INTELIGENCIA ARTIFICIAL
Grado en Ingeniería Informática

Práctica 3

Métodos de Búsqueda con Adversario (Juegos)

CONECTA-4

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

UNIVERSIDAD DE GRANADA

Curso 2018-2019

1. Introducción

1.1. Motivación

La tercera práctica de la asignatura ***Inteligencia Artificial*** consiste en el diseño e implementación de técnicas de búsqueda con adversario en un entorno de juegos. Al igual que en la práctica anterior, se trabajará con una versión modificada del simulador, en ese caso adaptada para el juego CONECTA-4. Este simulador se ha realizado de cero para esta práctica, orientado para facilitar el proceso de exploración.

El entorno de simulación simular el juego CONECTA-4 (también conocido como 4 en Raya en algunas versiones). CONECTA-4 es un juego de mesa para dos jugadores distribuido por Hasbro, en el que se introducen fichas en un tablero vertical con el objetivo de alinear cuatro consecutivas de un mismo color. Fue creado en 1974 por Ned Strongin y Howard Wexler para Milton Bradley Company.

Para la realización de esta práctica, el alumno deberá conocer en primer lugar las técnicas de búsqueda con adversario explicadas en teoría (Tema 4). En concreto, **el objetivo de esta práctica es la implementación de MINIMAX (con cota máxima de 4) o MINIMAX con PODA ALFA-BETA, con profundidad limitada (con cota máxima de 8)**, para dotar de comportamiento inteligente deliberativo a un jugador artificial para este juego, de manera que esté en condiciones de competir y ganar a su adversario.

A continuación, explicamos cuáles son los requisitos de la práctica, los objetivos concretos que se persiguen, el software necesario junto con su instalación, y una guía para poder programar el simulador.

2. Requisitos

Para poder realizar la práctica, es necesario que el alumno disponga de:

- Conocimientos básicos del lenguaje C/C++: tipos de datos, sentencias condicionales, sentencias repetitivas, funciones y procedimientos, clases, métodos de clases, constructores de clase.
- El entorno de programación **CodeBlocks en el caso de trabajar bajo Sistema Operativo Windows**, que tendrá que estar instalado en el computador donde vaya a realizar la práctica. Este software se puede descargar desde la siguiente URL: **http://www.codeblocks.org/** o desde la web de la asignatura facilitada por el profesor.
- La librería SFML, descargable en https://www.sfml-dev.org/, esta librería ya se ha usado en otras asignaturas como en **Metodología de la Programación**. En todo caso, en la primera sesión de prácticas se explicará cómo instalarlo (en particular en Windows, en Linux se puede instalar fácilmente usando el repositorio de paquetes de su propia distribución).
- El código de Connect4, disponible en PRADO.

3. Objetivo de la práctica

La práctica tiene como objetivo diseñar e implementar un agente deliberativo que pueda llevar a cabo un comportamiento inteligente dentro del juego CONNECT4 que se explica a continuación.

El objetivo de CONNECT4 es alinear cuatro fichas sobre un tablero formado por ocho filas y ocho columnas (en el juego original, el tablero es de seis filas). En el juego original cada jugador dispone de 25 fichas de un color diferente. Por turnos, los jugadores deben introducir una ficha en la columna que prefieran (de la 1 a la 8, numeradas de izquierda a derecha, siempre que no esté completa) y ésta caerá a la posición más baja. **Gana** la partida **el primero que consiga alinear cuatro fichas consecutivas** de un mismo color en horizontal, vertical o diagonal. Si todas las columnas están ocupadas se produce un empate.

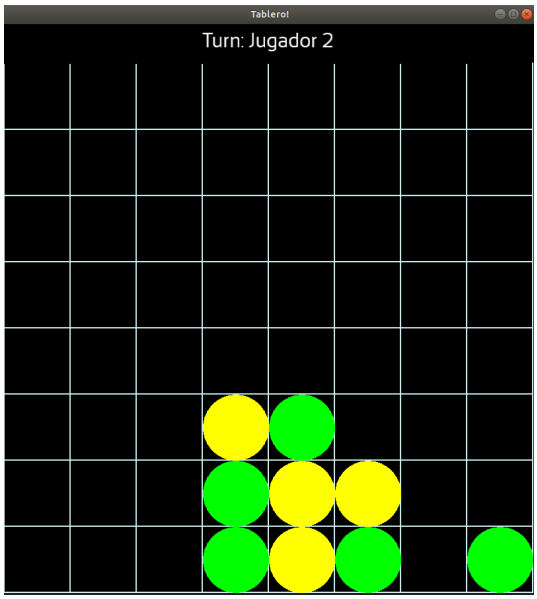

Figura 1. Imagen de una partida de CONECTA.

OBJETIVO DE LA PRÁCTICA:

A partir de estas consideraciones iniciales, el objetivo de la práctica es implementar MINIMAX **(con cota máxima de 4)** o MINIMAX con PODA ALFA-BETA, **con profundidad limitada (con cota máxima de 8)**, de manera que un jugador pueda determinar el movimiento más prometedor para ganar el juego, explorando el árbol de juego **desde** el estado actual **hasta** una profundidad máxima de 8 dada como entrada al algoritmo.

También forma parte del objetivo de esta práctica, la definición de una heurística apropiada, que asociada al algoritmo implementado proporcione un buen jugador artificial para juego del CONECTA-4.

Los conceptos necesarios para poder llevar a cabo la implementación del algoritmo dentro del código fuente del simulador se explican en las siguientes secciones.

4. Simulador Connect4

4.1 Instalación

El simulador **connect4** nos permitirá

- Implementar el comportamiento de uno o dos jugadores en un entorno en el que el jugador (bien humano o bien máquina) podrá competir con otro jugador software o con otro humano.

- Visualizar los movimientos en una interfaz de usuario basado en ventanas.

Para instalarlo, seguir estos pasos:

1. Descargue el fichero **connect4.zip** desde la web de la asignatura en **PRADO**, y cópielo su carpeta personal dedicada a las prácticas de la asignatura de ***Inteligencia Artificial***. Supongamos, para los siguientes pasos, que esta carpeta se denomina "**U:\IA\practica3**".
2. Desempaquete el fichero en la raíz de esta carpeta.
3. La compilación se puede hacer con "cmake .; make" o con el IDE C++ de su elección.

4.2 Ejecución

El simulador tiene varios modos:

- Sin argumentos, permite jugar dos jugadores humanos por medio del ratón.

- Con argumento: Ejemplo "connect4 random". De esta forma se permite jugar al humano con las decisiones tomadas por el programa pasado por parámetro (random en el ejemplo).

- Con dos argumentos: Ejemplos "connect4 random random" ó "connect random minmax", de esta manera se permite a dos programas inteligentes (random vs random ó random vs minmax) jugar entre sí.

4.3 Estructura y funcionamiento

El simulador posee una estructura relativamente sencilla que detallamos para que esté claro. No es necesario conocerlo en detalle porque el programa inteligente es un programa totalmente independiente, aunque es conveniente utilizar código del simulador (como la clase State) para simplificar el problema.

El programa posee las siguientes clases:

- Clase **Board**, responsable de pintar el tablero. Entre sus responsabilidades está:

  - Visualizar el tablero.

  - Implementar la acción de colocar una casilla en una posición.

  - Implementar el evento del ratón para el movimiento.

- Clase **State**, que almacena la información del estado sin dependencias con el entorno de representación.

  Esta clase almacena:

  - El estado actual del tablero, como una matriz cuadrada 8x8.
  - De quién es el turno.
  - Si ha ganado alguien, quién lo ha hecho, por medio del método **has_winner() y get_winner()**.

  Adicionalmente, permite conocer la siguiente información:

  - Método **check_winner** Detectar si algún jugador ha ganado actualizando la información del ganador.

- Método **move** que permite al usuario actual (para ese estado) añadir una nueva ficha. Dicho método no modifica el estado, si no que devuelve un nuevo estado, el resultado de la nueva ficha.
- Métodos para cargarse en un fichero (el constructor que recibe el nombre del fichero), y leerse de fichero (mediante el método **save)**.

- Clase **Game**, responsable del juego, responsable de la lógica del juego.

4.4 Pasos de llamada al programa inteligente

Para simplificar el proceso, el programa inteligente es totalmente independiente del proceso juego. Para ello, la comunicación entre **Game** y el programa se hace únicamente por medio de ficheros intermedios (ya que el obtener la salida de un proceso en C++ no es estándar, depende del SO. Existe un estándar para ello, pero sólo lo implementan los SO que siguen el formato POSIX como Linux o MacOS).

A la hora de llamar Game a nuestro programa, se realizan los siguientes pasos:

- Se guarda en “state.txt” el estado del juego (mediante el método *save* de **State**).

- Se llama al programa sin ningún argumento.

- El programa inteligente escribe en el fichero de texto “output.txt” una única línea con un valor numérico entre 1 y 8.

- Game abre el fichero “output.txt” y realiza el movimiento pedido.

Por tanto, no es necesario que el programa utilice ningún código del proyecto. Sin embargo, se recomienda mirar el código y reutilizar la clase **State** para simplificar el proceso. Dicha clase **State no se podrá modificar**, si se detectase algún error debe de comunicarse al profesorado correspondiente.

Como ejemplo se ha incluido en el código un programa (agent_random.cpp) que elige el movimiento de forma totalmente aleatoria.

Software a entregar

Se pide desarrollar dos programas: Uno que implemente el algoritmo MINIMAX y otro con el MINIMAX con PODA ALFA-BETA en los términos en que se ha explicado previamente. Se deberá de entregar:

1. Código fuente de los programas, preparado para compilar.
2. Fichero README, indicando cómo se puede compilar.
3. una memoria de prácticas en formato PDF (no más de 5 páginas) que, como mínimo, contenga los siguientes apartados:
   a. Análisis del problema
   b. Descripción de la solución planteada

Estos ficheros deberán entregarse mediante PRADO dentro de la fecha de entrega, en un fichero ZIP que descomprima en una carpeta con el nombre del autor. **No se evaluarán aquellas prácticas que no compilen, o compilen con errores, ni aquellos que cumplan sin memoria**

5. Evaluación y entrega de prácticas

La **calificación final** de la práctica se calculará de la siguiente forma:

- Se entregará una memoria de prácticas al finalizar las tareas a realizar. La fecha límite de la entrega de la memoria será comunicada con suficiente antelación por el profesor de prácticas en clase, y publicada en la página web de la asignatura.

- La práctica se califica numéricamente de **0 a 10**. Se evaluará como la suma de los siguientes criterios:
  - La memoria de prácticas se evalúa de **0 a 2.**
  - El comportamiento e implementación del algoritmo se evaluará de **0 a 8** puntos:
    - Hasta 3 puntos por el algoritmo MINIMAX.
    - Hasta 3 puntos por el algoritmo con PODA.
    - Se valorará con 2 puntos el criterio heurístico.
  - La nota final será la suma de los dos apartados anteriores. Es necesario entregar la memoria de prácticas para que sea evaluada la práctica.

- La **fecha de entrega de la práctica**

Domingo 2 de Junio antes de las 23:00 horas.

5.4. Observaciones Finales

Esta **práctica es INDIVIDUAL**. El profesorado, para asegurar la originalidad de cada una de las entregas, someterá a estas a un procedimiento de detección de copias. En el caso de detectar prácticas copiadas, los involucrados (tanto el que se copió como el que se ha dejado copiar) tendrán suspensa la asignatura. Por esta razón, recomendamos que en ningún caso se intercambie código entre los alumnos. No servirá como justificación del parecido entre dos prácticas el argumento "es que la hemos hecho juntos y por eso son tan parecidas", o "como estudiamos juntos, pues...", ya que como se ha dicho antes, **las prácticas son INDIVIDUALES**.

Como se ha comentado previamente, el objetivo de la defensa de prácticas es evaluar la capacidad del alumno para enfrentarse a este problema. Por consiguiente, se asume que todo el código que aparece en su práctica es original y ha sido introducido por alguna razón, de forma que el alumno domina perfectamente el código que entrega. Así, si se producen coincidencias sospechosas se convocará a los alumnos implicados para una defensi. Si durante cualquier momento del proceso de defensa el alumno no justifica adecuadamente algo de lo que aparece en su código, la práctica se considerará copiada y tendrá suspensa la asignatura. Por esta razón, aconsejamos no incluir nada en el código que el alumno no sea capaz de explicar qué misión cumple dentro de su práctica y que revise el código con anterioridad a la defensa de prácticas.

Por último, las prácticas presentadas en tiempo y forma, pero no defendidas por el alumno cuando se le convoque, se considerarán como no entregadas y el alumno obtendrá la calificación de 0. El supuesto anterior se aplica a aquellas prácticas no involucradas en un proceso de copia. En este último caso, el alumno tendrá suspensa la asignatura.