Práctica 2 Inteligencia Artificial
Agentes Deliberativos

Javier Martín Gómez
2ºC

En esta práctica ha habido una mejora con respecto al Agente Reactivo, ya que en esta he podido implementar el mapaPulgarcito con lo que ayuda mucho a recorrer el mapa y una gestión más eficiente de los objetos.
El resto de código ha sido mejorado, ultilizando más métodos que antes.
Y por último el algoritmo usado es el Algoritmo A* que explicaré más adelante.
Los comportamientos del agente son los siguientes

1. Brújula

La brújula nos indica a dónde estamos orientados en cada momento, este trozo de código nos lo daban en el tutorial, está dentro del método think.

2. MapaPulgarcito

Esta nueva implementación debí hacerla en la práctica 1, pero como no supe y no me dio tiempo no pude implementarla.
Gracias a la ayuda de mis compañeros he podido implementarla y aumentar considerablemente el mapa descubierto.

Simplemente voy añadiendo a una nueva matriz, denominada MapaPulgarcito un número en cada celda conforme vaya pasando (dicho número aumenta de uno en uno) Mi personaje decide entonces pasar por el número menor que tenga en el eje cartesiano, consiguiendo así que recorra todo el mapa.

3. Comportamiento ante la naturaleza (bosque, agua, precipicio...)

Mi agente reactivo toma decisiones simples ante los elementos de la naturaleza que tiene delante.

Si tiene bosque delante se adentra en él si tiene el objeto 'zapatillas' equipadas

Si es agua lo que tiene delante realiza el mismo procedimiento que el bosque pero usando el objeto 'biquini'.

En caso de tener una puerta delante también realiza el mismo procedimiento anterior pero con el objeto 'llaves'.

En caso de que el objeto que necesite esté en la mochila, me lo equipo

Todo esto se hace mientras se completa el mapaPulgarcito

4. Comportamiento ante objetos

Aquí ha habido una mejora con respecto a la práctica anterior ya que además de coger objetos, los suelto antes de morir para no perderlos objetos.
En caso que ya tengamos un objeto no cogemos otro para no tener repetidos.

5. Comportamiento ante NPCs

Hay 3 NPCs en el juego, aldeanos, lobos y reyes.

Si hay un aldeano mi agente no hace nada.
Si hay un lobo le da el hueso en caso que lo tenga en la mano o en la mochila con la acción actGIVE
Con el rey solo interactúo en caso de tener un regalo en la mano que en ese caso se lo damos con actGIVE y nos aumenta el número de misiones completadas

6. Dibujar mapa según lo que vean los sensores

En un principio mi agente dibujaba lo que tenía bajo sus pies pero ahora dibuja todo lo que vea los sensores.
Cuando el agente alcanza una casilla 'K', el agente reconoce en qué posición del mapa está y empieza a dibujarla.
Con un switch-case de la brújula, dependiendo de la orientación puedo dibujar todo el mapa que rastrea los sensores consiguiendo así un porcentaje de mapa amplio.

7. Mapa auxiliar, volcar el mapa auxiliar en el mapa resultante

Se desperdicia mucho mapa y tiempo antes de encontrar una casilla 'K' por lo que he creado una matriz auxiliar del doble de tamaño que la original.
En esta matriz he ido desde el primer momento metiendo lo que los sensores han rastreado y estoy en una casilla 'K' vuelco todo lo que he encontrado en la matriz auxiliar en la matriz resultante consiguiendo así descubrir mucho más mapa y desperdiciando el menor tiempo posible.

8. Reinicio

Cuando el personaje se queda sin vida un sensor llamado reset se pone a true. Se pone las variables al valor que tiene el constructor en el .hpp Para poder así comenzar de nuevo y seguir recorriendo el mapa con normalidad.

9.Algoritmo de búsqueda. A*

El algoritmo de búsqueda que he usado es el A* ya que es el más óptimo si encuentro el h(n) admisible.
Por desgracia, aun teniendo las ideas claras del algoritmo e intentar plasmarlo en código, no funciona como yo esperaba. Igualmente lo voy a explicar.

```
//Aquí creo las 2 listas de nodos, el abierto y cerrado, además de 2 iteradores
//para avanzar en los nodos

list<nodo> nodos_abiertos, nodos_cerrados;
list<nodo>::iterator it, mejor_nodo;

//asigno al struct estado la posición origen
estado st = origen;

//calculo la distancia Manhattan entre el origen y el destino
int distManh = distanciaManhattan(origen,destino);

//inicializo el primer nodo de todos
nodo inicial(origen.fila, origen.columna, NULL, distManh);

//Ahora mientras haya nodos abiertos (con while(!nodos_abietos.empty()))

//itero sobre los nodos abiertos y lo comparo con el mejor nodo, si tiene
//menor distancia manhattan, lo asigno al numero mejor_nodo
mejor_nodo = nodos_abiertos.begin();
        for(it = nodos_abiertos.begin(); it != nodos_abiertos.end(); it++){
                if((*it).distancia < (*mejor_nodo).distancia){
                        mejor_nodo = it;
                }
        }

//elimino el nodo padre de los nodos abiertos y lo pongo en el cerrado
nodos_abiertos.erase(mejor_nodo);
nodo nodoActual = *mejor_nodo;
nodos_cerrados.push_back(nodoActual);

//En la línea 73 a la 82 si el nodo actual es igual al destino significa que ya
//hemos
//encontrado el camino. Itero sobre los nodos_cerrados, buscando el padre
de    //cada uno y poniendolos en una lista nueva, para eliminar así los
//nodos_cerrados que no nos sirven
```

//Desde la línea 88 a la 125 voy comparando la orientación que tengo con
//respecto al nodo al que debo ir, para poder así situarme correctamente y
//avanzar

//Desde la línea 138 a la 170 va buscando los hijos posibles del padre, en caso
//de tener una distancia Manhattan satisfactoria lo añade como nuevo nodo,
//así hasta llegar al destino

10. Conclusión

Como se puede ver es un algoritmo sencillo de comprender, pero de implementar ha sido un poco complicado. Aunque no funcione es satisfactorio haberlo intentado junto a más compañeros ayudándonos unos a otros.
Aunque no funcione del todo bien el algoritmo, he conseguido mejorar el descubrimiento del mapa gracias al mapaPulgarcito considerablemente y en menor medida gracias a la mejor gestión de objetos,
Ha sido una práctica un poco complicada pero entretenida.