Técnicas de los Sistemas Inteligentes

PRÁCTICA 1:

DESARROLLO DE UN AGENTE BASADO EN HEURÍSTICA PARA EL ENTORNO GVG-AI

Autor: José Javier Alonso Ramos
Grupo: B

DESCRIPCIÓN GENERAL DEL ALGORITMO

Función act:

{

{ vector_de_casillas_a_evitar := casillasQueLiberanAEnemigos();

{ Para cada enemigo tomamos sus 4 casillas colindantes y la suya

{ propia y las guardamos en un vector para tenerlas en cuenta a la hora

{ de crear un camino (deliberativo) y a la hora de realizar un

{ movimiento (reactivo).

{

Deliberativo { path := obtencionDelCaminoASeguirConA*();

{ Creamos un nodo inicial con las coordenadas de nuestro personaje y

{ vamos generando hijos hasta alcanzar un nodo con las mismas

{ coordenadas que nuestra casilla objetivo. Los nodos escogidos para

{ formar el camino serán aquellos con el menor valor de su función 'f'

{ el cual se calcula mediante distintos factores explicados más adelante.

{

{ movimientos := transformarPathAAcciones();

{ Dado que path es una lista de nodos debemos adecuar esto a un tipo de

{ dato que el programa entienda (acciones). Para ello, teniendo en

{ cuenta datos como la orientación del personaje, nuestra próxima

{ casilla a recorrer o nuestra última acción realizada, iremos cambiando

{ nuestros nodos por acciones y las incluiremos en el vector

{ movimientos.

{

{ sig_mov := movimientos.poll();

{ Tomamos el primer movimiento de la lista de movimientos.

{

{ if (tieneRocaEncima(sig_mov) or enemigoCerca(sig_mov)) then:

Reactivo { alejarse;

{ recalcularPath;

{ Si el resultado de nuestro próximo movimiento nos pone en peligro

{ debemos intentar escapar de esa situación (deshaciendo nuestras

{ anteriores acciones o moviéndonos a casillas sin peligro) y, además,

{ volver a calcular un nuevo recorrido que posiblemente sea más

{ seguro.

return sig_mov;

Devolvemos la acción a tomar.

COMPORTAMIENTO REACTIVO:

x:= Jugador.x
y:=Jugador.y
Creamos variables con las coordenadas de nuestro Jugador.

x, y := simularSiguienteMovimiento();
Actualizamos las coordenadas suponiendo que hemos realizado el movimiento que hemos seleccionado como el siguiente.

if (tieneRocaEncima(x,y) or enemigoCerca(x,y)) then:
Si la casilla en la que nos encontraríamos tiene una roca encima que nos pudiese aplastar o un enemigo cerca que nos pudiera matar:

boolean sinPeligro := false;
while(not sinPeligro):
Mientras exista peligro comprobamos si nos podemos mover a las casillas laterales para evitar la roca

if(casillaLateralLibre(x,y)) then:
moverACasillaLateral(x,y);
sinPeligro := true;
Si podemos nos movemos hacia una de ellas

else
descender(x,y);
Si no podemos, descendemos una casilla para volver a comprobar si sus casillas laterales son accesibles

if(enemigoCerca(x,y)) then:
irEnDireccionContrariaAlEnemigo(x,y);
Si tenemos un enemigo cerca nos movemos una casilla en dirección contraria.

path.clear();
Eliminamos el path que tenemos
path := recalcularPath();
Calculamos uno nuevo

COMPORTAMIENTO DELIBERATIVO:

vector_de_casillas_a_evitar := casillasQueLiberanAEnemigos();
Si a la hora de crear un nodo sus coordenadas coinciden con alguna que esté almacenada en este vector su función heurística ‘h’ aumentará de valor y por tanto disminuirá su prioridad a la hora de ser escogido para formar el camino.

path := obtencionDelCaminoASeguirConA*();
A la hora de crear nodos, su función ‘g’ es igual a la función ‘g’ de su nodo padre más 1 siendo la ‘g’ del nodo inicial igual 0. La ‘h’ de cada nodo la inicializamos con la distancia Manhattan hasta la casilla objetivo. Si es un muro, una piedra o un enemigo (casillas no transitables) aumentaremos ‘h’ 90 puntos. Si tiene una roca encima aumentaremos otros 50 puntos. Y por último si es una casilla que si desapareciese podría liberar a un enemigo o es una casilla colindante a un enemigo, sumaremos 50 puntos.

Creamos un nodo inicial con las coordenadas de nuestro personaje y lo añadimos a la lista de abiertos (se trata de una cola con prioridad ordenada de menor a mayor función heurística). A partir de aquí entramos en un bucle que finaliza cuando alcanzamos el objetivo o nuestra lista de abiertos queda vacía.
Sacamos de ABIERTOS el primer nodo (al inicio del bucle será el inicial). Comprobamos si sus coordenadas coinciden con las de la casilla objetivo. Si coinciden acabamos si no, generamos los hijos de este nodo y los añadimos a ABIERTOS. Los hijos solo se generan si sus coordenadas señalan una casilla transitable y o bien, esa posición no ha sido visitada nunca o el valor actual de ‘f’ en esa casilla es mayor del que tendría generando este hijo. Una vez acabemos de estudiar un nodo lo guardamos en CERRADOS.
Una vez alcanzada la casilla objetivo devolveremos una lista de nodos generada mediante la obtención recurrente del padre del los nodos empezando desde el nodo final (el que coincide en coordenadas con el objetivo).

movimientos := transformarPathAAcciones();
Según nuestra orientación y última acción transformaremos los nodos en movimientos.
Si nuestra última acción fue nula tendremos que fijarnos en nuestra orientación para incluir los siguientes movimientos. Si la orientación coincide con la dirección a la que nos queremos mover solo añadiremos una acción con ese movimiento; si por el contrario no coincide añadiremos dos.
Si la última acción no fue nula solo tendremos que comprobar si coincide con la acción que queremos tomar: si coincide solo añadiremos un movimiento, si no coincide añadiremos dos.