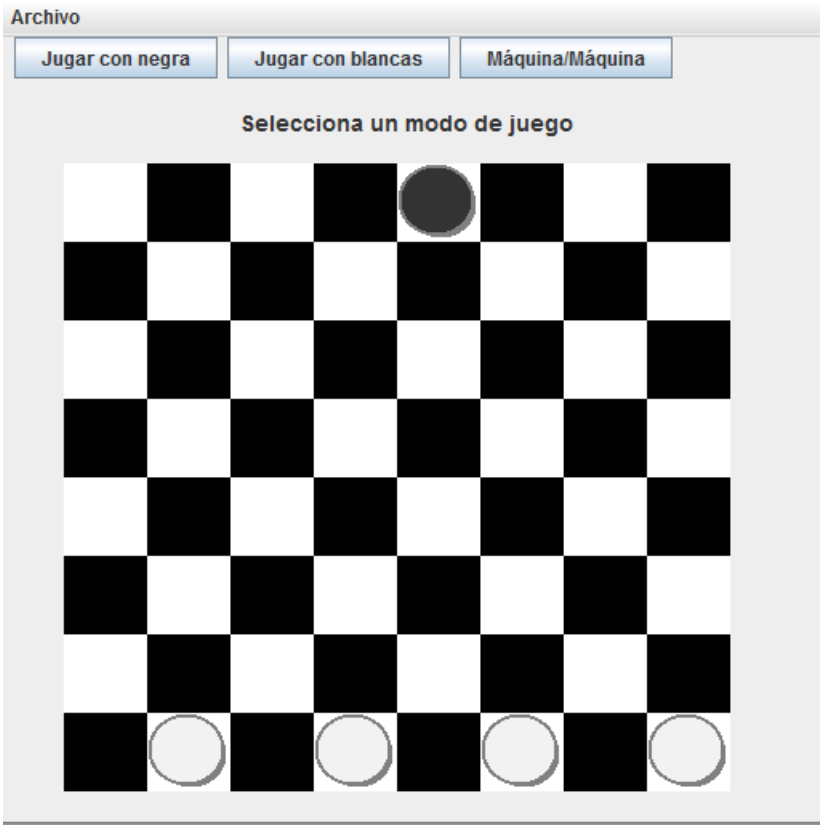
GATOS Y RATÓN

INTRODUCCIÓN

Gatos y Ratón es un juego que se desarrolla sobre un tablero de ajedrez donde se colocan cuatro fichas blancas, los gatos, en un extremo y una ficha negra en el extremo opuesto, el ratón.

El objetivo del ratón es conseguir llegar al otro extremo, mientras que el objetivo de los gatos es atrapar al ratón.

Se pide implementar el algoritmo alfa-beta de búsqueda en juegos para calcular la mejor jugada.



ALGORTIMO ALFA-BETA

La implementación del algoritmo alfa-beta ha girado en torno a la clase Nodo. La clase Nodo es la que contiene toda la información de las distintas jugadas y la que las evalúa.

```
class Nodo
{
Tablero tablero;
int alfa;
int beta;
int profundidad;
Movimiento movimiento;
int jugador;
}
```

Cuando un Nodo se crea se le asigna el valor de alfa y beta de su padre. También se le pasa por parámetro al constructor el tablero, el movimiento que se debe ejecutar y el jugador que ejecuta el movimiento. Es en el constructor cuando se hace la tirada que indica el movimiento y modifica el tablero.

El algoritmo alfa-beta es un algoritmo recursivo que recibe la información de una jugada (en forma de Nodo) y devuelve un entero que puede ser alfa o beta.

- Se comprueba si es un nodo hoja, con la función `isNodoHoja()` del Nodo.
  - Si es un nodo hoja se devuelve el valor de la función de evaluación. El valor se obtiene llamando a `evaluarNodo(jugador)`.
  - Si no es nodo hoja:
    - Obtenemos todas las posibles jugadas del jugador que juega en este instante con `getMovimientos()`.
    - Dependiendo si es nodo MAX o si es un nodo MIN recorremos todos los posibles movimientos generando los nuevos hijos y quedándonos con el mejor o peor valor. Teniendo en cuenta las condiciones de poda y si es un nodo MAX o MIN devolvemos el valor de alfa o beta.
    - En caso de que el nodo sea el Nodo Raíz modificamos el valor de su movimiento (que en principio es `null`) para asignarle el movimiento a ejecutar.

La primera llamada al algoritmo recursivo es la siguiente:

Se crea el nodo raíz, con el movimiento a ejecutar a null. Además alfa y beta se han inicializado a -9999 y 9999 respectivamente.

```
Nodo nodo = new Nodo(m_tablero, alfa, beta, 0, null, m_jugador);
```

A continuación se llama al método recursivo.

```
alfabeta(nodo);
```

Por último se almacena el valor del movimiento en la variable `m_movimiento`

```
m_movimiento = nodo.movimiento;
```

FUNCIÓN DE EVALUACIÓN

La función de evaluación es distinta para las blancas y para la negra.

He implementado diferentes métodos que he ido combinando para obtener una mejor función de evaluación.

Función	Descripción
int distanciaNegra()	Indica la distancia en filas a la que se encuentra la ficha negra de llegar al final, y por tanto de ganar.
int centradoNegras()	Devuelve un valor que depende de si la ficha negra está o no centrada (en columnas) en el tablero. -4 si no lo está, 2 si está poco centrada y 5 si está en las columnas centrales.
float distanciaBlancas()	Devuelve la distancia media de la negra a las blancas que tiene por debajo.
int distanciaBlancasCorta()	Devuelve la distancia de la ficha blanca más cercana a la negra.
int distanciaEntreBlancas()	Indica la distancia entre las blancas en filas. Para saber si están muy separadas o formando una fila.
int movimientosNegras()	Devuelve el número de movimientos posibles que puede ejecutar la ficha Negra.
boolean isWin(int jugador)	Devuelve true si el jugador está en una situación en la que tiene asegurada la victoria. (Pero no ha ganado aún).
boolean finWin(int jugador)	Devuelve true si el jugador está en una situación de victoria.
int ladoCorrectoNegra()	Indica si la negra está en el lado (izquierda o derecha) donde menos blancas hay.
boolean situacionGanadora(int jugador)	Indica si un jugador está en una situación ganadora. Tiene la victoria asegurada
int blancasPorEncima()	Devuelve el número de fichas blancas que la negra tiene por encima (en filas menores).
boolean caminoDeNegra()	Indica si la negra puede tener un camino de paso. Se calcula si la distancia en filas de cada ficha es mayor o igual que la distancia en columnas.
boolean isLose(int color_jugador)	Devuelve true si el jugador está en una situación en la que tiene asegurada la derrota. (Pero no ha perdido aún).
boolean finLose(int color_jugador)	Devuelve true si el jugador está en una situación de derrota.

Como he comentado he ido combinando las distintas funciones asignándoles distintos pesos para conseguir una mejor función de evaluación.

En primer lugar se evalúa si la situación del tablero es una situación de victoria (se devuelve 1000) o una situación de derrota (se devuelve -1000). Esta comprobación se hace en la función de evaluación de las blancas y de la negra.

Las funciones de evaluación de ambos jugadores han pasado por distintos estados.

Para la negra en primer lugar ponderé las funciones de la siguiente manera:

- Cercanía de la negra al objetivo (30%).
- El número de Blancas por encima (20%).
- Los movimientos posibles de la Negra (20%).
- Distancia de a la blanca más cercana (20%).
- Distancia media a las blancas (10%).

Quedando la función de evaluación de la siguiente manera.

```
valor = (8 - distanciaNegra()) * 40;
valor += blancasPorEncima() * 20;
valor += movimientosNegras() * 20;
valor += distanciaBlancasCorta() * 20;
valor += distanciaBlancas() * 10;
```

Tras probar con distintas combinaciones acabé añadiendo la comprobación de que la ficha Negra estuviera centrada. (Aunque tiene mucho que ver con los movimientos disponibles que le quedan a la negra, porque al estar centrado tiene 4 posiciones donde moverse y estando en los extremos solo dos). Además le resté importancia a la distancia a la meta.

```
valor = (8 - distanciaNegra())*3;
valor += blancasPorEncima() * 3;
valor += movimientosNegras() * 4;
valor += centradoNegras();
valor += distanciaBlancas();
```

Y así es como quedó la configuración definitiva de la función de evaluación. Simplifiqué bastante la ponderación. Esta función de evaluación es la que mejores resultados dio. Además de que es la que hacía movimientos más humanos (por ejemplo, no avanzaba y retrocedía en las mismas posiciones).

Lo que más se bonifica es la distancia a la meta, el número de blancas que tiene por encima (y por tanto no le pueden cortar el paso) y los movimientos posibles que tiene para la siguiente jugada. También tenemos en cuenta que esté centrada, aunque se valora menos y la distancia media a las fichas blancas que tenga por debajo (no entran en el cómputo de la distancia las fichas blancas por encima de la negra).

La función de evaluación de las blancas ha sido la más complicada. Al principio le di más importancia al hecho de que la ficha Negra esté lo más alejada posible de ganar. Pero al final acabó siendo mejor que las blancas se movieran sin separarse muchos en filas.

Al principio la función de evaluación era así:

```
valor = distanciaNegra() * 30;
valor -= blancasPorEncima() * 40;
valor -= movimientosNegras() * 40;
valor -= ladoCorrectoNegra() * 20;
valor -= distanciaBlancas() * 10;
valor -= centradoNegras() * 20;
```

Y al final acabó siendo de la siguiente manera:

```
valor = distanciaNegra();
valor -= distanciaEntreBlancas();
valor -= distanciaBlancas();
valor -= movimientosNegras();
valor += (4 - blancasPorEncima()) * 3;
```

En esta función de evaluación se penaliza en vez de bonificar para aprovechas las funciones hechas para la Negra.

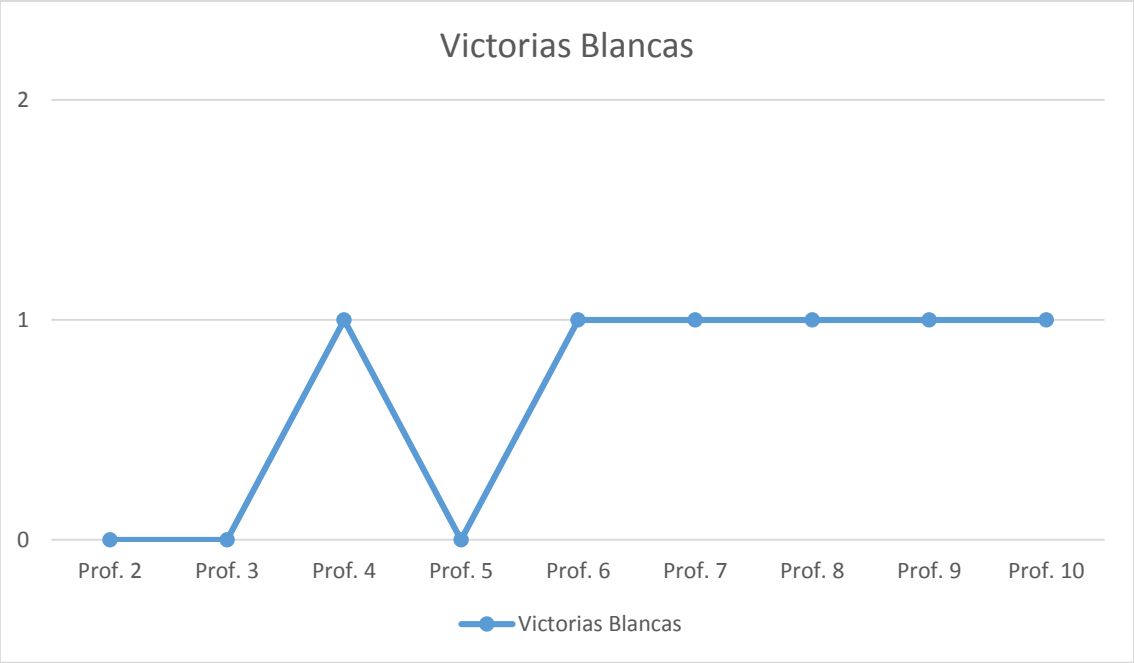
Se penaliza mucho (se multiplica por 3) que haya mucho espacio entre las fichas blancas en filas, así como que haya fichas blancas por encima de la Negra. Las fichas Blancas que estén por encima de la Negra ya no le pueden cortar el paso, por tanto los Gatos tendrían menos efectivos para acorralar al Ratón.

Además se ha tenido menos en cuenta la distancia de la Negra a la meta.

EXPERIMENTACIÓN

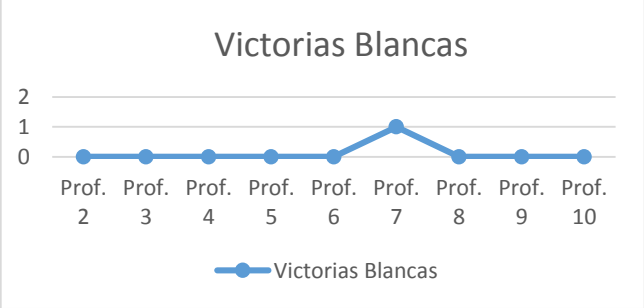
Como se ha comentado antes las pruebas han sido varias durante la realización de la práctica. Gracias a las pruebas se han extraído datos como por ejemplo que es mejor para las blancas mantener una formación en filas que intentar acercarse mucho a la negra. Las pruebas se han realizado variando la ponderación en la función de evaluación.

Por otra parte faltaba realizar las pruebas sobre la profundidad del árbol. En general en las pruebas a distintas profundidades el comportamiento de la negra es aceptable, pero el de las blancas cambia bastante, así que vamos a hacer un estudio sobre las victorias sobre 10 partidas de las fichas blancas.



Como se puede observar en el gráfico las blancas ganan 1 de cada 10 partidas a profundidad 4, 6, 7, 8,9 y 10. Por motivos de rendimiento y gracias a los datos recopilados vamos a establecer la profundidad en 8.

Tras cambiar el funcionamiento de la asignación del movimiento en el alfa-beta y añadir un valor random que indica si cambiamos el movimiento actual por el anterior cuando su valor de alfa es igual se obtienen los siguientes resultados:



Se han mejorado los comportamiento de los jugadores, lo que pasa que ha aumentado la diferencia entre la habilidad de la negra y de las blancas.

Nota: Antes cuando se mejoraba el valor de alfa se hacía un random para sustituir o no el movimiento anterior por el movimiento mejor. Esto posibilitaba no quedarte siempre con el mejor, sino con un bueno. Al cambiar esto hacer que siempre se quede con el mejor movimiento la habilidad de la negra es muy superior a la de las blancas.

RESULTADOS

El general los resultados de la ficha Negra son bastantes buenos, es capaz de ganar a las blancas en un gran porcentaje de veces. Las blancas suelen mantener la formación, y tapan los posibles caminos de la Negra, pero por lo general la Negra suele ganar.