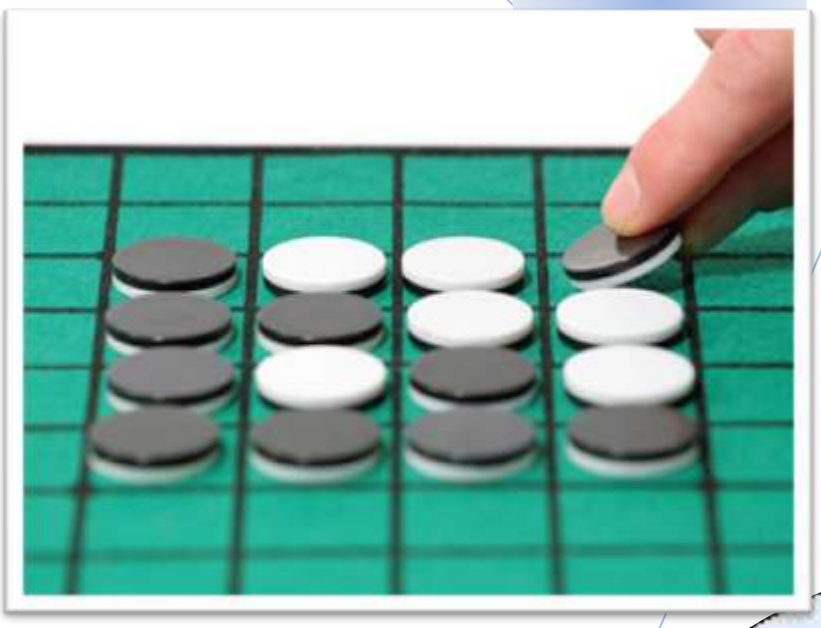
UNIVERSIDAD DE JAÉN

ESCUELA POLITÉCNICA SUPERIOR DE JAÉN

Departamento de Informática



Inteligencia Artificial

Práctica 3. Juego Adversarios Min-Max

Segundo Curso del Grado en Ingeniería Informática

Curso 2015-16

Práctica 3. Juego entre Adversarios usando Min-Max. Othello (Reversi)

1. Introducción

Othello (también llamado ***Reversi***) es un conocido juego de mesa donde se utiliza un tablero de *nxn* celdas idénticas (normalmente es de *8x8*, pero se pueden encontrar escenarios de distintos tamaños) en el que dos jugadores van colocando fichas sobre el tablero alternativamente (una ficha por turno). Todas las fichas son exactamente iguales, y cada una tiene dos caras, una de color blanco y otra de color negro. Durante la partida, si un jugador encierra (en línea recta: vertical, horizontal o diagonal) una serie de fichas del color del oponente entre dos de sus fichas, las primeras son capturadas, volteándose para tomar el color del jugador.

Más formalmente, las reglas son las siguientes:

- Al inicio del juego, se sitúan 4 fichas (2 de cada jugador) en el centro del tablero, tal y como se muestra en la figura 1. El jugador con las fichas negras mueve primero.

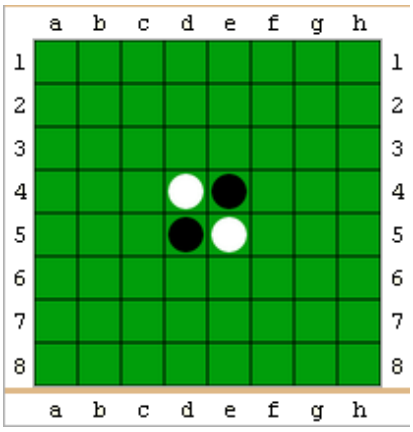


Figura 1. Estado inicial del juego

- En cada turno, cada jugador debe colocar una ficha de su color en el tablero, de forma que exista al menos una línea recta (vertical, horizontal o diagonal) entre la nueva ficha y una ficha del mismo color, con una o más fichas del otro jugador entre medias. Por ejemplo, la figura 2 muestra las posibles opciones que tendría el jugador Negro como jugada inicial.

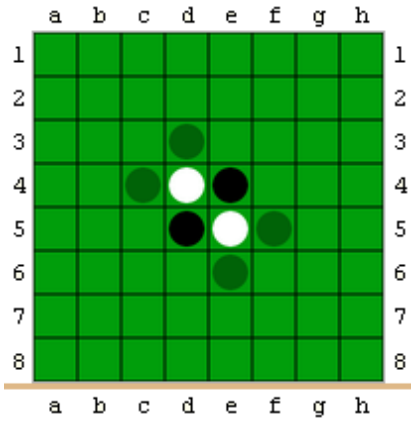


Figura 2. Posibles movimientos para el jugador Negro

- Tras la jugada, todas aquellas fichas rivales en línea recta entre la nueva ficha y cualquier ficha del jugador son “capturadas”, y se les da la vuelta, pasando a ser propiedad del jugador que realiza la jugada, que puede usar en próximos turnos, tal y como se muestra en la figura 3. Dicho de otra forma, una jugada legal es aquélla en la que al menos se captura una ficha rival.

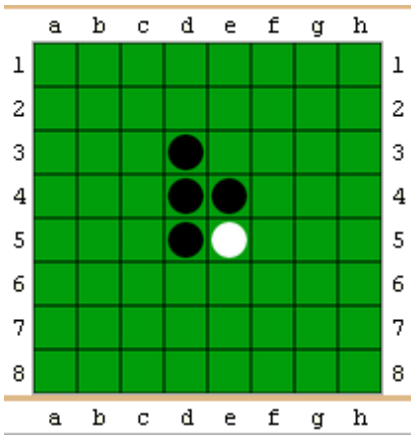


Figura 3. Estado del juego tras el movimiento del jugador Negro

- A continuación, el otro jugador puede llevar a cabo un movimiento similar con las fichas de su color.
- Si un jugador no pudiera hacer ningún movimiento legal, salta el turno al otro jugador.
- El juego termina cuando el tablero se completa, o ninguno de los jugadores puede realizar ningún movimiento legal (puede ocurrir que queden celdas vacías, pero inalcanzables).
- El ganador del juego es aquél que, al final de la partida, tiene mayor número de fichas de su color.

2. Objetivos

Con la presente práctica se pretenden que el alumno comprenda el funcionamiento de los algoritmos **min-max** y **poda alfa-beta** para búsqueda en juegos, desarrollando y probando una función de evaluación para un juego en concreto, en nuestro caso el **Othello**.

3. Características

Se proporciona un proyecto en **Netbeans** (ver árbol de clases en la Figura 4) con el entorno gráfico ya programado.

Como en prácticas anteriores, crearemos un proyecto vacío (sin clase principal), y sustituiremos la carpeta ***"src"*** con la contenida en el archivo **othelloUJA.zip** que se descarga desde ILIAS.

En el árbol de clases resultante destacan las siguientes:

- **Main.java** en el paquete **othello.gui**. Es ésta la que debemos elegir como clase principal, ya que es la encargada de iniciar la interfaz gráfica y poner el juego en funcionamiento.
- Paquete **othello.Utils**. Contiene las clases **Casilla.java**, **Heuristica.java** y **Tablero.java**, correspondientes a los elementos del juego a los que tendremos acceso a la hora de llevar a cabo la partida.
- Paquete **othello.jugador**. Contiene la clase abstracta **Jugador**, de la cual heredan las clases **JugadorHumano** y **JugadorMaquina**. Con la primera no hay que hacer nada, puesto que en ese caso el programa nos deja mover libremente las fichas (él controla que sean movimientos legales). La segunda clase hace uso de las clases del paquete **othello.algoritmo**, las que nos interesan.
- Paquete **othello.algoritmo**. En él se encuentran las clases que heredan de la

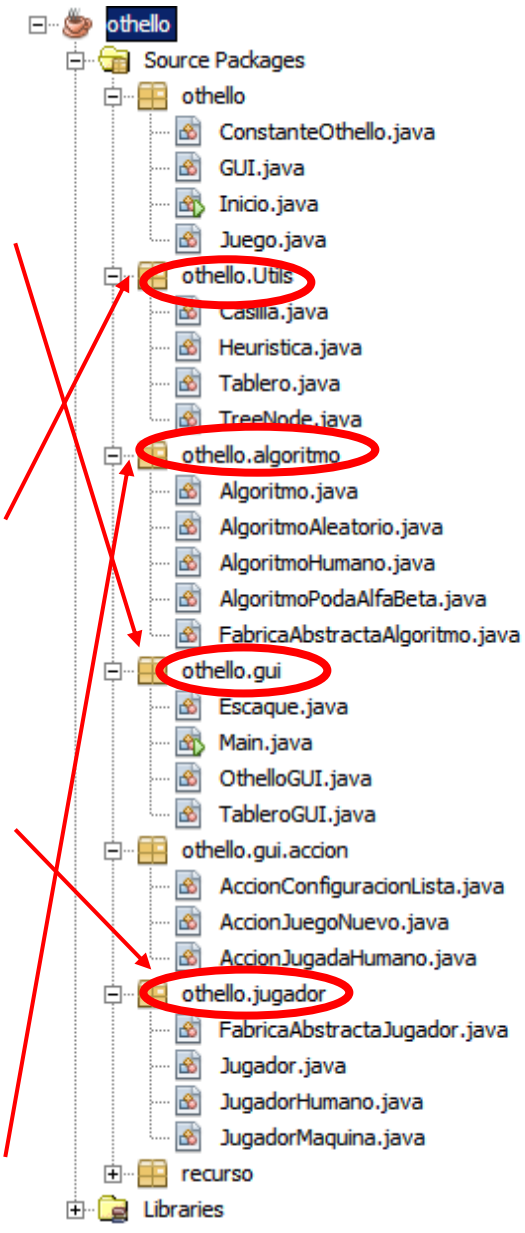


Figura 4. Árbol de clases

clase abstracta **Algoritmo**, y ellas se definen los diferentes cursos de acción en cada turno. Como ejemplo para nuestra implementación, podemos partir de la clase **AlgoritmoAleatorio**, la cual nos muestra cómo, cuando le toca el turno al jugador, se elige al azar una casilla del tablero que esté libre y a la que se pueda acceder mediante un movimiento válido.

Dentro del paquete anterior, se hallan la clases **AlgoritmoPodaAlfaBeta** y **AlgoritmoMiniMax**, que será con las que tengamos que trabajar en la presente práctica. Concretamente, tendremos que modificar los métodos **minimax** y **alfaBeta**, inicialmente vacíos, para que incluya la implementación de los correspondientes algoritmos considerados.

Otra clase interesante es la clase **Heuristica** (paquete **othello.Utils**). Como métodos de la misma se pueden implementar diferentes heurísticas que el jugador Máquina pueda aplicar en su beneficio. Se proporcionan dos ejemplos, **h1** (que no hace nada), y **h2**, que usa un valor basado en la diferencia de piezas ganadas entre cada rival. Podemos idear una nueva heurística que obtenga resultados más ventajosos para el jugador e incluirla en esta clase como un método más.

El proyecto proporcionado trae consigo una interfaz gráfica mediante la que podemos ver cómo se va desarrollando el juego. El aspecto de la aplicación gráfica es el que se muestra en la siguiente imagen. Podemos iniciar una nueva partida pulsando sobre el botón ***"Jugar"***.

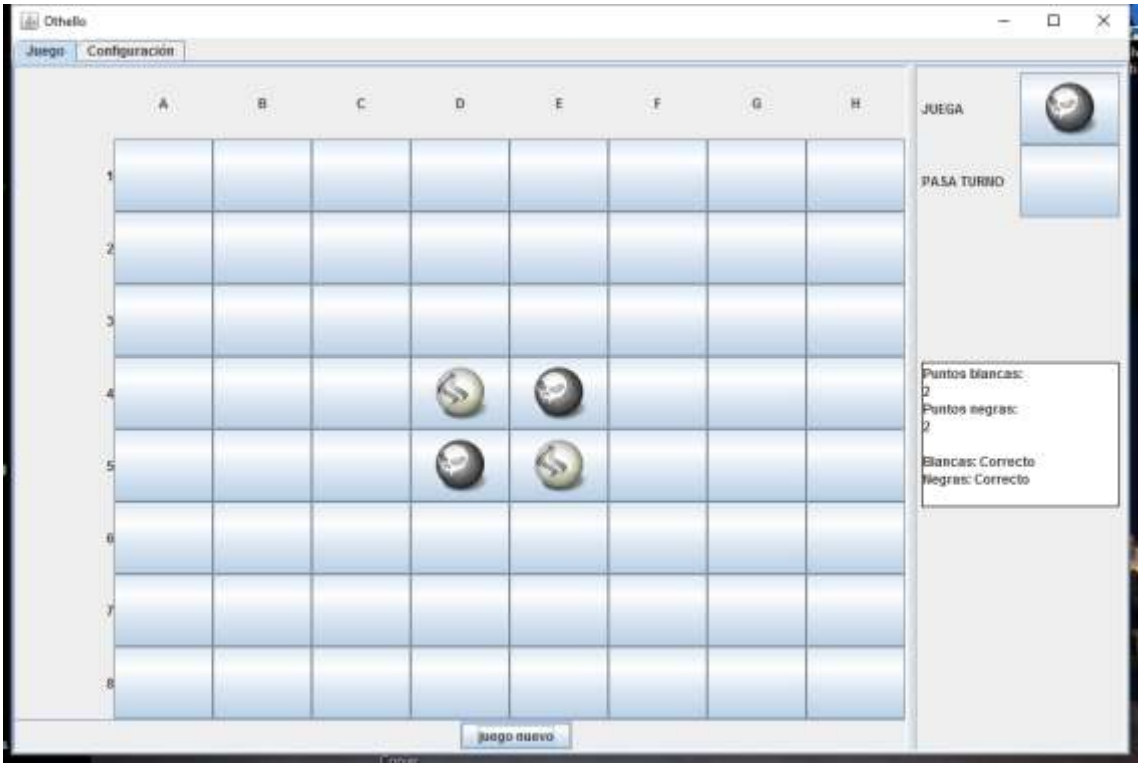


Figura 5. Ventana inicial de juego

Previamente, podemos configurar el juego seleccionando entre las opciones disponibles en la pestaña ***"Configuración"***: Tipo de jugadores (*Humano/Máquina*), algoritmo (*AlfaBeta, MiniMax o Aleatorio*, en nuestro caso) y profundidad máxima de exploración. Una consideración importante es que, para evitar situaciones anómalas, es conveniente NO modificar la configuración del juego una vez haya comenzado la partida.

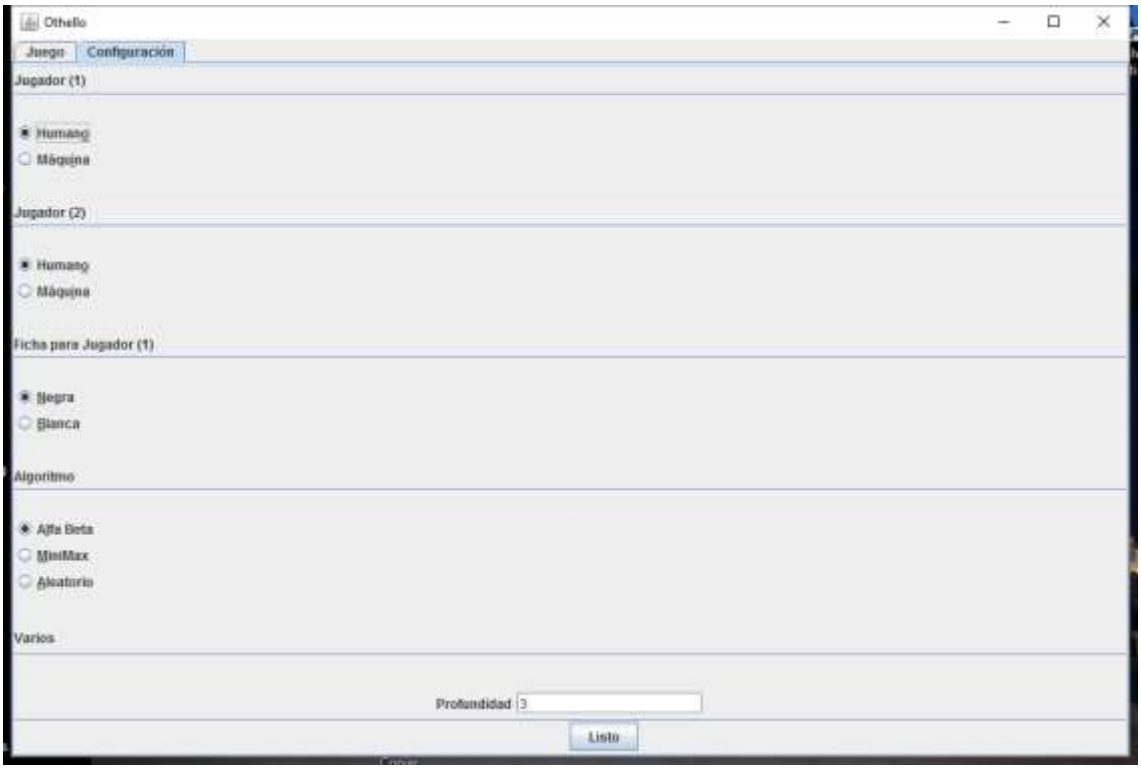


Figura 6. Ventana de configuración de la partida

4. Tareas

- Los alumnos tendrán que implementar el algoritmo **MiniMax**, estableciendo una profundidad máxima de exploración (prof = 5), y probar su funcionamiento jugando contra la Máquina durante una serie de partidas.
- En segundo lugar, hay que implementar el algoritmo de poda **Alfa-beta**, e integrarlo dentro del juego, para que sea empleado por los jugadores Máquina durante una serie de partidas. Al igual que con el algoritmo anterior, los alumnos deberán testearlo jugando contra la máquina, y además, comparar el funcionamiento de ambos algoritmos, enfrentándolos entre sí en una batería de pruebas.

- Además, los alumnos deberán plantear e implementar una función heurística que, de manera justificada, busque ayudar al jugador en la elección de su siguiente movimiento.

5. Análisis del comportamiento de los jugadores.

Con objeto de evaluar el comportamiento conjunto de los diferentes algoritmos y estrategias (heurísticas), ante diferentes adversarios (humanos o máquinas), vamos a llevar a cabo una batería de diferentes partidas, y recogeremos en tablas como la que aparece a continuación el resultado de las mismas:

Ejecución	Jugador1	Algoritmo1	Heurística1	Jugador2	Algoritmo2	Heurística2	Negras	Fichas1	Fichas2
1									
2									
3									
4									
5									

Como mínimo, será necesario recoger resultados de tres tipos diferentes de enfrentamientos:

- ***Jugador humano vs. Jugador máquina (Minimax).***
- ***Jugador humano vs. Jugador máquina (Alfabeta).***
- ***Jugador máquina (Minimax) vs. Jugador máquina (Alfabeta).***

Se jugarán un mínimo de 5 partidas para cada una de las combinaciones anteriores. En la tabla, habrá que rellenar los campos correspondientes:

- **Jugador(1,2)**: Tipo de jugador (Humano o máquina)
- **Algoritmo(1,2)**: En caso de que el jugador sea la máquina, indicar el algoritmo utilizado (Minimax o Alfabeta). Si el jugador es humano, dejar el campo vacío.
- **Heurística(1,2)**: Indicar la heurística utilizada por el jugador máquina (si es humano, dejar vacío). La heurística puede ser alguna de las incluidas en el proyecto, o la implementada por los alumnos.

- **Negras**: Indicar qué jugador empieza la partida.
- **Fichas(1,2)**: Completar con los resultados obtenidos por cada jugador al término de la partida. Como se ha indicado antes, gana aquel jugador con más fichas capturadas.

6. Aspectos que se valorarán en la nota final

- **La explicación o descripción de la estrategia seguida en cada uno de los algoritmos.**
- **La limpieza del código entregado, la documentación interna del mismo, la documentación externa en el informe y el uso correcto de convecciones de Java.**
- **La inclusión, razonada, de nuevas funciones heurísticas en el juego. La función habrá de ser debidamente justificada y se deberá demostrar que ésta funciona.**
- **La práctica se evalúa hasta un máximo de 3 puntos: 1 punto por la implementación del algoritmo min-max, 1 punto para el algoritmo de poda alfa-beta, y 1 punto por la inclusión y aplicación de una nueva heurística.**

7. Entrega y evaluación

- La práctica se realiza por parejas y se entregará comprimida en un único fichero .zip que contenga:
- Los archivos fuente:
    - **AlgoritmoMiniMax.java**, debidamente comentado, que contendrá UNICAMENTE la clase **AlgoritmoMiniMax**, y excepcionalmente, clases auxiliares si se han definido.
    - **AlgoritmoPodaAlfaBeta.java**, con exactamente las mismas consideraciones del primer algoritmo.
    - La función heurística se incluirá como método en el archivo fuente **Heuristica.java**.
- Documentación adicional en formato PDF explicando cómo se ha implementado el alfa-beta, la función de evaluación al completo, la función heurística ideada, y ejemplos ilustrativos y las pruebas que se hayan realizado del algoritmo.
- No se tendrá en cuenta la modificación de ninguna otra clase del entorno proporcionado que no sean las clases **AlgoritmoMiniMax**,

AlgoritmoPodaAlfaBeta o **Heuristica**. Dichas modificaciones se pueden llevar a cabo durante la realización de la práctica para hacer pruebas o familiarizarse con el entorno, pero en la entrega final sólo se aceptarán exclusivamente los archivos .java referidos en el punto anterior.

- En el código que se entregue no está permitido emitir ningún símbolo por la salida estándar ni de error ya que dichas salidas se usan por el juego para evaluar las jugadas durante la ejecución.
- El documento comenzará, además, con una portada con, al menos, la siguiente información: nombre de los autores, curso académico, grupo de prácticas y nombre del profesor de prácticas.
- La fecha tope oficial para la entrega de la práctica 3 es el **martes 3 de mayo, a las 23:59**. El trabajo entregado se defenderá en clase de prácticas el día **4 de mayo**. **La defensa de las prácticas es obligatoria para todos los alumnos**.