IA: Práctica Algoritmos de Búsqueda

Rafael Rey y Álvaro Lalinde

Objetivo

Implementar la búsqueda de caminos en el proyecto proporcionado para que los enemigos puedan moverse por el mapa.

Implementación de AStarGraph

La clase AStarGrapth escaneará el mundo para generar el grafo asociado. Se recorre el mundo de juego, detecta aquellas posiciones que tienen algún obstáculo y guarda la información de cada Nodo. Esta información se guarda en las clase AStarNode y AStarConnection.

-AStarNode:

<public Vector2 position> Vector con la posición.

<public AStarConnection north, north_east, east, south_east, south, south_west, west, north_west> Posibles conexiones

<public List<AStarNode> vecinos> Lista con los vecinos acesibles

-**AStarConnection:**

<public float distance> distancia de la conexión

<public AStarNode otherNode> Información del vecino

El metodo Scan de la clase AStarGraph es el encargado de escanear el mapa e ir guardando la información. Para ello usaremos dos matrices, una de AstarNode y otra de bool. En la matriz de AStarNode guardamos la información del Nodo y en la matriz de bools guardamos si la posición es un obstáculo.

En el método Scan recorremos dos veces el mapa. En la primera guardaos la información de si es obstáculo o es accesible y en la segunda guardamos la información de los nodos y sus vecinos para establecer las posibles conexiones.

Después de aplicar este método tendremos guardada la información de los nodos y sus posibles conexiones, de manera que ya podemos ponernos a calcular el camino más optimo con el algoritmo A*.

Implementación de AStarNavigator

Le pasamos como parámetros la posición inicial y final del camino y nos devuelve las coordenadas de cada punto por el que pasará el camino obtenido.

Para obtener este camino implementamos el algoritmo A*, que combina el coste real(g) para llegar al nodo actual y el coste estimado(h) que calcula la heurística desde donde nos encontramos hasta el destino final. El camino lo escogeremos según que nodo tienen un coste mejor para ser accedidos.

```
<List<AStarNode> abiertos> Lista de Nodos a explorar
<List<AStarNode> cerrados> Lista de Nodos explordos
<var g = new Dictionary<AStarNode, float> Coste Real para llegar al nodo
<float h> Distancia estimada al destino
<var f = new Dictionary<AStarNode, float> Suma de g y h
<Dictionary<AStarNode, AStarNode> nodopadre> Diccionario con el dodo padre como
Value
<AStarNode posicionActual> posición que estamos analizando
```

Ya está todo definido, podemos ir al grano. Lista de pasos del algoritmo A*:

1.Cogemos el nodo inicial y lo metemos en la lista abierta.

2.Cogemos de la lista abierta, el nodo con menor valor de f (coste total).

3.El elegido (nodo activo) lo pasamos a la lista cerrada.

4.Cogemos los nodos vecinos del nodo activo y con cada uno hacemos lo siguiente:

– Si no está en la lista abierta: Lo metemos, le ponemos como padre el nodo activo y le calculamos los valores de g, h y f.

– Si ya está en la lista abierta: Verificamos si el camino por el que acabamos de llegar es mejor que el camino por el que llegamos anteriormente. Para eso vemos si su g es mejor que la g que le correspondería ahora. Si su g es mayor que la nueva g, es que el camino actual es mejor, así que le ponemos como padre el nodo activo y le asignamos los nuevos valores de g, h y f.

5.Si alguno de los nodos del punto 4 era el nodo final, hemos terminado.

6.Si quedan nodos en la lista abierta, volvemos al punto 2

7.Si no quedan nodos en la lista abierta y no hemos llegado al final, el destino es inalcanzable.

Por cierto, cuando lleguemos al final, hay que rehacer el camino completo desde el último nodo hasta el primero mirando la información del padre de cada nodo. Para ellos implementamos el método construircaminofinal, donde iteramos sobre el diccionario nodopadre y cambiando actual por el valor del padre del anterior. Por último, le damos la vuelta a la Lista para mandar el camino con el sentido correcto.

En AStarNavigator también implementamos tres métodos para calcular g, f y h. Y otro para conseguir el nodo con menos valor de f que necesitamos en el paso 2 del A*.

Conclusiones

Cuanto mayor es el mapa más le cuesta al algoritmo encontrar mejor camino. Para mejorar este problema creemos que sería necesaria una revisión del escaneo del mapa para intentar reducir costes y tiempos.