Práctica final: conecta4
Estructura de Datos

Antonio Coín Castro Miguel Lentisco Ballesteros
José María Martín Luque

27 de enero de 2017

Índice

1. Consideraciones de diseño

1.1. Modificaciones realizadas al código proporcionado

Hemos modificado la clase `tablero` añadiendo la siguiente funcionalidad:

- Una variable en tablero llamada `ult_col` que almacena la última columna en la que se insertó una ficha.
- Una función llamada `int GetElemento(int i, int j)` que devuelve qué es lo que hay en la posición indicada del tablero.

- Una sobrecarga de la función `asignar_subarbol`, que recibe un nodo hijo de la raíz y lo convierte en la nueva raíz, eliminando el resto del árbol.

1.2. Funciones auxiliares

Hemos diseñado una función, `cantidadAlineadas`, que cuenta cuántas alineaciones de `nRaya` fichas hay en el tablero que se le proporcione, para el jugador que insertó ficha en último lugar. Como nota, decir que esta función ha sido añadida de última hora y somos conscientes de que podría facilitar la programación de otros métodos. Sin embargo, estos métodos estaban ya implementados cuando decidimos incorporar esta función y no disponemos del tiempo necesario para reescribir el código.

1.3. Código descartado

En una versión preeliminar del proyecto, escribimos una función `generarHijos`, que se encargaba de crear los hijos de un tablero hasta la profundidad deseada. Se trataba de una función recursiva en vez de iterativa, pues la idea inicial era utilizar un procedimiento recursivo para general el árbol de soluciones. La indicamos aquí:

```
JugadorAuto::generarHijos(ArbolGeneral<Tablero>::Nodo& n, int profundidad)
{
  if (profundidad)
  {
    Tablero original = partida.etiqueta(n);
    int num_cols = original.GetColumnas();

    for (int col = 0; col < num_cols; ++col)
    {
      if ((original.hayHueco(col) > -1) && !original.quienGana())
      {
        // Crear nuevo tablero y hacer movimiento
        Tablero nuevo(original);
        nuevo.colocarFicha(col);
        nuevo.cambiarTurno();
        ArbolGeneral<Tablero> hijo(nuevo);

        // Crear hijos del nuevo nodo
        ArbolGeneral<Tablero>::Nodo raiz = hijo.raiz();
```

```
        generarHijos(raiz, profundidad - 1);
        // Enganchar el nodo al árbol
        partida.insertar_hijomasizquierda(n, hijo);
      }
    }
  }
}
```

Finalmente se desechó este código, pues a la hora de ampliar el árbol de soluciones teníamos algunos problemas. En el código final, hemos optado por una versión iterativa, que funciona sin problemas.

1.4. Profundidad del árbol generado

Hemos considerado que la profundidad más adecuada del árbol de soluciones es `5`, puesto que con profundidades mayores el programa tarda más tiempo en generar el árbol, ralentizando el inicio y el desarrollo del juego.

Como apunte, tenemos la sensación de que aun con un valor de `N` relativamente bajo como es 5, el turno del jugador automático no se desarrolla de forma inmediata. Desconocemos si esto se debe a la forma en que está programado nuestro jugador automático, o es un problema inevitable.

2. Métricas utilizadas

Hemos optado por utilizar 4 métricas diferentes, cada una de las cuales con una serie de mejoras respecto a la anterior, pero manteniendo la funcionalidad existente. Este modelo de mejora incremental nos garantiza que la `métrica 1` es la mejor. Hemos podido comprobar experimentalmente, jugando varias partidas contra el jugador automático, que esto es así.

2.1. Métrica 4

Bajo esta métrica, el jugador automático insertará la ficha en una columna que tenga hueco, elegida aleatoriamente.

2.2. Métrica 3

Utilizando esta métrica, el jugador automático seguirá los siguientes pasos para elegir la columna en la que insertar:

1. Primero intentará buscar una columna en la que insertando su ficha puede ganar.

2. En caso de que no exista dicha columna, buscará columnas de tal forma que al insertar su ficha, se generen tableros en los que el jugador humano no pueda ganar en ese turno, insertándola en una comlumna aleatoria que cumpla dichas condiciones.

3. Si no existe ninguna columna como la descrita en el paso anterior, el jugador humano podrá ganar sin importar dónde insertemos la ficha, por lo que insertamos la ficha en la primera columna en la que sea posible.

Exploramos un segundo nivel del árbol con el fin de evitar programar una función que compruebe si el jugador humano tiene tres fichas alineadas y un hueco libre donde insertar una cuarta que le haga ganar la partida. Estamos primando la **simplicidad** frente a la **eficiencia**.

2.3. Métrica 2.

En esta métrica, el jugador automático se encarga de asignar a cada subárbol que cuelga de cada uno de los hijos del tablero actual una puntuación. Las puntuaciones se asignan de la siguiente forma:

- Por cada tablero en el que el jugador automático pierde, la puntuación disminuye 2 puntos.

- Por cada tablero en el que el jugador automático empata, la puntuación aumenta 1 punto.

- Por cada tablero en el que el jugador automático gana, la puntuación aumenta 2 puntos.

Se tiene en cuenta también el nivel en el que se encuentran cada uno de estos tableros para ajustar la puntuación, dando más importancia a conseguir una victoria o un empate en niveles más altos.

El jugador automático introducirá la ficha en la columna correspondiente al tablero cuyo subárbol tenga más puntuación.

2.4. Métrica 1.

En esta métrica, el jugador automático decide en qué columna debe insertar una ficha de la siguiente forma:

- Primero, comprueba si puede insertar una ficha que le haga ganar la partida, en cuyo caso lo hace.
- En otro caso, computa un vector con todos los posibles nodos en los que insertar una ficha en su tablero asociado no le haría perder la partida (es decir, no provocaría que el jugador humano ganase fácilmente al turno siguiente).
- Para los nodos de este vector, comprueba si existe un tablero asociado en el que tiene una alineación de 3 fichas. En caso afirmativo, elige la columna cuyo tablero asociado tenga **el mayor número** de alineaciones de 3 fichas.
- En caso negativo, descarta del vector de nodos posibles, aquellos nodos que desemboquen en la siguiente jugada en una alineación de 3 fichas por parte del jugador humano.
- Ahora, repite los últimos dos pasos, pero buscando alineaciones de 2 fichas.
- Si aún así no ha elegido aún dónde insertar, se elige la columna utilizando el mismo criterio de puntuaciones que en la **métrica 2**, pero eligiendo únicamente de entre los nodos presentes en el vector de posibilidades.

3. Posibles mejoras

3.1. Mejorar el sistema de puntuaciones

El sistema elegido para puntuar un tablero es muy simple: únicamente tiene en cuenta si se gana, empata o pierde, y en qué nivel del árbol de soluciones sucede la jugada. Una posible mejora para la **métrica 1** sería conseguir un sistema de puntuaciones que mirase, en un tablero dado, el número de alineaciones de 3 fichas y de 2 fichas, además de lo que ya comprueba con el sistema antiguo. Asignando a cada tablero una puntuación ponderada según estos nuevos parámetros, creemos que conseguiríamos una mejora significativa en la elección de columna.