Grado en Ingeniería Informática

Inteligencia Artificial

Curso 2019/2020

Universidad de Jaén

Guión 4 (A)
Juegos

1. Juego Min-Max entre adversarios

1.1. Introducción

Conecta-4 (**4 en Raya**, etc.) es un juego para dos jugadores donde éstos introducen, alternándose 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, aunque existen otras variaciones con diferentes tamaños.

Cada jugador dispone de un conjunto de fichas de un color determinado y, alternativamente, deben 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 primero 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.

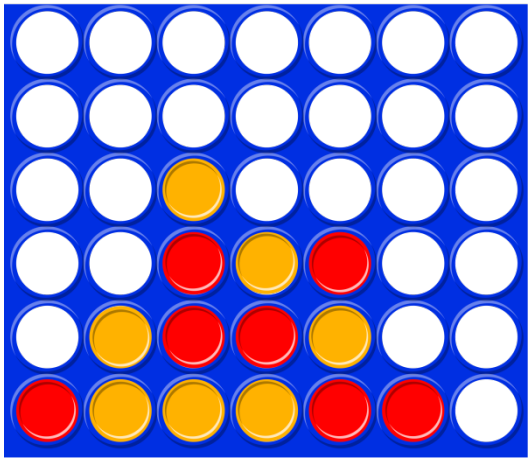


Figura 1. Conecta 4, el juego original (Milton Bradley, 1974)

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 (Además, el juego se resolvió completamente en 1988).

1.2. Características

- Requiere Java versión 7 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.
- A pesar de que el tamaño del tablero es variable, vamos a configurarlo por defecto al inicio del juego con el tamaño de 6x7. El número de fichas ganadoras permanecerá fijo a 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 usuario y es quién 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. El programa no podrá escribir datos en ningún fichero ni debe mostrar ninguna otra información por pantalla.

1.3. Instalación

La instalación del proyecto es análoga a como hicimos en prácticas anteriores. Desde **NetBeans**, creamos un nuevo proyecto (**Java with Ant en la V11**), bajo la opción *"Java Application"*.

Una vez hecho esto, descargamos desde ILIAS el fichero *conecta.zip* con el juego, y lo descomprimimos. Buscaremos la carpeta del nuevo proyecto y allí copiamos/reemplazamos tal cual el contenido del fichero descomprimido.

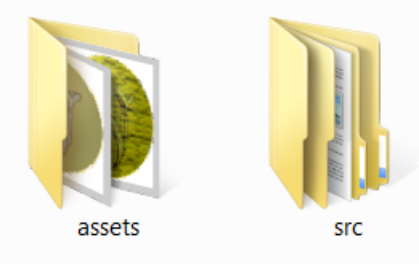


Figura 2. Contenido del archivo conectan.zip

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 *conecta4.Conecta4*).

Una vez configurado, podremos ejecutar el proyecto y el juego dará comienzo.

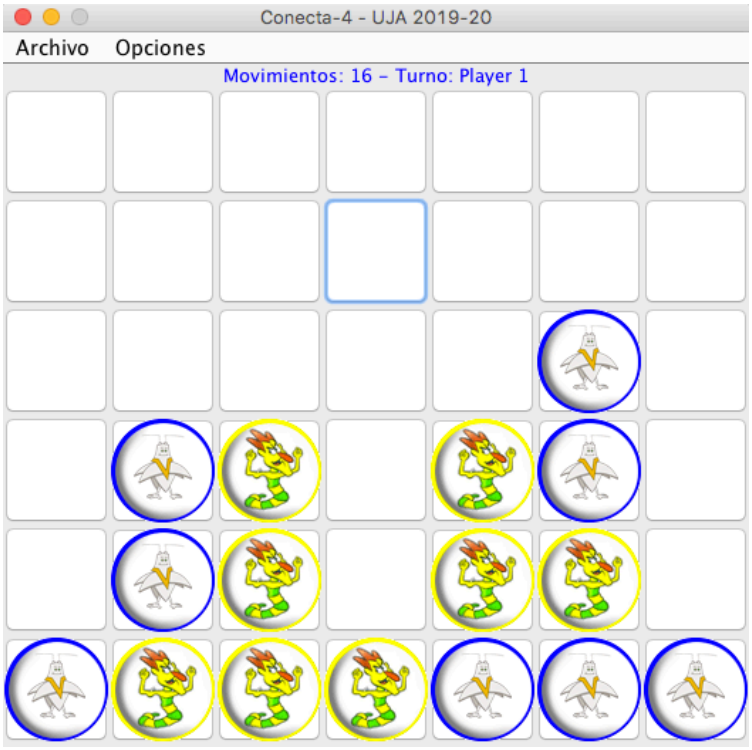


Figura 3. Ejemplo de partida en tablero 6x7

1.4. Funcionamiento

El Jugador 1 (Humano) siempre empieza la partida y juega con las fichas amarillas del COVID. El Jugador 2 juega con las fichas azules de los ANTICUERPOS.

En el menú **Archivo** tenemos solamente 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 Aleatorio* (opción por defecto al inicio de la aplicación), o *CPU IAPlayer*. Al seleccionar cualquiera de las anteriores opciones, se interrumpirá la partida en curso y se iniciará una nueva 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.

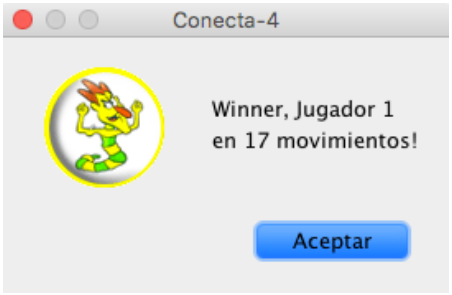


Figura 4. Ventana modal de fin de partida

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

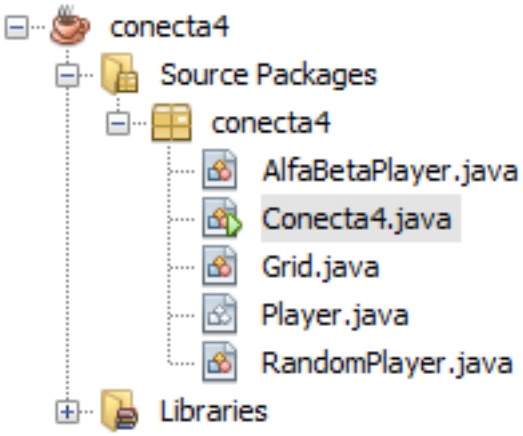


Figura 5. Árbol de clases del proyecto conectan

- La clase principal de nuestro juego es **Conecta4**, 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 **RandomPlayer** (la cual implementa un algoritmo rudimentario para que la máquina pueda jugar contra un adversario humano) y **IAPlayer**, que será la clase sobre la cual trabajaremos (**no será necesario modificar el código del resto de clases**).
- Clase **Grid**: Un grid o tablero de juego se representa como un array de casillas, al que podemos acceder tanto en su versión gráfica para la representación en la ventana (**JButton [ ][ ] boton**), como en una versión más cómoda, como array de enteros (**int [ ][ ] boton_int**). En este último las celdas vacías se representan con un 0, con un 1 las celdas ocupadas con fichas del jugador 1, y con un -1 las del jugador 2. Algunos métodos interesantes son:
  - **public int checkWin (int x, int y, int conecta)**. Comprueba si, tras el último turnoJugada (colocar una ficha en la casilla <x,y>), alguno de los jugadores puede haber ganado la partida (es decir, si ha situado seguidas un número de fichas igual al valor de

conecta). Devuelve 1 si ha ganado el jugador 1, -1 si gana el jugador 2, o 0, si aún no ha ganado ninguno.
    - **public int[ ][ ] toArray()**. Devuelve un array de enteros con el estado actual del tablero, siguiendo la notación indicada más arriba.
    - **public int setButton(int col, int jugador)**. Coloca una ficha del **jugador** correspondiente en la columna **col** del tablero, siempre y cuando aún quedara 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.
- Clase **Player**: Clase abstracta para representar un jugador en la partida. Define el método **public abstract int turnoJugada(Grid tablero, int conecta)**, el cual ha de implementarse en cada una de sus subclases. Este método debe llevar a cabo 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 1 si gana el jugador 1, -1 si gana el jugador 2, o 0 en cualquier otro caso.
- Clase **RandomPlayer**: Subclase de **Player**. Como tal, implementa el método **turnoJugada**, 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.
- Clase **IAPlayer**: Ésta es la clase donde vamos a trabajar exclusivamente, sobrecargando el método **turnoJugada** para que implemente el algoritmo de Inteligencia Artificial junto con sus modificaciones (ahora mismo, simplemente coloca una ficha en una columna al azar). Se permite la implementación de tantos métodos auxiliares dentro de la clase como se considere necesario.

1.5. Objetivo general

Con la presente práctica se pretende que los alumnos comprendan el funcionamiento de los **algoritmos minimax y poda alfa-beta**, visto en clase, para búsqueda en juegos, desarrollando y probando una función de evaluación para un juego en concreto, en nuestro caso, **Conecta-4**.

Los alumnos tendrán que implementar dicho algoritmo e integrarlo dentro del juego, para que sea empleado por el jugador **IAPlayer** 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 nuestro jugador, contenido en el archivo **IAPlayer.java**.

Este objetivo general se dividirá en tres objetivos más específicos que se corresponderán con los distintos guiones de los que está compuesta esta práctica con duración de tres semanas:

- Objetivo 1.5.A: Implementación del algoritmo MINIMAX para el juego CONECTA-4.

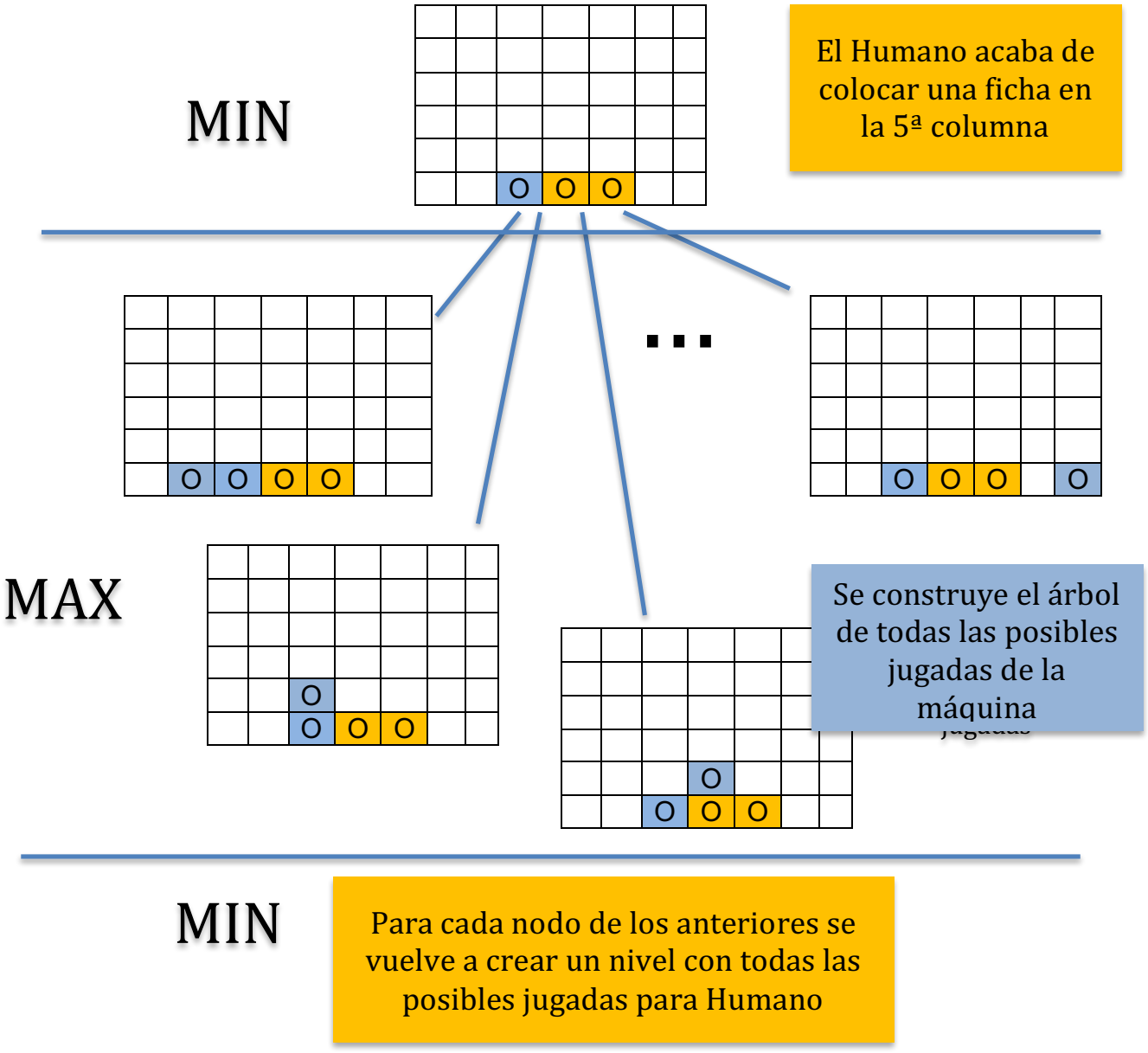
- Objetivo 1.5.B: Implementación del algoritmo MINIMAX **RESTRINGIDO** con información heurística.

- Objetivo 1.5.C: Algoritmo MINIMAX con poda ALFA-BETA.

1.5.A. Objetivos específicos para este guión: **Implementación del algoritmo MINIMAX**

La implementación de un algoritmo MINIMAX puede variar, pero lo importante es tener clara la idea de su funcionamiento.

El algoritmo primero debe generar un árbol de soluciones completo a partir de un nodo dado. Veamos un ejemplo:



Este árbol se debería de desplegar hasta los nodos finales con todas las posibles combinaciones a lo largo de su desarrollo. Como puedes imaginar el coste es muy alto y necesitamos de mecanismos que optimicen esta técnica.

NOTA: Para facilitar el desarrollo se aconseja modificar el tamaño del tablero a 4x4.

El objetivo a evaluar esta semana 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.

Para facilitar la corrección se debe desarrollar un método de visualización, por ejemplo, mediante consola que presente de una forma intuitiva la estructura del árbol MINIMAX con las posibles jugadas de la máquina.

2. Aspectos que se valorarán en la nota final

- La explicación o descripción de la estrategia seguida.
- 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.

3. Entrega y evaluación

- **El objetivo 1.5.A de la práctica se evalúa con un máximo de 1 punto para el algoritmo MINIMAX.**
- La práctica se realizará por parejas, y el árbol MINIMAX se **defenderá** en la clase del **día 17 de abril**. Se analizará el archivo fuente **IAPlayer.java**, debidamente comentado y clases auxiliares si se han definido.
- No se tendrá en cuenta la modificación de ninguna otra clase del entorno que no sea la clase **IAPlayer**. 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 no cumplimiento de las anteriores normas repercutirá negativamente en la calificación de la práctica.**