Técnicas de los Sistemas Inteligentes

Práctica 1:

Desarrollo de un agente basado en búsqueda heurística para el entorno GVGAI

Curso 2019-20

Javier Gálvez Obispo

javiergalvez@correo.ugr.es

Grupo 2, Viernes 17:30 – 19:30

1. Comportamiento deliberativo

1.1. Simple

Nos encontramos ante un problema sencillo en el que queremos que nuestro agente se mueva hasta el portal de la forma más rápida posible. Para ello se ha implementado el algoritmo A* utilizando como heurística la distancia Manhattan. Utilizamos ésta heurística ya que es la distancia que mejor se adapta a nuestro mundo cuadriculado. Ya que nos encontramos antes un problema muy simple no es necesario implementar ninguna variante del algoritmo A*.

Para la implementación del algoritmo es necesario crear una clase que represente los nodos, implementada en Nodo.java. Cada nodo almacena la posición y la dirección del agente, el coste hasta llegar a esa situación y la distancia Manhattan entre la posición actual y el objetivo. Además, también guarda la información sobre su nodo padre y la acción que se realizó para alcanzar ese estado. Para la implementación del algoritmo A*, se considera que dos nodos son iguales cuando la posición y la dirección del agente son la misma.

El algoritmo A* se encuentra implementado en el archivo Pathfinder.java y está dividido en varios métodos *calc_path, calc_neighbours, isObstacle,* y *astar* para facilitar la comprensión del código.

- *calc_path* devuelve el camino a tomar una vez el algoritmo A* termina.
- *calc_neighbours* devuelve todos los hijos de un nodo.
- *isObstacle* devuelve si una posición es un obstáculo o si se encuentra fuera del mapa.
- *astar* contiene la implementación del propio algoritmo y hace uso de los tres métodos anteriormente descritos.

1.2. Compuesto

Añadimos ahora la necesidad de recoger diez diamantes repartidos por el mapa antes de poder utilizar el portal para acabar el nivel. Para resolver este problema se ha seguido una estrategia *greedy*, el agente va a por el diamante que tenga más cerca según la distancia Manhattan. Ésta solución no es la más eficaz pero es la única factible debido a las limitaciones de tiempo que tiene nuestro agente. Encontrar el camino más corto entre dos puntos pasando obligatoriamente por otros diez puntos distintos es intratable en tan solo 40ms.

2. Comportamiento reactivo

2.2. Simple

Nuestro objetivo ahora es sobrevivir durante toda la partida teniendo que escapar de un enemigo con movimientos pseudoaleatorios. Para esto añadimos una representación extra del mapa para reflejar lo peligrosa que es una posición, un mapa de calor. Las posiciones seguras se representan con un 0 y las que suponen un peligro con un 1. Las casillas en una circunferencia de radio 3 alrededor de un enemigo son consideradas peligrosas. Un ejemplo de la representación del mapa de calor podría ser el siguiente:

0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	1	1	0	0
0	0	0	0	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	0
0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	1	0	0	0

donde la posición en verde representa a nuestro agente, la roja al enemigo y en amarillo se encuentran las posiciones que suponen peligro.

Se ha utilizado un radio de 3 ya que nuestro agente gasta una acción en girarse mientras que los enemigos no. El mapa es lo suficientemente grande como para permitir este rango y así nuestro agente es más seguro.

Cuando el agente se encuentra en una posición marcada con un 1 entra en el estado de peligro, observa por donde se le está acercando el enemigo y se mueve en dirección contraria. Utilizando el mismo ejemplo de antes, el agente se movería hacia abajo a la izquierda para huir del enemigo que se aproxima por arriba a la derecha. Para esto hacemos uso del algoritmo A* implementado en el primer nivel pero cambiando nuestro objetivo por la esquina del mapa que se encuentra en la dirección opuesta al enemigo.

Hay que tener especial cuidado cuando el enemigo nos acorrala en una esquina del mapa. Para evitar una situación así, el agente actúa de forma diferente si se encuentra cerca de una. Si, por ejemplo, nos estamos acercando a la esquina inferior derecha y el enemigo se aproxima hacia nosotros por arriba a la izquierda, en un caso normal huiríamos hacia la esquina inferior derecha, pero, al encontrarnos cerca de ésta, conseguiríamos que el enemigo nos acorrale. Para evitar que esto ocurra corremos hacia la esquina inferior izquierda o la superior derecha según cuál sea la opción más segura.

Tabla 1: Caso 1

1	1	1	1
1	1	1	1
0	1	1	1
0	0	1	0

Tabla 2: Caso 2

1	1	1	0
1	1	1	1
1	1	1	1
1	1	1	1

En el caso 1, el agente huiría hacia la izquierda mientras que, en el caso 2 lo haría hacia arriba.

2.2. Compuesto

Complicamos la situación añadiendo un nuevo enemigo, es decir, ahora tenemos que sobrevivir escapando de dos enemigos simultáneamente.

Debido a los movimientos pseudoaleatorios de los enemigos, los casos en los que el agente se encuentra amenazado por los dos al mismo tiempo no son abundantes a lo largo de la partida. Por lo que cada vez que esto ocurra reduciremos el problema al de un enemigo, es decir, si nos viene un enemigo por arriba y otro por abajo, el agente actuará de la misma forma que lo haría si le viniese un sólo enemigo por un lado. El lado por el que se le acerca el enemigo dependerá de cuál sea la opción más segura para huir, igual que ocurría en las esquinas en el caso anterior.

Para poder simplificar el problema de ésta manera tenemos que modificar la representación del mapa de calor. Bastará con representar con un 2 las zonas que estén cerca de dos enemigos al mismo tiempo y seguir marcando con un 1 las que solo tengan un enemigo cercano.

3. Comportamiento reactivo-deliberativo

Para acabar, ahora queremos que el agente pueda cumplir los objetivos de los apartados 1.2. y 2.1. al mismo tiempo. Para ello unimos los dos comportamientos descritos haciendo pequeñas modificaciones.

A partir de ahora, el algoritmo A* tendrá en cuenta las posiciones de los enemigos cuando vaya a calcular el camino a un diamante o al portal. Añadiendo el valor del mapa de calor de esa posición al coste de un nodo logramos que el agente evite pasar cerca de enemigos. El problema con esta solución es que no podemos predecir las posiciones de los enemigos, por lo que recalcularemos el camino a nuestro objetivo cada pocos pasos. Así, tendremos un camino a seguir que se actualiza cada poco tiempo.

Debido a la naturaleza *greedy* del algoritmo que nos dice que diamante coger primero, es posible que tras huir de un enemigo el diamante más cercano al agente sea otro distinto. Obviamente, el agente cambiará de objetivo y calculará el camino al nuevo diamante más cercano.