THE BABEL TOWEL PUZZLE

Inteligencia Artificial – IC6200

Descripción

Documento que describe la descripción matemática de la función de evaluación implementada con el algoritmo A*.

Tecnológico de Costa Rica

Profesor Jorge Vargas

Semestre II – 2020

Jean Paul Barrit 2016014628 – Gabriel Quesada 2017126064

Índice

Descripción del problema

Se requiere desarrollar una inteligencia artificial que solucione el juego de "*la torre de Babel*". Tiene que solucionarlo con el algoritmo A*. Debe tener una interfaz gráfica en el que el usuario humano pueda interactuar de manera sencilla. Al usuario se le solicita la posición inicial de las bolitas en el juguete (tomando en cuenta que la bolitas no están marcadas, así que no se le puede pedir al usuario que las individualice), y también se le pedirá la configuración final; el programa lo pensará un rato para, finalmente, decirle al usuario, paso a paso, cómo debe manipular el juguete para llegar a la configuración final. Es muy importante tomar en cuenta que se va a suponer que la sección gruesa sin bolitas (en donde está la muesca vacía) es la parte de arriba. Los estudiantes deberán crear asimismo un pequeño lenguaje formal para describir el juguete y su manipulación, pues se exigirá también que ya sea la configuración inicial o la meta, o ambas, puedan ser dadas mediante un archivo tipo txt. Asimismo, si el usuario lo desea, las configuraciones inicial y final, y las instrucciones de armado pueden quedar en un archivo de texto.

Descripción de la solución

A continuación, se van a detallar el paso a paso de cómo se llegó a la solución del problema descrito anteriormente, recalcar que la solución de este problema fue desarrollada en C#.

Utilizando la función F que va del conjunto de nodos del grafo a un conjunto numérico, y que se define como:

$$F(N) = g(N) + h(N)$$

Donde *g(N)* es la función que brinda el costo óptimo de ir desde el nodo inicial al nodo N, y *h(n)* es la función que brinda el costo óptimo de ir desde el nodo N hasta el nodo meta.

Pero como F(n) es no es directamente calculable, se trabaja con una estimación que se define:

$$\hat{F}(N) = \hat{g}(N) + \hat{h}(N)$$

- **Función de costo de transición:** Como medida de coste se escoge la cantidad de movimientos necesarios para pasar de un estado al inmediatamente siguiente. Se define la función que nos da el costo exacto en cada arco del grafo. En este caso encontramos que *C(ni,nj)=1*, para cualesquiera *ni,nj*.
- Entonces, $\hat{g}(N)$ es la suma de los costos $C(N_i,N_j)$ en una ruta.
- Este punto es sencillo de calcular, por cada ronda de movimientos que avanza, se incrementa un i (siendo este un contador que empieza en 0), esto quiere decir que, por cada posible jugada, se incrementa mi lista

de sucesores para saber cuántas jugadas se han hecho a partir del anterior:

```
public int calculateCost() {
    return this.sucesors.Count - 1;
}
```

Ilustración 1: Función que calculo el costo de un nodo

- **Función heurística de costo futuro:** Para esta función se realizó exitosamente con la distancia de euclidiana. Es un poco complicado la manera en que adaptó para este problema, sin embargo, se pudo lograr. Primero, para hay que explicar qué es la distancia de euclidiana. En un espacio bidimensional, la distancia euclidiana entre dos puntos P1 y P2, de coordenadas cartesianas (x1, y1) y (x2, y2) respectivamente, es:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Ahora bien, sabiendo esto, claramente no se podría aplicar a este proyecto, porque no solo se tiene un punto destino, sino varios, donde un punto *P1* se compara con otros cuatro puntos de la matriz destino *P2, P3, P4, P5*, que representan todos los espacios en los que el punto *P1* podría estar. Luego de calcular la distancia euclídea entre un punto y todas las posibles posiciones para ese punto, se calcula el valor mínimo entre los puntos y ese es valor de distancia del punto en evaluación.
- Lo que se hizo fue lo siguiente, calcular por cada punto de la matriz de origen se le asigna un valor de distancia, que se calcula con mínimo de las distancias euclídeas en sus posibles posiciones. Luego se suman todos los valores de distancia y da como resultado la distancia total entre la matriz origen y la matriz destino. De modo que:

$$h(n) = \sum_{i=0}^{4} \sum_{j=0}^{4} d_E(P_{i,j}, P')$$

- Siendo $P_{i,j}$ un punto en la matriz de origen y P' el punto de la ubicación en la que debería estar $P_{i,j}$. Por lo tanto, cuando todos los puntos estén en el lugar que corresponda $h(n) = 0$.
- A continuación, los algoritmos implementados en el lenguaje C#:

```
private List<int[]> GetIndexBySimbol(string[,] matrix, string simbol)
{
    List<int[]> list = new List<int[]>();
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (matrix.GetValue(i, j).ToString() == simbol)
            {
                list.Add(new int[2] { i, j });
            }
        }
    }
    return list;
}
```

Ilustración 2 Función que retorna la lista de puntos de un color

```
public double CalculateHeuristFunction()
{
    double count = 0;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            string initialValue = matrix.GetValue(i, j).ToString();
            List<int[]> points = GetIndexBySimbol(finishMatrix, initialValue);
            switch(points.Count)
            {
                case 1:
                    count += Pitagoras(i, j, points[0][0], points[0][1]);
                    break;
                case 3:
                    count += Math.Min(Pitagoras(i, j, points[0][0], points[0][1]),
                        Math.Min(Pitagoras(i, j, points[1][0], points[1][1]), Pitagoras(i, j, points[2][0], points[2][1])));
                    break;
                case 4:
                    count += Math.Min(Pitagoras(i, j, points[0][0], points[0][1]),
                        Math.Min(Pitagoras(i, j, points[1][0], points[1][1]),
                        Math.Min(Pitagoras(i, j, points[2][0], points[2][1]), Pitagoras(i, j, points[3][0], points[3][1]))));
                    break;
            }
        }
    }
    HeuristValue = count;
    return count;
}
```

Ilustración 3 Función encargada de calcular la función heurística

- **Demostración de admisibilidad de la función de costo total estimado:**

Entonces para que $\hat{h}(N)$ sea admisible se tienen que cumplir las siguientes condiciones de admisibilidad:

1. Cada nodo del grafo tiene un número finito de sucesores
2. El costo de cada arco del grafo es mayor que una cierta cantidad positiva ε.
3. Para todos los nodos del grafo se cumple que $\hat{h}(n) \leq h(n)$. Esto es, $\hat{h}$ es un estimador optimista.

Según la demostración de $\hat{h}(N)$ calculada anteriormente, se puede concluir:

El requisito 1 se cumple porque para cada nodo hay a lo sumo 12 sucesores.

El requisito 2 se cumple porque ningún costo puede ser menor a 1

Y el requisito 3 se concluye informalmente del hecho de que, en la realidad, tratar de poner una pieza en su sitio correcto podría sacar de lugar a otras de su sitio, aumentando el número de movimientos a efectuar.

Por lo tanto $\hat{h}(N)$ es admisible.