MEMORIA DE PRÁCTICAS

Guillermo Sandoval Schmidt

ÍNDICE

1. Descripción de la solución

La solución general adoptada frente al problema de los “Extraños Mundos de Belkan v2” ha sido partir del la versión 1 y adaptar las reglas del mundo a las nuevas interacciones, objetos y personajes.

Respecto a la versión anterior, esta nueva ha sufrido los siguientes cambios:

-Con el fin de mejorar el porcentaje de exploración, se ha mejorado las interacciones con los objetos. En esta versión, el personaje no coge los huesos, ya que los considera un obstáculo, mientras que luego recoge solo uno de cada tipo de los demás objetos.

-Además, cuando se encuentra con obstáculos superables si se tiene equipado el objeto adecuado, si el personaje los tiene en la mochila y no se encuentra en una casilla peligrosa, decide cambiar de objetos para así poder atravesar estos obstáculos.

-Finalmente, se han retocado algunos otros métodos con el fin de mejorar la navegabilidad y desarrollo del viaje del personaje.

Nota: a pesar de haber desarrollado y estudiado teóricamente un planteamiento deliberativo para recoger regalos y entregarselos a los reyes y así cumplir misiones, me ha sido imposible llevarlo a la práctica. El planteamiento que deseaba implementar era básicamente, un algortimo A.

-Función heurística: el valor heurístico de cada casilla estaría compuesto de dos partes, la primera, una función g(x) que devolviese la distancia hasta la casilla objetivo, siendo la distancia el número de movimientos necesarios para llegar a esta (giros incluidos), e iría incrementando en 1 cada vez que se desplegasen los nodos a partir de un nodo cerrado; y la segunda que fuese la distancia Manhattan desde el nodo hasta el objetivo. Nótese que aquellas casilla que se considerasen un obstáculo se les asignaría un valor heurístico muy alto para descartarlas.

-Para empezar, mandaríamos la casilla origen a un vector de nodos cerrados, e introduciriamos las casillas colindantes a un vector de nodos cerrados, siendo estas nodos hijo de la casilla origen. Seleccionariamos la casilla con mejor valor heurístico al vector de nodos cerrados y la eliminaríamos de la de abiertos (en el caso de empate entre varias casilla, el algoritmo introduciría todas en este vector). Repetiriamos el proceso con la(s) casilla(s) que se encuentre(n) en el vector de nodos cerrados. Una vez alcanzasemos el nodo objetivo (rey o regalo), comenzaría una llamada hacia atrás de los nodos que han llegado a una solución, creando un camino válido entre el origen y el objetivo.

-Finalmente, se transformaría el camino a una planificación de movimientos que aplicará el personaje, dejando de realizar un comportamiento reactivo y pasando a realizar uno deliberativo. Si durante este trayecto se nos cruzase un aldeano o un lobo en nuestro camino (sensor de colision = true), simplemente deberíamos dejar de ejecutar el plan, retroceder un paso en este y recalcular una ruta hasta nuestro objetivo.

2. Variables utilizadas

Se han añadido las siguientes variable:

int objetos[5]
-Uso: contiene 1 si el objeto asociado a la posición *i* está en posesión del jugador, ya sea en su mano o en la mochila, y 0 si no está.

3. Funciones implementadas

Se han **añadido** las siguientes funciones:

void piensaObjetos(Sensores sensores, Action &accion)
-Uso: la función comprueba si el personaje ya posee el objeto que se encuentra enfrente del este. En caso afirmativo, si no tiene ningún objeto activo, lo recoge y si tiene alguún objeto activo, guarda el objeto actual en la mochila para recoger el nuevo en la siguiente iteración. En caso negativo, si el objeto equipado es el mismo que el que tiene delante, tira el actual, para así recoger en la siguiente iteración este. En otro caso, llama a la función *rotaObjetos* hasta encontrar el objeto idéntico al que tiene en la casilla de delante.

bool tieneObjeto(int i)
-Uso: devuelve *true* si el jugador posee en su mano o en la mochila el objeto *i.*

char tipoObstaculo(Sensores sensores)
-Uso: devuelve el una valor de tipo *char* con el valor de la casilla de justo enfrente del jugador cuando tiene un obstáculo. En caso de que no tenga ningún obstáculo, devuelve el valor ‘ ‘.

void rotaObjetos(Sensores sensores, Action &accion)
-Uso: modifica la variable *accion* para cambiar de objeto entre el que tiene el jugador en la mano (si tiene) por el primero objeto de la mochila.

void casillaSegura(Sensores sensores)
-Uso: devuelve *true* si la casilla sobre la que se encuentra el jugador no es ni agua ni bosque, y *false* en caso contrario.

void autofillAux(int a, int b, int c, int d)

-Uso: realiza la función que realizaba la funcióna *autofill* en la versión anterior, pero desde la casilla (a,b) hasta la (c,d).

Se han **modificado** las siguientes funciones:

void autofill()

-Uso: divide la *matrizResultado* en “cuadrantes” de menor tamaño y aplica la función *autofillAux a cada uno de ellos.*

void interacciones(Sensores sensores, Action &accion)

-Uso*:* respecto a la versión anterior, se ha modificado el comportamiento de la parte asociada al uso de objetos, y ahora llama a la función *piensaObjeto.*

**Nota:* Se han realizado pequeños cambios en algunas otras funciones pero no son suficientemente relevantes como para comentarlos, puesto que simplemente incluyen los regalos y reyes en las dinámicas establecidad anteriormente.