Práctica 2 IA

Víctor Manuel Arroyo Martín

Nivel 1

Función think:

Para recorrer el plan había que ir cambiando la brújula en cada giro. Para ello, he hecho un switch con la acción del plan que se iba a tomar y he cambiado la brújula convenientemente.

```
switch (accion){
    case actTURN_R: brujula= (brujula+1)%4; break;
    case actTURN_L: brujula= (brujula+3)%4;  break;
}
```

Si no hubiera plan, he implementado unas sentencias con las que el jugador se moverá siempre hacia delante hasta que encuentre un obstáculo en cuyo caso girará hacia la derecha.

```
if(!hay_plan){
    if(sensores.terreno[2]=='P' or sensores.terreno[2]=='M' or
        sensores.terreno[2]=='D' or sensores.superficie[2]=='a'){
        accion=actTURN_R;
    }else{
        accion=actFORWARD;
    }
}
```

Búsqueda en anchura:

Para ello, he creado una función recursiva (anchura_Recursivo) que para cada hijo del nodo que se le pasa por parámetro crea nuevos nodos a los que más tarde se les aplicará la recursividad añadiéndolo a la lista de hijos del nodo si no se ha recorrido. Así se va explorando y generando el árbol nivel a nivel hasta que se encuentra el destino, momento en el que la función devuelve *true* y se detiene la recursividad. Por otro lado, está el pathFinding_Anchura que es la función principalque luego llama a la recursiva.

Búsqueda de costo mínimo

Para implementarla, lo primero ha sido crear un nuevo struct en jugador hpp con un nodo y el coste del camino para llegar a él.

```
struct costo{
  nodo n;
  int valor;
};
```

También ha hecho falta una nueva función encuentra_Minimo que de una lista de costo te cambia el iterador que se le pasa por entrada apuntando al costo que tiene el menor valor de todos.

```
void ComportamientoJugador::encuentra_Minimo(list<costo> &costos, list<costo>::iterator &min){
    list<costo>::iterator it;
    int minimo=3000;

    for(it=costos.begin(); it!=costos.end(); it++){
        if((*it).valor<minimo){
            minimo=(*it).valor;
            min=it;
        }
    }
}
```

Así hice la función principal de forma que recoja cada nodo no visitado de una lista de nodos hijos de forma que siempre coja el de menor coste hasta llegar al destino, como se explicó en clase. La función se llama pathFinding_Costo.

Nivel 2

Para la función think he implementado un comportamiento parecido al principio al del nivel uno: si no puede avanzar, va a la derecha. Cuando llega a un punto de referencia, lo primero que hace es cambiar las variables de estado como la fila, columna y orientación. Tras ello, guarda en el mapaResultado todo lo que sus sensores le detectan y lo introduce en las filas y columnas correspondientes. Por último, el bool “llega” definido en el hpp cambia a true y la acción a tomar es estarse quieto.
En la siguiente acción ya se hará un plan para llegar al destino. Se usará el de costo mínimo suponiendo que la parte no conocida cuesta 1 y como se va a ir replanificando a cada paso, el plan cambiará según vaya descubriendo terreno. También hay que ver si el destino se ha actualizado a otro lugar por haber llegado a él, así como ir actualizando el estado y posición del jugador en cada paso.

Si se encuentra un aldeano u obstáculo en su paso, girará hacia la derecha y replanificará una vez ahí.
En cada paso se va guardando el mapa que vamos descubriendo en mapaResultado para que se vaya pintando y se vaya actualizando el plan.
Esto se va guardando con un switch que, según la orientación del jugador en ese momento, guarda en las filas y columnas correspondientes lo que llega de los sensores.

```
switch(brujula){
    case 1:
        mapaResultado[fil][col] = sensores.terreno[0];
        mapaResultado[fil-1][col+1] = sensores.terreno[1];
        mapaResultado[fil][col+1] = sensores.terreno[2];
        mapaResultado[fil+1][col+1] = sensores.terreno[3];
        mapaResultado[fil-2][col+2] = sensores.terreno[4];
        mapaResultado[fil-1][col+2] = sensores.terreno[5];
        mapaResultado[fil][col+2] = sensores.terreno[6];
        mapaResultado[fil+1][col+2] = sensores.terreno[7];
        mapaResultado[fil+2][col+2] = sensores.terreno[8];
        mapaResultado[fil-3][col+3] = sensores.terreno[9];
        mapaResultado[fil-2][col+3] = sensores.terreno[10];
        mapaResultado[fil-1][col+3] = sensores.terreno[11];
        mapaResultado[fil][col+3] = sensores.terreno[12];
        mapaResultado[fil+1][col+3] = sensores.terreno[13];
        mapaResultado[fil+2][col+3] = sensores.terreno[14];
        mapaResultado[fil+3][col+3] = sensores.terreno[15];
  break;

  case 3:
        mapaResultado[fil][col] = sensores.terreno[0];
        mapaResultado[fil+1][col-1] = sensores.terreno[1];
        mapaResultado[fil][col-1] = sensores.terreno[2];
        mapaResultado[fil-1][col-1] = sensores.terreno[3];
        mapaResultado[fil+2][col-2] = sensores.terreno[4];
        mapaResultado[fil+1][col-2] = sensores.terreno[5];
        mapaResultado[fil][col-2] = sensores.terreno[6];
        mapaResultado[fil-1][col-2] = sensores.terreno[7];
        mapaResultado[fil-2][col-2] = sensores.terreno[8];
        mapaResultado[fil+3][col-3] = sensores.terreno[9];
        mapaResultado[fil+2][col-3] = sensores.terreno[10];
        mapaResultado[fil+1][col-3] = sensores.terreno[11];
        mapaResultado[fil][col-3] = sensores.terreno[12];
        mapaResultado[fil-1][col-3] = sensores.terreno[13];
        mapaResultado[fil-2][col-3] = sensores.terreno[14];
        mapaResultado[fil-3][col-3] = sensores.terreno[15];
    break;
```

```
case 0:
    mapaResultado[fil][col] = sensores.terreno[0];
    mapaResultado[fil-1][col-1] = sensores.terreno[1];
    mapaResultado[fil-1][col] = sensores.terreno[2];
    mapaResultado[fil-1][col+1] = sensores.terreno[3];
    mapaResultado[fil-2][col-2] = sensores.terreno[4];
    mapaResultado[fil-2][col-1] = sensores.terreno[5];
    mapaResultado[fil-2][col] = sensores.terreno[6];
    mapaResultado[fil-2][col+1] = sensores.terreno[7];
    mapaResultado[fil-2][col+2] = sensores.terreno[8];
    mapaResultado[fil-3][col-3] = sensores.terreno[9];
    mapaResultado[fil-3][col-2] = sensores.terreno[10];
    mapaResultado[fil-3][col-1] = sensores.terreno[11];
    mapaResultado[fil-3][col] = sensores.terreno[12];
    mapaResultado[fil-3][col+1] = sensores.terreno[13];
    mapaResultado[fil-3][col+2] = sensores.terreno[14];
    mapaResultado[fil-3][col+3] = sensores.terreno[15];
  break;

  case 2:
    mapaResultado[fil][col] = sensores.terreno[0];
    mapaResultado[fil+1][col+1] = sensores.terreno[1];
    mapaResultado[fil+1][col] = sensores.terreno[2];
    mapaResultado[fil+1][col-1] = sensores.terreno[3];
    mapaResultado[fil+2][col+2] = sensores.terreno[4];
    mapaResultado[fil+2][col+1] = sensores.terreno[5];
    mapaResultado[fil+2][col] = sensores.terreno[6];
    mapaResultado[fil+2][col-1] = sensores.terreno[7];
    mapaResultado[fil+2][col-2] = sensores.terreno[8];
    mapaResultado[fil+3][col+3] = sensores.terreno[9];
    mapaResultado[fil+3][col+2] = sensores.terreno[10];
    mapaResultado[fil+3][col+1] = sensores.terreno[11];
    mapaResultado[fil+3][col] = sensores.terreno[12];
    mapaResultado[fil+3][col-1] = sensores.terreno[13];
    mapaResultado[fil+3][col-2] = sensores.terreno[14];
    mapaResultado[fil+3][col-3] = sensores.terreno[15];
  break;

}
```