PRÁCTICA 2: LOS EXTRAÑOS MUNDOS DE BELKAN:

Fichero de Documentación

Álvaro Vega Romero

Inteligencia Artificial

0) Nivel 0:

Para realizar este nivel, ya teníamos el método pathFinding_Profundidad, pero lo más importante es pararte a entender que se hace en esta función, ya que, si no, cuando lleguemos a próximos niveles no entenderemos muchas cosas.

Luego, lo que nos falta por implementar en este nivel, es el método think, en el cual, basándonos en el tutorial, podemos ver que tenemos que comprobar si hay plan activo o no, y en caso de que no haya, generarlo llamando al método pathFinding, en otro caso, sacamos la primera acción que se encuentre en el plan y así hasta llegar al nodo objetivo.

1) Nivel 1:

El único cambio implementado respecto al nivel anterior ha sido aplicar una cola para abiertos en vez de una pila, para así, poder hacer el algoritmo de búsqueda en anchura que consiste en ir viendo nivel a nivel si se ha llegado al nodo objetivo de la forma más óptima (en número de pasos) en vez de ir analizando un sucesor del nodo de mayor nivel generado hasta el momento.

Por tanto, al modificar eso, debemos cambiar el uso del método top de la pila que teníamos antes a front sobre la cola que tenemos actualmente.

Así, el método se llama pathFinding_Anchura y recibe los mismos parámetros que la anterior.

2) Nivel 2:

Lo primero que realizamos en este nivel fue el método Coste, para ver el coste de un nodo junto a añadir unos atributos al struct estado para ver si un jugador tiene bikinis o zapatillas que nos servirá para el método anteriormente mencionado (Coste).

Así, con el método coste, sabiendo el tipo de acción que se realiza, sobre que nodo se realiza y si tenemos el bikini o las zapatillas, sabremos el coste que tiene ese nodo a explorar y luego, como la cola de abiertos esta ordenada de forma que los nodos de menos coste van primeros (priority queue), podremos ir haciendo un algoritmo de costo uniforme, cogiendo el top de abiertos en el que se expandirá el nodo de la frontera con menor coste.

Luego el algoritmo termina cuando se llega al nodo objetivo y se ha avanzado por el camino menor coste (según la batería).

Destacar que se ha hecho la función ComparaEstadosCU pero adaptada a los nuevos nodos para que en caso de comparación si dos nodos son iguales (orientación y posición), también se tenga en cuenta si tiene bikini o si tiene zapatillas (esto es para poder insertar el nodo en abiertos, pero solo si hemos conseguido las zapatillas/bikini podemos "repetirlo").

3) Nivel 3:

Para este nivel, tenemos que encontrar el camino óptimo para tres objetivos y hacerlo mediante el algoritmo A*.

Para este algoritmo, lo primero que añadiríamos es un método que se llama mejorDistManhattan que devuelve la menor distancia Manhattan que hay hasta un objetivo que aún no haya sido visitado. Esta parte será la que forme la heurística (h(n)). Respecto al coste, este seguirá siendo igual que en el nivel 2. (g(n)).

Luego, una vez tenemos calculados estos dos valores, ya podemos calcular f(n) y con este, ordenar la cola de prioridad para procesar y expandir el nodo cuyo valor de f(n) sea el menor.

También tenemos que comentar, que al estado de los nodos se le añade un vector de booleanos con 3 componentes, para indicar si se ha visitado el objetivo i, además de un entero que se incrementa en una unidad cada vez que se pone true alguna componente del vector.

Por último, y no, por ende, menos importante, el compara estados, también hemos tenido que modificarlo, añadiéndole una sentencia, que además de comparar lo mismo que en el nivel 2, también compare si el número de objetivos es el mismo. (Así, puede repetirse en cerrados un nodo, si el número de objetivos que se lleva en ese nodo, es diferente al número de objetivos que lleve "el mismo").

4) Nivel 4:

No implementado.