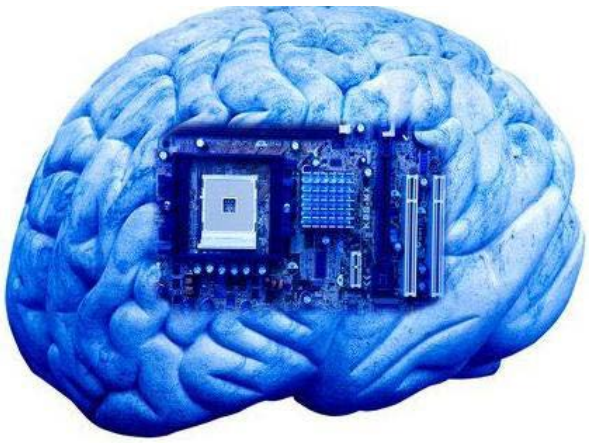
Práctica 3: Juegos con adversario



Piedad Garrido Picazo (piedad@unizar.es)

Grado en Ingeniería Informática. Inteligencia Artificial 2020/2021

Contenido

Esta práctica se corresponde con el Tema 2-Parte IV de la asignatura, que a su vez está basado en el Capítulo 5 de la bibliografía básica ("*Artificial Intelligence: A Modern Approach, Chapter 5: Adversarial Search*").

Objetivos de la práctica

La tercera práctica de la asignatura se centrará en la resolución de problemas en los que intentamos planear de forma adelantada en un entorno en el que otros agentes están planeando contra nosotros, es decir, en entornos adversarios. Este tipo de entornos son muy usuales en programas de ordenador representando diferentes juegos de más de un jugador en el que existen condiciones de victoria y derrota para alguno de los mismos, como el ajedrez, damas, tres en raya, etc. El alumno deberá ser capaz de entender las bases de los algoritmos que pretenden adelantarse a los movimientos del adversario en el futuro para elegir la acción más beneficiosa para sí mismo.

Búsqueda adversaria

Vamos a centrarnos en problemas en los que tenemos sólo dos jugadores (por simplicidad, aunque posteriormente podría ampliarse el sistema para un número mayor de jugadores). Uno de ellos los llamaremos MAX y al otro lo llamaremos MIN, ya que vamos a intentar maximizar el beneficio obtenido por el jugador MAX a la vez que minimizamos el beneficio obtenido por MIN. Para definir uno de estos problemas, debemos especificar los siguientes parámetros:

- El estado inicial del sistema, es decir, cómo empieza la partida.
- El jugador que debe mover en cada estado.
- Los movimientos legales de cada estado.
- Un modelo que determine el resultado de un estado.
- Una función que determine si un estado es final.
- Una función de utilidad que indica qué jugador ha ganado en un estado final.

El esquema general de un algoritmo de búsqueda adversaria MINIMAX aparece en la Figura 1, compuesto por tres funciones diferentes, aunque todas necesarias:

```
function MINIMAX-DECISION(state) returns an action
  return arg max a ∈ ACTIONS(state) MIN-VALUE(RESULT (state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE (RESULT(s, a)))
  return v

function MIN-VALUE (state) returns a utility value
  if TERMINAL-TEST (state) then return UTILITY (state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE (RESULT(s, a)))
  return v
```

Figura 1: Algoritmo para calcular decisiones minimax

Este algoritmo presenta un problema en juegos largos, ya que explora el árbol hasta los estados finales y puede llegar a ser muy costoso en cuanto a cálculo. Por ello, se modificará esta versión para que la evaluación termine cuando se sobrepase un nivel dado del árbol.

Existe una versión más eficiente de este algoritmo que hace uso de una función de evaluación para comprobar los estados intermedios, además de "podar" algunas ramas del árbol de las que se conoce que no se va a mejorar el resultado actual. Este algoritmo es la versión de minimax con poda alfa-beta (Figura 2):

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE (state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return EVAL(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE (RESULT(s, a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE (state, α, β) returns a utility value
  if TERMINAL-TEST (state) then return EVAL (state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE (RESULT(s, a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figura 2: Algoritmo de poda alfa-beta

Problema a resolver

Utilizaremos el juego del Conecta 4, o 4 en raya, para practicar la búsqueda con adversario. En este juego, se introducen fichas de colores en un tablero, de forma que siempre caen a la casilla más baja disponible. Cada jugador intenta formar una línea con 4 fichas de su color, a la vez que evita que las forme el otro jugador. La siguiente figura muestra un estado de victoria para el jugador azul en el juego:

Figura 3: Ejemplo de estado en el juego de Conecta 4

El tablero se encuentra inicialmente vacío, y el juego termina cuando alguno de los jugadores forma una línea o cuando el tablero se llena del todo, con lo que se llega al empate. Las diferentes acciones se representarán simplemente con un número, que indica la columna en la que se debe introducir la ficha por el jugador actual.

Ficheros de la práctica

La práctica está compuesta por los siguientes ficheros:

- `connect4game.py`: contiene una clase que representa un estado del juego del Conecta 4. El alumno debe rellenar el código de las funciones:
  - `result_state`
  - `successors`
  - `eval_board`
- `minimax.py`: contiene las funciones correspondientes al algoritmo minimax. El alumno debe rellenar el código de las siguientes funciones:
  - `max_value`
  - `min_value`
- `alphabeta.py`: contiene las funciones correspondientes al algoritmo minimax con poda alfa-beta. El alumno debe rellenar el código de las siguientes funciones:
  - `max_value`
  - `min_value`
- `players.py`: contiene las clases con los diferentes agentes jugadores que pueden emplearse con el juego.
- `connect4world.py`: contiene la interfaz gráfica del problema, y sirve como programa principal de la práctica, es decir, es el fichero que se debe ejecutar.

Entrega de la práctica

Deberán completare las siguientes tareas para considerar la práctica entregada:

1. Completar el código Python necesario para que los algoritmos minimax y alfabeta funcionen correctamente, siguiendo los algoritmos proporcionados. Además, deberá completarse la función *eval_board* que evalúa un estado del juego, no necesariamente final.
2. Estudiar diferentes partidas en las que intervenga el jugador aleatorio (*RandomPlayer*) con jugadores que hacen uso de los algoritmos minimax y alfabeta (*MinimaxPlayer* y *AlphabetaPlayer*). Probar diferentes niveles de profundidad en el árbol de búsqueda.

La entrega de esta práctica se realizará en una única modalidad a través del ADD. El alumno deberá realizar una memoria indicando cómo se ha realizado la implementación y los resultados obtenidos por los diferentes algoritmos. Se deberá entregar un fichero comprimido (ZIP) con el código de la práctica y la memoria realizada en formato PDF. La fecha límite de entrega **se llevará a cabo siguiendo las fechas indicadas en el documento de planificación de las prácticas del curso académico**.