INTELIGENCIA ARTIFICIAL
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

PRÁCTICA 2

Sofía Fernández Moreno
29 de abril de 2018

1. Algoritmo de búsqueda

El algoritmo utilizado para desarrollar el plan a realizar ha sido el A*, donde se ha usado un struct llamado Movimiento para guardar la información respecto a cada uno de los costes a generar y una lista "plan" donde guardará los movimientos y se obtendrá el más interesante. Además de un estado que guardará la posición actual del nodo.

Para este algoritmo hemos usado para:

- ABIERTOS una estructura priority_queue de los movimientos guardados.
- CERRADOS una estructura set con los estados de cada nodo es decir guardará la fila,la columna y la orientación de cada posición.

ABIERTOS inicialmente tendrá el nodo inicial, es decir, el origen, mientras que CERRADOS se encontrará vacío.
Mediante un while realizaremos el ciclo para encontrar la solución o hasta que ABIERTOS no contenga nada, es decir, se encuentre vacío.

En nuestro struct Movimiento tendremos tres costes:

- H: guardará la heurística usada
- G: coste hasta el nodo n
- F: suma de H y G

Para G lo que llamamos coste sería la cantidad de acciones que hemos tenido que realizar, por lo que utilizaremos plan.size().
Para el calculo de H, aplicaremos la heurística de manhattan que básicamente será el calculo de distancias desde un punto hacia otro.

Si nuestra heuristica(h(n)) es admisible, A* encontrará el óptimo.
Para elegir el siguiente nodo miraremos el coste y además el que tenga mejor h(n).

Obtendremos el mejor nodo mediante la comprobación de si podemos avanzar, girar a la izquierda o girar a la derecha, dependiendo el estado que tengamos.
Para avanzar lo único que debemos hacer es comprobar que podemos avanzar es decir que no nos encontremos con muros, puertas, agua o bosque. Una vez esto podremos avanzar hacia delante y comprobar si llegamos a nuestro destino.
Todo esto comprobando que el estado guardado en ABIERTOS se encuentra en CERRADOS. Además de esto se calculará los anteriores costes mencionados.
Para los giros izquierda y giros derecha será exactamente igual.
Una vez hecho lo anterior cogeremos el que mejor costeF tenga, es decir, el que su F sea el de menor valor.
Una vez hecho esto lo introduciremos en CERRADOS una vez introducido en ABIERTOS y comprobado que se encuentra.

2. Funciones Auxiliares

Para la realización del plan hemos tenido que utilizar varias funciones auxiliares.

```
estado ComportamientoJugador::avanza(estado pos){
    estado sig;
    if(pos.orientacion==0){ //Norte
        sig.fila=pos.fila - 1;
        sig.columna=pos.columna;

    }
    else if(pos.orientacion==1){ //Este
        sig.fila=pos.fila;
        sig.columna=pos.columna +1;
    }
    else if(pos.orientacion==2){    //sur
        sig.fila=pos.fila+1;
        sig.columna=pos.columna;
    }
    else if(pos.orientacion==3){    //Oeste
        sig.fila=pos.fila;
        sig.columna=pos.columna-1;

    }
    sig.orientacion=pos.orientacion; ///la orientacion no cambia
    return sig;

}

estado ComportamientoJugador::girarIzq(estado pos){
    estado posfutura = pos;
    posfutura.orientacion =(posfutura.orientacion + 3)%4;
    posfutura.fila = pos.fila;
    posfutura.columna = pos.columna;
    return posfutura;
}
estado ComportamientoJugador::girarDch(estado pos){
    estado posfutura = pos;
    posfutura.orientacion =(posfutura.orientacion + 1)%4;
    posfutura.fila = pos.fila;
    posfutura.columna = pos.columna;
    return posfutura;
}
```

```
//Funcion para calcular la heuristica de manhattan
int ComportamientoJugador::heuristica_manhattan(estado posInicio, estado
posDestino){
    int fil_posInicio=posInicio.fila;
    int col_posInicio=posInicio.columna;

    int fil_posDestino=posDestino.fila;
    int col_posDestino=posDestino.columna;

    int distanciafilas=abs(fil_posDestino - fil_posInicio);
    int distanciacolumnas=abs(col_posDestino - col_posInicio);


    return distanciafilas+distanciacolumnas;

}
bool ComportamientoJugador::puedoAvanzar(char valor){
    return (valor == 'S' || valor == 'T' || valor == 'K' );

}
```

Donde avanza, girarIzq, girarDch se dedican básicamente a calcular la orientación de la brújula y guardarlo en nuestro estado de movimientos.
La función heurística_manhattan se encarga de resolver la heurística para dos posiciones concretas.
La función puedeAvanzar se encargar de comprobar para un valor char que coincide con los terrenos ahí mencionados.

3. Operadores de comparación

Además de las funciones auxiliares creadas he usado estos dos operadores:

```
bool operator<(const Movimiento &m1,const Movimiento &m2){
//Para comparar el de menor costeF
    return m1.f>m2.f;
}
  bool operator<(const estado &e1, const estado &e2){
     if(e1.fila < e2.fila) return true;
     if(e1.fila == e2.fila && e1.columna < e2.columna) return true;
     if(e1.fila == e2.fila && e1.columna == e2.columna && e1.orientacion <
e2.orientacion) return true;
     return false;

}
```

Los cuales me han servidor para comparar los costes F y la limitación de las filas y las columnas.

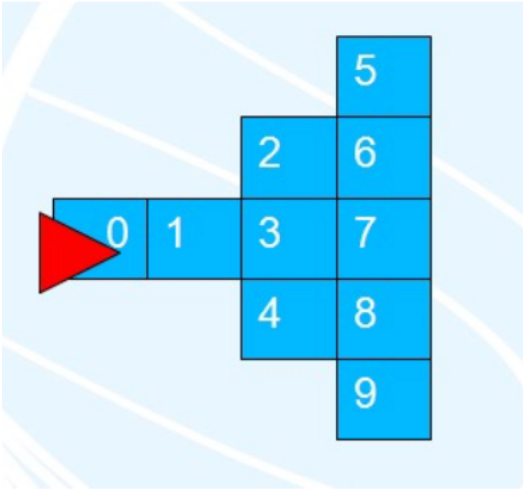
4. **Estrategia para el nivel 2(reconocimiento de aldeanos)**

La estrategia que he usado ha sido para cuando me encuentre con un aldeano, esta comprobación la realizo mediante sensores.superficie[2] == 'a', me detengo, es decir, no realizo ningún movimiento y cuando no lo tenga enfrente de mí me vuelvo a mover.

5. **Nivel 3**

Para la realización del nivel 3 he planteado varias funciones auxiliares para realizar la actualización del mapa y su pintado (el nuevo mapa).
He seguido la estrategia de pulgarcito, la cual va poniendo en la posición actual del mapa pulgarcito un numero que se va incrementando cada vez que pone una casilla.
Para el pintado del mapa, he utilizado los sensores de terreno y una vez descubierto el PK lo dibuja.

En cuanto a la búsqueda del PK he aplicado la regla de que dependiendo la posición del vector terreno realizará una acción que será incluida en nuestro plan.



Para la exploración he seguido la línea de que una vez nuestro personaje no tiene información intenta sacar mediante las veces que pasa por un lugar. Se irá obteniendo el valor mínimo comparando mediante la orientación y el mapa de pulgarcito(mapaPrioridades).
Una vez realizado esto se pasará a reorientar nuestro mapa.

6. **Problemas en la práctica**

A la hora de realizar el plan he tenido varios problemas con el control de las filas y las columnas, por eso la utilización del operator< para "estado".
Además de un problema en la no realización del plan en algunas circunstancias pero realizando "make clean" se solucionaba.