LABORATORIO - ANÁLISIS DE JUEGOS

Sistemas Inteligentes
Grado en Ingeniería de Computadores

L. Mandow
Dpto. Lenguajes y Ciencias de la Computación
Universidad de Málaga

CONTENIDO

1. El juego del Tic-Tac-Toe (3- en-raya)
2. Búsqueda en grafos de juegos: algoritmo MiniMax
3. Número de nodos hoja examinados por MiniMax
4. Práctica: número de nodos hoja examinados por Alfa-Beta

1. El juego del 3-en-raya

Los juegos en aima heredan de la **clase Game**, que define, entre otros, los métodos para la búsqueda MiniMax y Alfa-Beta.

Cada instancia de la clase Game guarda básicamente el estado del juego (clase StateGame). El estado guarda el contenido del tablero, el turno del jugador, etc.

La implementación que estudiaremos no utiliza funciones de evaluación, sino que realiza la búsqueda hasta las posiciones finales del juego (por tanto, sólo aplicable a juegos sencillos).

La **clase TicTacToe** hereda de la clase Game e implementa los métodos necesarios para el juego del 3-en-raya.

Para **crear un nuevo juego** del 3-en-raya

```
TicTacToe t3 = new TicTacToe();
```

Inicialmente es el turno de 'X' y el tablero está vacío.

Para **realizar un movimiento** sobre el mismo podemos elegir entre:

```
t3.makeMove(1, 1);
t3.makeMiniMaxMove();
t3.makeAlphaBetaMove();
```

Podemos **mostrar el contenido** del tablero usando:

```
t3.getBoard(t3.getState()).print();
```

El fichero

`aima.gui.demo.search.TicTacToeDemo.java`

posee una demo en la que dos jugadores se enfrentan con la misma estrategia (primero MiniMax y luego AlfaBeta). La partida jugada es siempre la misma, ya que a igualdad de evaluaciones, el programa se queda siempre con la primera jugada.

El fichero

`aima.gui.applications.search.games.TicTacToeApp.java`

posee una aplicación interactiva que nos permite jugar contra la máquina (es imposible ganarle).

2. Búsqueda en grafos de juegos: algoritmo MiniMax

Consideremos el método **makeMiniMaxMove()** de la clase Game.

Su tarea consiste en:

1. calcular el mejor movimiento posible con el método getMiniMaxValue()
2. ejecutarlo, actualizando el estado actual del juego

El método **getMiniMaxValue()** es particular de cada juego. Según el turno, calcular el valor:

a) Turno de MAX (X), se llama a **maxValue(GameState state)** de la clase Game
b) Turno de MIN (0), se llama a **minValue(GameState state)** de la clase Game

Estos dos métodos se llaman recursivamente entre sí, implementando el procedimiento MiniMax.

Para el caso de los nodos MAX:

```
public int maxValue(GameState state) {
  int v = Integer.MIN_VALUE;
  if (terminalTest(state)) {
        return computeUtility(state);   //  1, 0, -1 según quien gane
  } else {
            List<GameState> successorList = getSuccessorStates(state);
            for (int i = 0; i < successorList.size(); i++) {
                  GameState successor = successorList.get(i);
                  int minimumValueOfSuccessor = minValue(successor);
                  if (minimumValueOfSuccessor > v) {
                        v = minimumValueOfSuccessor;
                        state.put("next", successor);
                  }
             }
            return v;
    }
}
```

Análogamente, para los nodos MIN:

```
public int minValue(GameState state) {
    int v = Integer.MAX_VALUE;
    if (terminalTest(state)) {
        return computeUtility(state);    //  1, 0, -1 según quien gane
    } else {
            List<GameState> successorList = getSuccessorStates(state);
            for (int i = 0; i < successorList.size(); i++) {
                    GameState successor = successorList.get(i);
                    int maximumValueOfSuccessors = maxValue(successor);
                    if (maximumValueOfSuccessors < v) {
                         v = maximumValueOfSuccessors;
                         state.put("next", successor);
                    }
             }
             return v;
     }
}
```

3. Número de nodos hoja examinados por MiniMax

- Consideremos ahora los ficheros del paquete **sesionJuegos**
- Basándonos en los métodos estudiados, implementaremos un método en la **clase TicTacToe**:

```
public int getMiniMaxLeafs() {

   GameState state = presentState;
   int valor = 0;
   if (getPlayerToMove(state).equalsIgnoreCase("X")) {
            valor = maxValueLeafs(state);
   } else {
            valor = minValueLeafs(state);
  }
  return valor;
}
```

Los métodos `maxValueLeafs` y `minValueLeafs` perteneceran a la **clase Game**.

```
public int maxValueLeafs(GameState state) {
    int cont = 0;
    if (terminalTest(state)) {
          return 1;
    } else {
          List<GameState> successorList = getSuccessorStates(state);
          for (int i = 0; i < successorList.size(); i++) {
                cont = cont + minValueLeafs(successorList.get(i));
          }
         return cont;
}}


public int minValueLeafs(GameState state) {
    int cont = 0;
    if (terminalTest(state)) {
         return 1;
    } else {
         List<GameState> successorList = getSuccessorStates(state);
         for (int i = 0; i < successorList.size(); i++) {
               cont = cont + maxValueLeafs(successorList.get(i));
    }
    return cont;
}}
```

Podemos probar su funcionamiento con

```
sesionJuegos.TicTacToeDemo.java
```

4. Práctica: número de nodos hoja examinados por Alfa-Beta

Se pide elaborar un conjunto de métodos análogos para calcular el número de nodos hoja examinados por el procedimiento alfa-beta.