Grado en Ingeniería Informática

Inteligencia Artificial

Curso 2022/2023

Universidad de Jaén

Guión 3: Juegos

1. Juego Min-Max entre adversarios

1.1. Introducción

Conecta-4 (también conocido como **4 en Raya**) es un juego para dos jugadores donde éstos introducen, por turnos, fichas en un tablero vertical con el objetivo de alinear cuatro fichas de un mismo color de forma consecutiva (bien en horizontal, vertical o en diagonal). El tablero original está formado por 6 filas y 7 columnas (Figura 1), aunque existen otras variaciones con diferentes tamaños.

Cada jugador dispone de un conjunto de fichas de un color determinado y, alternativamente, debe introducir una ficha por una de las columnas disponibles (es decir, que aún no estén completas), cayendo ésta hasta la posición más baja. Gana el juego el primer jugador que consiga alinear cuatro fichas consecutivas de su color. Si el tablero se completa sin que ningún jugador haya logrado su objetivo, el juego termina en empate.

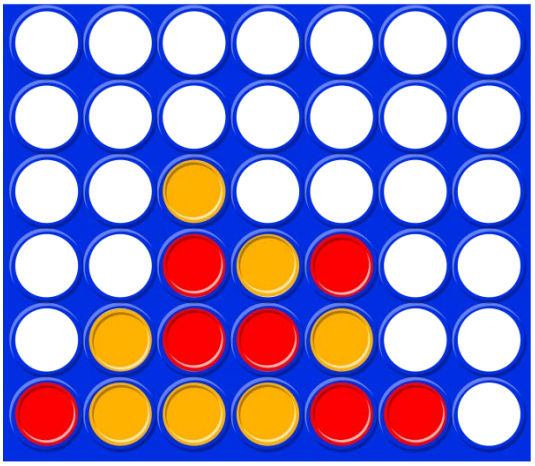

Figura 1. Conecta 4, el juego original (Milton Bradley, 1974)

Conecta-4 es un juego de *"información completa"*, esto es, cada jugador, en su turno, tiene conocimiento de todos los movimientos anteriores, más todos los posibles movimientos que se pueden generar a partir del estado actual. En condiciones ideales de juego perfecto, el jugador que inicia la partida es capaz de ganar siempre ésta, si coloca su primera ficha en la columna central. El juego se resolvió completamente en 1988.

1.2. Características del entorno de juego conecta

- Requiere Java versión 8 o superior. Se puede integrar en **Netbeans** (el entorno no es obligatorio, pero sí recomendable por comodidad) de forma muy sencilla, incorporando el contenido del proyecto, que se encuentra disponible en la plataforma ILIAS comprimido como un archivo .zip.
- Para la práctica vamos a mantener la configuración habitual del juego. Es decir, el tamaño de tablero por defecto al inicio del juego será de 6x7. El número de fichas que habrá que encadenar para ganar la partida será de 4.
- La lógica del programa se ha dividido entre varias clases, la principal, una clase para representar el tablero de juego, y varias clases para representar los distintos tipos de jugadores. El programa permite jugar contra un oponente humano, o bien contra la máquina (con diferentes estrategias según el tipo de jugador). El jugador 1 siempre será el jugador humano y es quien tiene el turno al inicio de la partida.
- El juego muestra en todo momento el estado actual del tablero, conforme se van colocando las fichas en él. Para ello hace uso tanto de una interfaz gráfica (la ventana de la aplicación) como de la salida estándar por pantalla.

1.3. Instalación de conecta

La instalación del proyecto es análoga a la realizada en prácticas anteriores. Desde **NetBeans**, creamos un nuevo proyecto (**Java with Ant**), bajo la opción *"Java Application",* desactivando la casilla *"Create Main Class"*.

Una vez hecho esto, descargamos desde PLATEA el fichero *plot4.zip* con el juego, y lo descomprimimos. Buscaremos la carpeta del nuevo proyecto y allí copiamos/reemplazamos tal cual el contenido del fichero descomprimido (Figura 2).

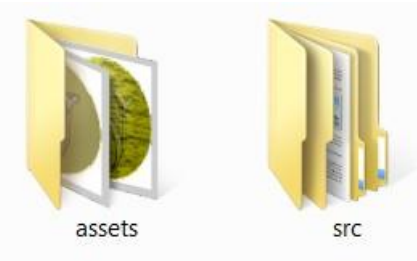

Figura 2. Contenido del archivo conectan.zip

Si lo hemos hecho bien, al volver a NetBeans y tratar de ejecutar el proyecto, nos pedirá la clase que contiene el método *main().* Debemos indicarle la clase *plot4.Main*. La estructura del proyecto se muestra en la figura 3.

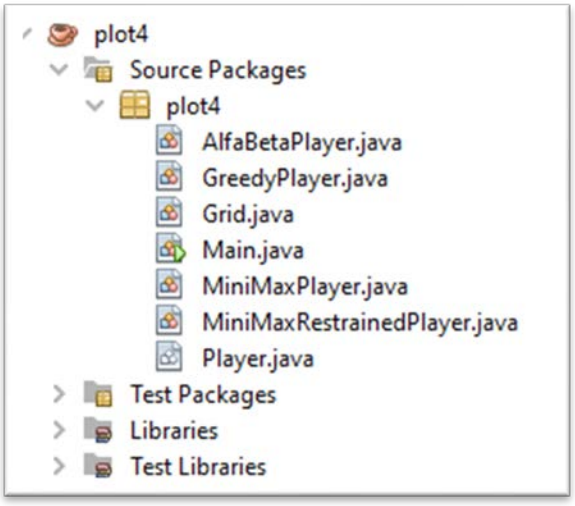

Figura 3. Árbol de clases del proyecto

Una vez configurado, podremos ejecutar el proyecto y podremos comenzar a jugar (Figura 4).

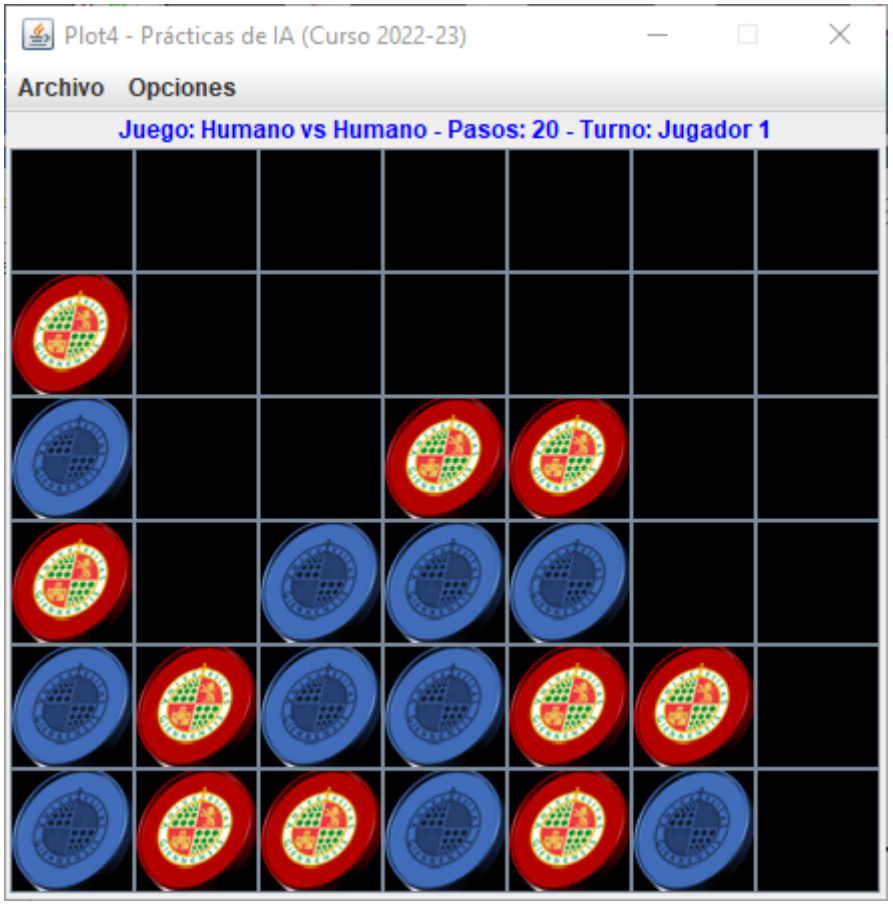

Figura 4. Ejemplo de partida en tablero 6x7

1.4. Funcionamiento del juego

El Jugador 1 (Humano) siempre empieza la partida y juega con las fichas azules.

El Jugador 2 juega con las fichas rojas.

En el menú **Archivo** tenemos solo la opción de **Salir** del programa. Desde el menú **Opciones** podemos configurar el tipo de los jugadores. Por defecto, el Jugador 1 será siempre *Humano*, mientras que podemos elegir si queremos que el Jugador 2 sea *Humano*, *CPU Greedy* (opción por defecto al inicio de la aplicación), y varios agentes que tendremos que diseñar nosotros: *CPU MiniMax*, *CPU MiniMax Restringido*, y *CPU MiniMax AlfaBeta*. Si en el transcurso de una partida, cambiamos el tipo de jugador, ésta se reiniciará con la nueva configuración.

Conforme se vaya desarrollando la partida, cada jugador irá colocando sus fichas alternativamente hasta que, o bien, uno de ellos logre situar el primero el número indicado de fichas consecutivas de su color, o bien se complete el tablero sin que ninguno de los jugadores gane (empate). El fin del juego se indicará mediante una ventana modal como la que muestra la siguiente figura.

Figura 4. Ventana modal de fin de partida

A continuación, comentaremos algunos de los métodos más interesantes dentro de las clases que conforman el proyecto:

- La clase principal de nuestro juego es **Main**, y en ella se encuentra la lógica principal del programa. Mediante la clase **Grid** tenemos acceso a una representación del estado del tablero actual. La clase **Player** es una clase abstracta de la cual heredan las subclases **GreedyPlayer** (la cual implementa un algoritmo sencillo para que la máquina pueda jugar contra un adversario humano) y **MiniMaxPlayer, MinMaxRestrainedPlayer y AlfaBetaPlayer**, que serán las tres clases sobre las que trabajaremos (**no será necesario modificar el código del resto de clases**).

    - En la clase principal, podemos destacar las siguientes propiedades. FILAS, COLUMNAS y CONECTA indican, respectivamente, el número inicial de filas y columnas del tablero, así como el número de fichas que, colocadas consecutivamente, darán la victoria a uno de los jugadores. No será necesario modificarlas, salvo en los casos que se indican más adelante.

- Clase **Grid**: Un grid o tablero de juego se representa como un array de casillas, al que podemos acceder como un array de enteros (**int [ ][ ] tablero**). En este último las celdas vacías se representan con un 0 (propiedad Main.VACIO), con un 1 las celdas ocupadas con fichas del

jugador 1 (propiedad Main.PLAYER1), con un -1 las del jugador 2 (propiedad Main.PLAYER2). Algunos métodos interesantes son:

- **public int checkWin()**. Comprueba si un tablero se encuentra en un estado ganador. Devuelve 1 si ha ganado el jugador 1, -1 si gana el jugador 2, o 0, si aún no ha ganado ninguno.
- **public int[ ][ ] getGrid()**. Devuelve un array de enteros con el tablero en su estado actual, siguiendo la notación indicada más arriba.
- **public int[ ][ ] copyGrid()**. Devuelve un array de enteros con una **copia** del tablero en su estado actual, siguiendo la notación indicada más arriba.
- **public int set(int col, int jugador)**. Coloca una ficha del **jugador** correspondiente en la columna **col** del tablero, siempre y cuando exista sitio en ella. Devuelve la posición (fila) en la que ha quedado situada la ficha, o -1 si la columna estaba completa y no se ha podido colocar la ficha.
- **public boolean fullColumn(int col)**. Devuelve true si la columna está completa (no se pueden dejar más fichas en ella).
- **public int TopColumn(int col).** Puedes usar este método para saber el color de la última ficha colocada en la columna (la que estará sobre el resto de fichas). Devuelve 1 (Main.PLAYER1), -1 (Main.PLAYER2), o -2 en caso de error (columna vacía).

- Clase **Player**: Clase abstracta para representar un jugador en la partida. Define el método **public abstract int turno(Grid tablero, int conecta)**, el cual ha de implementarse en cada una de sus subclases. Este método debe realizar un movimiento sobre el tablero y a continuación, comprobar si el jugador puede ganar la partida, para lo que puede invocar al método **checkWin()** de la clase **Grid**. Devuelve un valor entero con la columna donde dejar caer la ficha.
  - **protected final int getRandomColumn(Grid tablero).** Devuelve una columna al azar donde dejar caer una ficha, siempre que sea posible hacerlo.

- Clase **GreedyPlayer**: Subclase de **Player**. Como tal, implementa el método **turno**, el cual nos puede servir de referencia para nuestro objetivo en esta práctica. Actualmente, el método comprueba el estado del tablero y, de manera un tanto rudimentaria, intenta colocar la ficha para impedir que el otro jugador gane, y al mismo tiempo intentar ganar él.

- Clases **MiniMaxPlayer, MinMaxRestrainedPlayer y AlfaBetaPlayer**: Estas son las clases donde vamos a trabajar exclusivamente, sobrecargando el método **turno** para que implemente el algoritmo de Inteligencia Artificial junto con sus modificaciones (ahora mismo, simplemente devuelve una columna al azar donde colocar la siguiente ficha).

  - **Se permite la implementación de tantos elementos auxiliares dentro de la clase (propiedades, métodos, clases internas, etc.) como se considere necesario.**

1.5. **Objetivo general de la práctica**

Con esta práctica se pretende que los estudiantes trabajen con los **algoritmos MINIMAX y poda alfa-beta**, vistos en clase de teoría, para búsqueda en juegos, desarrollando y probando una heurística para un juego en concreto, en nuestro caso, **Conecta-4**.

Los estudiantes tendrán que implementar dicho algoritmo e integrarlo dentro del juego, para que sea empleado por el jugador **CPU (MiniMaxPlayer, MinMaxRestrainedPlayer o AlfaBetaPlayer)** durante una partida.

Haciendo uso de las librerías que Java nos proporciona, podemos definir e implementar tantos atributos y métodos auxiliares como consideremos necesarios, pero **siempre dentro de nuestra clase**. Es decir, podemos revisar el código del resto de la aplicación para aprender cómo funciona ésta, **pero bajo ningún concepto debemos modificarlo**.

Por este motivo, el único código que los alumnos deben entregar, y el único que se tendrá en cuenta en la corrección de la práctica, será el relativo a nuestros jugadores, contenido en los archivos **MiniMaxPlayer.java, MinMaxRestrainedPlayer.java y AlfaBetaPlayer.java**.

El objetivo general se divide en tres objetivos más específicos que se abordarán en tres actividades:

- Actividad 1: Implementación del algoritmo MINIMAX para el juego CONECTA.
- Actividad 2: Implementación del algoritmo MINIMAX restringido con información heurística.
- Actividad 3: Algoritmo MINIMAX con poda ALFA-BETA.

2. Actividad 1. Implementación del algoritmo MINIMAX

La implementación de un algoritmo MINIMAX puede llevarse a cabo de formas diferentes, pero lo importante es tener clara la idea de su funcionamiento.

El algoritmo debe generar un árbol de soluciones completo a partir de un nodo dado, y este árbol se debería desplegar hasta los nodos finales con todas las posibles combinaciones a lo largo de su desarrollo. Como se puede imaginar, el coste es muy alto y necesitamos mecanismos que optimicen esta técnica (en los que se centrarán los dos puntos siguientes de la práctica).

NOTA: Para implementar el algoritmo MINIMAX completo será conveniente reducir el tamaño del tablero (FILAS, COLUMNAS) o el número de fichas a

conectar (CONECTA), dentro de la clase Main. Indicar qué valores se han usado en la memoria de la práctica.

El objetivo a evaluar será la construcción del árbol MINIMAX sobre cada una de las jugadas que se vayan realizando por parte del jugador, es decir, cuando el Jugador 1 pone una ficha se debe generar un árbol MINIMAX con todas las posibles jugadas hasta los nodos terminales. **El algoritmo debe elegir el movimiento que, tras propagar la función de utilidad por el árbol, la maximice**. Para facilitar la corrección, **se puede desarrollar un método de visualización**, por ejemplo, mediante consola/fichero que presente de una forma intuitiva la estructura del árbol MINIMAX con las posibles jugadas de la máquina.

3. Actividad 2. Incorporación de restricciones y heurísticas al algoritmo MINIMAX

En este apartado se propone limitar a N (un valor elegido por el estudiante) el número de niveles a explorar. Si antes de llegar a dicho nivel preestablecido se alcanza un nodo terminal, se le asignará un **valor de utilidad**. Si no, se le asignará un valor heurístico. En ambos casos, ese valor asignado será propagado hacia el nodo raíz, atendiendo al nivel MAX o MIN en el que nos encontremos.

En este caso puedes utilizar una función de evaluación estática (heurística) asociada al número de piezas consecutivas que tendrías con el movimiento indicado que viene determinado por el valor $-(n^2)$ y n^2, siendo n el número de piezas consecutivas en todas las direcciones.

NOTA: Para desarrollar las restricciones y la heurística asociada se debe mantener el tamaño original de tablero de 6x7.

En este caso, **el algoritmo debe elegir el movimiento que, tras propagar la heurística por el árbol, lo maximice.** Por tanto, en la consola debe mostrarse el árbol con las heurísticas calculadas y el movimiento asociado al proceso MINIMAX.

Para finalizar, destacar que para el algoritmo RESTRINGIDO MINIMAX, el proceso recursivo finaliza con distintas condiciones:

- Gana algún jugador.
- Se han explorado N niveles, siendo N el límite establecido. Este aspecto es fundamental pues la eficiencia, eficacia y calidad de nuestro algoritmo dependerá de los niveles utilizados. El valor de N debe ser elegido por los estudiantes, de forma justificada. Puedes probar el rendimiento del

algoritmo con diferentes valores de N.

- Se llega a una situación estática donde no hay grandes cambios. En este caso, el algoritmo debe devolver un movimiento de entre los posibles. Es importante justificar de forma adecuada qué valor devuelve en este caso y por qué.

4. Actividad 3. Algoritmo MINIMAX RESTRINGIDO con poda ALFA-BETA

Analizando el comportamiento del algoritmo MINIMAX RESTRINGIDO seguimos viendo la necesidad de mejorar la eficiencia y eficacia del algoritmo, que van relacionadas con la gran cantidad de jugadas posibles que se pueden desarrollar. En este sentido necesitamos seguir mejorando la construcción de nuestro algoritmo restringiendo el número de jugadas.

En este apartado se pide **incorporar la poda ALFA-BETA** de forma que nos permita restringir a un mayor nivel nuestro árbol MINIMAX.

Aquí podéis ver el funcionamiento de una poda alfa-beta:
http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html

Introducir los siguientes valores para ver el funcionamiento:

Enter the game tree structure: 3 3 3 3
Enter the game terminal values: 3 12 8 2 4 6 14 5 2

Para evaluar el comportamiento del algoritmo de poda alfa-beta, vamos a realizar diferentes partidas, con diferentes configuraciones, y recogeremos en una tabla como la que aparece a continuación el resultado de las mismas:

Ejecución	Nivel de restricción	Jugada inicial	Movimientos	Ganador
1	3			
2				
3				
1	4			
…				
1	…			
…				
1				

…	7			

- En la columna **Nivel de restricción** se indicará el nivel de profundidad a la que se realizará la poda de los árboles generados.
- En la columna **Jugada inicial** hay que indicar la columna en la que pone la primera ficha el jugador humano (PLAYER1). ***En cada una de las tres ejecuciones de cada nivel, la primera ficha que se pone en juego debe estar en casillas diferentes.***
- En la columna **Movimientos** se mostrará el número total de movimientos que se han realizado en la partida.
- Finalmente, en la última columna hemos de indicar qué jugador ha **ganado** cada partida: PLAYER1, ALFABETA o EMPATE.

La tabla debe contener 3 ejecuciones para cada nivel. En todas las partidas, el jugador 1 será el propio usuario, mientras que el jugador 2 será la CPU con el algoritmo de poda Alfa-beta implementado.

5. Aspectos que se valorarán en la nota final

- La explicación o descripción de la estrategia seguida en cada una de las actividades.
- La limpieza del código, la documentación interna del mismo, la documentación externa en el informe y el uso correcto de convenciones de Java.
- La correcta implementación del algoritmo, así como la justificación del buen funcionamiento del mismo.
- El análisis de los resultados obtenidos.

6. Entrega y evaluación

- La puntuación máxima de la práctica 2 será de **3,5 puntos**, distribuidos de la siguiente forma:
  - Entrega 1:
    - Actividad 1. Implementación del algoritmo MINIMAX: 1,5 puntos
  - Entrega 2:
    - Actividad 2. Incorporación de restricciones y heurísticas al algoritmo MINIMAX: 0,5 puntos
    - Actividad 3. Algoritmo MINIMAX RESTRINGIDO con poda ALFA-BETA: 1,5 puntos

- La **primera entrega** se llevará a cabo a través de la actividad correspondiente en PLATEA con fecha límite el **jueves 27/04/23**.
- La **segunda entrega** se llevará a cabo a través de la actividad correspondiente en PLATEA con fecha límite el **jueves 18/05/23**.
- La defensa de ambas partes de la práctica se realizará en la sesión de prácticas siguiente a la entrega. En ella, los dos alumnos de cada pareja

de prácticas deberán demostrar su conocimiento sobre el trabajo realizado.

- La práctica se realizará por parejas y cada pareja entregará a través de la plataforma **un único fichero .zip** que contenga:
  - Los archivos fuente **MiniMaxPlayer.java, MinMaxRestrainedPlayer.java o AlfaBetaPlayer.java** que se corresponden con la entrega, debidamente comentados.
  - Documentación adicional en formato PDF explicando cómo se han implementado los diferentes apartados, ejemplos ilustrativos y las pruebas que se hayan realizado de los diferentes algoritmos.
- No se tendrá en cuenta la modificación de ninguna otra clase del entorno que no sean las clases **MiniMaxPlayer.java, MinMaxRestrainedPlayer.java o AlfaBetaPlayer.javar**. 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 aceptará exclusivamente el archivo .java** referido en el punto anterior.
- **El incumplimiento de las normas anteriores repercutirá negativamente en la calificación de la práctica.**