Belkan's World

1. Contenido general

Para comenzar, la tarea de mi agente es realizar una búsqueda intensiva por todo el mapa de manera que logre un alto porcentaje de acierto con respecto al mapa solución en todos los casos. Procederé a explicar brevemente el proceso que sigue mi agente en esta práctica, centrándome sobre todo en el método *think()* de la clase *Agent*.

2. Búsqueda de los puntos PK.

La primera tarea de nuestro agente reactivo es buscar los puntos de color amarillos, denominados *PK*. Estos puntos son de vital importancia ya que contienen las coordenadas correspondientes al mapa solución de ese punto en concreto. Por tanto, nuestra misión será almacenar éstas coordenadas en variables para no perderlas. Estas variables en mi caso se llaman *f_PK1, c_PK1, f_PK2, c_PK2.* Además, debemos de guardar el contenido de nuestras coordenadas relativas ya que serán necesarias para el posterior cálculo de la matriz solución. Estas variables son f_myPK1, c_myPK1, f_myPK2, c_myPK2.

Una vez tenemos ambos puntos localizados en nuestro mapa procedemos al siguiente punto, rotar la matriz.

3. Rotación del mapa solución.

Una vez tenemos los dos puntos clave, procedemos a rotar nuestro mapa. Sabemos en qué sentido tenemos que girar la matriz ya que podemos calcular unas variables *f1,f2,c1,c2 (*mirar código) y con estas comprobamos que:

- *Si f1==f2 Estamos orientados, no hace falta rotar.*
- *Si f1==-f2 Estamos boca abajo, girar 180º.*
- *Si f1==c2 Estamos girados hacia la izquierda, girar 90º derecha.*
- *Si f1==-c2 Estamos girados hacia la derecha, girar 90º izquierda.*

Sabiendo esto, tan solo llamamos al método *giraMatriz(),* que gira la matriz 90º hacia la izquierda, el número necesario de veces según el ángulo que debamos girar.

4. Ajuste de mapas.

Este apartado es de los más importantes ya que en él se decide la nota de la exploración de nuestro mapa al comparar nuestro entorno con el mapa secreto.

Para comenzar, una vez encontrado ambos PK's, se activa el booleano *foundPKS*. Con este booleano damos paso a que se vaya ajustando la matriz de nuestro entorno al mapa secreto en cada iteración, por ello, incluímos la llamada a la función *ajustaMatriz()* en el método *ActualizarInformacion* de *Agent*.

Esta función lo que hace es calcular cada vez que es llamada las coordenadas clave para la matriz de 100x100, llamadas *esquina_primera_f, esquina_primera_c, esquina_segunda_f, esquina_segunda_c.* Estas dos primeras variables se refieren a la esquina superior izquierda y las otras a la esquina inferior derecha, delimitando de esta manera el cuadrado de la matriz solución. Después va rellenando con las coordenadas relativas de nuestra matriz entorno, las coordenadas recalculadas de nuestra matriz solución y actualizándose así en cada paso.

5. Exploración del mapa.

La exploración del mapa se realiza con una búsqueda de la posición menos frecuentada de todas las que rodean directamente a nuestro agente, es decir, *norte, este, oeste y sur*. Para ello disponemos de una matriz de enteros de tamaño 200x200 (mismo tamaño que el mapa entorno) llamada *frec_pasos.* En esta matriz se incrementa en 1 la coordenada correspondiente a la posición [*y_, x_]* de nuestro agente.

Además de la matriz de frecuencias necesitamos de 4 funciones auxiliares que devuelvan el valor de la matriz *frec_pasos* de las casillas correspondientes. Estas funciones se llaman *casillaDelante(), casillaDetras(), casillaIzquierda(), casillaDerecha()* y además si la casilla que tenemos delante es un obstáculo establece su valor en la matriz *frec_pasos* a un valor muy elevado para asegurarse que no intenta ir a por él más, con el consiguiente ahorro de pasos..

Por tanto, nuestro agente evaluará las casillas mencionadas anteriormente, junto a otras 2 funciones que se encargan de devolver si existe un obstáculo estático o dinámico, llamadas *obstaculo_estatico* y *obstaculo_movil,* y decidirá hacia cual debe de desplazarse. Cabe destacar que siempre que elige la casilla de la derecha o de la izquierda, se activa un booleano llamado *needFORWARD* cuya única misión es asegurarse de que nuestro agente avanza en su siguiente iteración, a no ser que haya un obstáculo, que en este caso gira y lo vuelve a comprobar. De esta manera forzamos al agente a que cuando gire, avance también.

También mencionar que cuando elige la casilla de detrás se activa otro booleano, llamado esta vez *needBACK* y cuya única misión es asegurarse de que nuestro agente hace un giro de 180º y avanza.

6. Recogida de objetos

Nuestro agente está programado para que recoja tan solo los objetos que le ayudan a desplazarse por el mapa, es decir, las zapatillas y el bikini. Además está programado de tal manera que una vez que logra tener los dos objetos, a través de una secuencia de condicionales y booleanos que producen que:

- Si nuestro agente no ha encontrado ningún objeto, esquiva todos los obstáculos.
- Si nuestro agente ha encontrado solo las zapatillas, las equipa y ahora los obstáculos no son un obstáculo.
- Si nuestro agente ha encontrado solo el bikini, lo equipa y ahora el agua no es un obstáculo.
- Si nuestro agente ha encontrado ambos objetos, equipa el último que haya recogido hasta que detecte que necesita el otro, en este caso, inicia una secuencia para desequipar el actual y equipar el otro. Por tanto, una vez que dispone de los dos objetos, siempre se asegura de que lleva equipado el necesario para la zona que tiene delante.

7. Opinión personal.

En mi opinión creo que esta práctica hubiera necesitado algo más de explicación en clase a nivel general, ya que había bastantes cosas que resultaban inciertas a rasgos generales como era el oso, si los personajes se movían incluso si les querías dar un objeto...

Autor: Sergio Cervilla Ortega **DNI: 14275233F**