Memoria Algoritmo A Estrella

Nerea Jiménez González
Yhondri Acosta Novas

Marzo 2020

UNIVERSIDAD
COMPLUTENSE
MADRID

Índice

1. Detalles de la implementación **3**
1.1. Lenguaje utilizado 3
1.2. Procedimiento seguido para la implementación 3
1.3. Ampliaciones realizadas 4
1.3.1. Casillas con obstáculos 4
1.3.2. Casillas con penalización 4
1.3.3. Casillas de paso obligatorio 5
1.4. Otros elementos de interés 5

2. Código ejecutable **5**
2.1. Simulación ejemplo enunciado 5
2.2. Simulación sin obstáculos 5
2.3. Simulación con penalización 5
2.4. Simulación con waypoints 5

3. Manual de usuario **5**

1. Detalles de la implementación

1.1. Lenguaje utilizado

Para la implementación, se utiliza el lenguaje JAVA.

1.2. Procedimiento seguido para la implementación

El procedimiento que se ha seguido para la implementación ha sido el siguiente:
En primer lugar, se resolvió el ejemplo dado en el enunciado, y se comenzó a programar este ejemplo en concreto. Durante esta explicación, nos referimos a nodos y casillas. El nodo representa la casilla, por lo que puede que se utilicen ambas denominaciones en algún momento.
Empezamos por definir las entidades a tener en consideración, las cuales son:

- **Coordinate**
  Coordenada compuesta por fila y columna, para evitar confusiones a la hora de programar.
- **Coordinate type**
  Enumerado utilizado para saber que tipo de movimiento es para dibujar en el tablero este.
- **Node**
  Contiene toda la información del nodo: g, h, f y coordenada, que es la casilla del tablero en cuestión.

Comenzamos por las comprobaciones a donde se puede expandir el nodo de forma que, no se puede expandir donde:

- Casillas inaccesibles
  Hablaremos sobre ellas en el apartado de ampliaciones.
- Casillas fuera del tablero

Una vez tenemos las casillas a donde podemos avanzar, estas las decidimos guardar en una cola de nodos de prioridad ordenada de menor a mayor por f. Esta cola, *openNodesPriorityQueue*, se correspondería a la lista abierta.
Para guardar los nodos que vamos cerrando, lo que se correspondería a la lista cerrada, utilizamos una lista de coordenadas, *closedCoordinateList*, ya

que una vez que cerramos un nodo, sólo nos interesa su coordenada, que sería como el nombre del nodo.
El algoritmo en sí es un bucle *while*, del cual se sale cuando no nos quedan nodos que recorrer en *openNodesPriorityQueue*, o bien cuando el nodo meta, *goalNode*, es null.
Si se sale del bucle porque no tenemos para nodos que recorrer, y no se ha llegado a la casilla meta, *goalNode=null*, no existirá un camino.
Cuando las coordenadas del *goalNode* coinciden con las coordenadas del nodo Meta, *goalCoordinate*, se ha encontrado el mejor camino posible y el algoritmo termina.

1.3. Ampliaciones realizadas

Las ampliaciones realizadas son las siguientes:

1.3.1. Casillas con obstáculos

Son las casillas nombradas como **Pared**. Estas casillas actúan como un muro, y no se puede pasar por ellas.
Para esta ampliación, se ha utilizado una lista, *obstacleCoordinateList*, en donde se guardan las coordenadas de los obstáculos. Cada vez que se intenta acceder a una nueva casilla, se consulta si dicha coordenada está en la lista de obstáculos de forma que si está, no se puede expandir a esta casilla. En caso contrario, sí se puede.

1.3.2. Casillas con penalización

Estás casillas conllevan una penalización al pasar por ellas, por lo que puede que el primer camino que pensamos que es el óptimo para llegar a la casilla meta, deje de serlo por esto. Las casillas están nombradas como **Barro**, y la penalización sobre la f del nodo es de 10.
Para esta ampliación, hemos utilizado un hashmap, *penaltyMap*, en el cual guardamos la coordenada de la casilla junto a su penalización, por si se quiere ampliar de forma que cada casilla con penalización tenga diferentes penalizaciones. Al igual que con las casillas con obstáculos, cada vez que se quiera expandir a una casilla se consulta si esta coordenada está en el hashmap. En el caso de estar, se aplica a la f la penalización.

1.3.3. Casillas de paso obligatorio

Estas casillas son casillas por las que hay que pasar obligatoriamente en nuestro camino en el orden que el usuario las coloca, *waypoints*. Estas casillas están nombradas como **Castillo**.
Para esta ampliación, hemos utilizado una lista, *waypointlist*, en donde guardamos las coordenadas de los puntos donde hay que pasar obligatorioamente. Introducimos las coordenadas en el orden que el usuario las coloca, y se sacan en el mismo orden. Lo que hacemos es hacer que el primer waypoint sea nuestra meta provisional, de forma que el árbol se expande hasta encontrar un camino. Una vez llegados a esta meta, pasa a ser el inicio y el siguiente waypoint la meta. Se sigue de esta forma hasta que no nos quedan waypoints, y nuestra meta es la que se colocó inicialmente como tal.

1.4. Otros elementos de interés

2. Código ejecutable

Se adjunta en el archivo tipo zip con nombre **Código** zip, y un ejecutable tipo jar con nombre **Ejecutable**.

2.1. Simulación ejemplo enunciado

2.2. Simulación sin obstáculos

2.3. Simulación con penalización

2.4. Simulación con waypoints

3. Manual de usuario

Para empezar con la aplicación, clickeamos dos veces sobre el archivo ejecutable.
A la hora de marcar las casillas, seleccionaremos en el menú izquierdo, bajo los botones de iniciar y reiniciar, el tipo de casilla a dibujar.
Una vez seleccionado el tipo de casilla que queremos, clickeamos las casillas que queremos que sean de dicho tipo. Los tipos de casilla son los siguientes:

- **Inicio**: marca donde va a comenzar.
- **Meta**: marca el final.
- **Pared**: casillas por las que no se pueden pasar.
- **Barro**: casillas que tienen penalización al pasar por ellas.
- **Castillo**: casillas por las que es obligatorio pasar antes de llegar a la meta.

Es obligatorio poner un inicio y una meta.
Una vez satisfechos con nuestro tablero, clickeamos en el botón *Empezar*.
Si queremos limpiar el mapa, clickeamos sobre el botón *Reiniciar*.
Las casillas exploradas aparecerán de color rosa magenta.