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” ha sido implementar el método “Pulgarcito”.

El programa comienza actualizando la orientación y posición del personaje. En el caso de haber muerto en la ejecución anterior, resetea las variables relativas y en el caso de no estar situado y encontrarse encima de un “punto GPS”, actualiza la posición absoluta y adapta el mapeado absoluto en función de los datos relativos que ha recogido previamente.

A continuación actualiza tanto el mapa resultado como la matriz pulgarcito en función de si está orientado o no; actualizando las matrices auxiliares correspondientes (*mapaResultadoAux y pulgarcitoAux*) si no está orientado o las matrices absolutas (*mapaResultado y pulgarcito*) si se encuentra localizado.

Después llama a las funciones que deciden cual será la siguiente acción a realizar, primero llamando a una función general que decide en función de los valores de la matriz pulgarcito y a continuación llama a una función más específica que decide dependiendo del elemento que se encuentre delante del personaje.

Finalmente, si la acción escogida es “avanzar”, y el personaje se encuentra rodeado por obstáculos, sustituiremos esta por la acción “girar a la izquierda”.

Para acabar, actualiza el valor de la posición actual de la matriz *pulgarcito* y el número de ejecuciones . En el caso de que sea la última ejecución, llama a una función que se ocupa de rellenar las casillas marcadas con un ‘?’.

2. Variables utilizadas

int orientacion
-Uso: representa la orientación hacia la que está mirando el personaje. Toma los siguientes valores: N=0,E=1,S=2,O=3.

bool perdido
-Uso: indica si el personaje ha encontrado o no un “punto de GPS”.

int pulgarcito[100][100]
-Uso: se utiliza para saber por las casillas por las que ya ha pasado el personaje, dandoles un valor mayor cada vez que el personaje visita la casilla.

int pulgarcitoAux[200][200]
-Uso: se utiliza para lo mismo que la variable anterior, pero mientras se usa únicamente cuando el personaje está perdido.

char mapaResultadoAux[200][200]
-Uso: guarda el terreno visto mientras el personaje está perdido.

int ejecuciones
-Uso: cuenta el número de ejecuciones que ha realizado el programa.

int posX
-Uso: representa la posición relativa en el eje X.

int posY
-Uso: representala posición relativa en el eje Y.

int posRX
-Uso: representa la posición absoluta en el eje X.

int posRY
-Uso: representa la posición absoluta en el eje Y.

int objetosMochila
-Uso: cuenta el número de objetos dentro de la mochila.

Action lastAction
-*Uso*: contiene la útlima acción realizada.

3. Funciones implementadas

void gira(int i)

-Uso: modifica la variable *orientacion* sumandole el valor *i* que se le pasa a esta. Finalmente, le aplica módulo 4 a esta variable para dejarla dentro de los valores del 0 al 3. Notar que le pasaremos el valor 1 cuando queramos girar a la derecha y el valor 3 cuando queramos girar a la izquierda.

void autofill()

-Uso: rellena los valores del *mapaResultado* en el caso de que sean *'?'*. La función recorre esta matriz, contando cuantas casillas se encuentras rellenas con agua, bosque, suelo y tierra, y rellena las casillas vacías con el elemento que más se repita en el mapa visualizado.

int minimo(int x,int y,int z)

-Uso: calcula el mínimo entre las tres variables que se le pasan. Devuelve este mínimo.

void resetear()

-Uso: reinicia a su valor predeterminado las variables temporales utilizadas hasta el momento.

void decidirDestino(Action &accion)

-Uso: modifica el valor de la variable que se le pasa, decidiendo si la siguiente acción debe ser *actTURN_L*, *actTURN_R* o *actFORWARD* en función del mínimo de los valores de las casillas que rodean al personaje de la matriz *pulgarcito*. Distingue los valores que compara en función de la orientación del personaje.

bool hayObstaculo(Sensores sensores,int i)

-Uso: determina si en la casilla de visión *i hay obstáculos. Para ellos, utiliza los sensores de visión del personaje. Toma como obstáculos tanto los precipicios como los muros siempre, y el bosque, el agua y las puertas solo las toma como obstáculos en el caso de que el personaje no tenga el objeto necesario equipado para atravesarlas.*

void actualizarPulgarcito(Sensores sensores)

-Uso: en el caso de no haber encontrado un "punto GPS", actualiza los valores de la matriz *pulgarcitoAux* en función de si lo que está viendo en su campo de visión es un obstáculo o no. Si es un obstáculo, le añade a esa casilla un valor de *1000*, si no es un obstáculo le aplica el módulo *1000* para que, en caso de que anteriormente fuese un obstáculo pero se cumplan los requisitos para que deje de serlo, no suponga un problema haberlo visto con anterioridad y haberle asignado un valor muy alto y finalmente, en el caso de que la casilla sea un "punto GPS", le da el valor *-1* para darle preferencia respecto al resto.

En el caso de encontrarse localizado, se siguen aplicando las dos primeras normas de las anteriormente nombradas, pero sobre la matriz *pulgarcito*.

Nótese que se ha utilizado un algoritmo que permite encapsular el recorrido del vector que almacena los sensores de visión en dos bucles for, y que ya varía en qué posición guarda estos valores en función de la orientación de nuestro personaje.

void actualizarMapaResultado(Sensores sensores)

-Uso: en el caso de no haber encontrado un "punto GPS", actualiza los valores de la matriz *mapaResultadoAux* en función de los sensores de visión y en el caso de si haberlo encontrado, actualiza los valores de la matriz *mapaResultado.*

Nótese que se ha utilizado un "algoritmo" que permite encapsular el recorrido del vector que almacena los sensores de visión en dos bucles for, y que ya varía en qué posición guarda estos valores en función de la orientación de nuestro personaje.

void orientar(Action accion,bool colision)

-Uso: la función se encarga de actualizar la posición del personaje y su orientación.

Para lo primero, comprueba si la última acción ha sido *actFORWARD* y si no ha habido colisión en la ejecución anterior y actualiza la posición en función de la orientación del personaje. Para lo segundo,llama a la función auxiliar *gira(int i)* en el caso de que la última acción haya sido *actTURN_L* o *actTURN_R*.

void interacciones(Sensores sensores, Action &accion)

-Uso: decide que acción realizar en función del objeto/ente que se encuentra en la casilla enfrente de nuestro personaje que acción tomar.

En el caso de tener un objeto delante decide si:

-*actTHROW*: en el caso de tener la mochila llena y de tener un objeto equipado.

-*actPICKUP*: en el caso de no tener un objeto equipado.

-*actPUSH*: en otro caso, es decir, si tiene un objeto en la mano, y no tiene la mochila llena.

En el caso de tener un lobo delante decide si:

-*actGIVE*: si el objeto equipado es un hueso.

-*actIDLE*: en otro caso.

En el caso de tener una puerta delante decide si:

-*actGIVE*: si el objeto equipado es una llave.

-*actTURN_L*: en otro caso.

En el caso de tener un aldeano delante hace actIDLE.

En el caso de no haber ningún objeto/ente, no modifica la variable *accion*.

void ajustar(Sensores sensores)

-Uso: actualiza el valor de las posiciones absolutas de las matrices *mapaResultado* y *pulgarcito* en función de las posiciones relativas de las matrices *mapaResultadoAux* y *pulgarcitoAux.*