Stalyn Alejandro Alcocer Valencia DNI: 20947870E

Documentación PacMan

15.06.2020

Implementación Algoritmo A*

El objetivo es encontrar un camino o ruta entre dos puntos, y este camino debe ser el de menor coste. Es decir que dados dos puntos debemos encontrar el camino más corto. Para este trabajo utilizamos el algoritmo A* que tiene la peculiaridad de utilizar una función heurística. La función heurística es una estimación optimista de cómo de lejos está el objetivo.

En este caso vamos a tener dos casos:s

1. Inicio = posición Fantasma y Objetivo = posición Pacman.
2. Inicio = posición Pacman y Objetivo = contrario a posición Fantasma.

El primer caso es fácil de ver : es cuando un fantasma busca a Pacman y elige el camino más corto hacia él. A continuación se muestra el código del algoritmo A* para el Fantasma.

En esta imagen se puede ver lo que sería la inicialización de los parámetros:

```
//////////////////////////////
// A ESTRELLA PARA FANTASMA //
//////////////////////////////
int AestrellaFantasma(Laberinto laberinto){
    int result= -1; //Devuelve el movimiento a realizar
    boolean encontrado=false;

    inic(laberinto.tam());

    //Inicializo las dos listas que voy a utilizar.
    ArrayList<Nodo> listaInterior = new ArrayList<>();
    ArrayList<Nodo> listaFrontera = new ArrayList<>();

    //Guardo la posición de Inicio y el objetivo(Pacman)
    int[] posFantasma = laberinto.obtenerPosicionFantasma(numeroFantasma);
    int[] posPacman = laberinto.obtenerPosicionPacman();

    Nodo nodoInicial = new Nodo(null, posFantasma[0], posFantasma[1]);
    Nodo nodoFinal = new Nodo(null,posPacman[0], posPacman[1]);

    //Inicializo las listas. ListaInterior vacia y listaFrontera con el nodoInicial.
    listaInterior.clear();
    listaFrontera.add(nodoInicial);

    int cont = 0;
    expandidos = 0;
```

IMAGEN 1

En esta segunda imagen vemos el bucle del algoritmo, se va a repetir hasta que la "listaFrontera" se quede vacía.

```
while(!listaFrontera.isEmpty()){

    //Elijo el nodo con f mas prometedor de listaFrontera
    int winner = 0;
    for(int i = 0; i < listaFrontera.size(); i++){
        if(listaFrontera.get(i).f < listaFrontera.get(winner).f){
            winner = i;
        }
    }

    Nodo nodo = listaFrontera.get(winner);
    nodo.addVecinos(laberinto);

    //Si el hemos llegado al objetivo. Recreo el camino y lo guardo en 'path'
    if(nodo.equals(nodoFinal)){
        ArrayList<Nodo> path = new ArrayList<>();
        path.add(nodo);
        coste_total = nodo.f;
        while(nodo.padre != null){

            camino[nodo.y][nodo.x] = 'X';
            path.add(nodo.padre);
            nodo = nodo.padre;
        }
        camino[nodo.y][nodo.x] = 'X';
        encontrado = true;
        //Busco el movimiento que tiene que hacer.
        //System.out.println("Result: " + result);
        result = moveTo(path);
        break;
    }

    camino_expandido[nodo.y][nodo.x] = cont++;
    expandidos++;
    listaInterior.add(nodo);
    //removeNodoFromFrontera(nodo.x, nodo.y, listaFrontera);
    listaFrontera.remove(nodo);
```

IMAGEN 2

En esta primera parte del bucle, hemos conseguido dos cosas importantes: la primera es escoger un nodo prometedor según las casillas de nuestro alrededor y segundo, comprobar si este nodo prometedor nos lleva a una solución aceptable. Es decir si nos lleva hasta PacMan

Vamos a ver la última parte del bucle del algoritmo A*

```
        //Analizo los nodos vecinos, por si son más prometedores.
        for(int i = 0; i < nodo.vecinos.size(); i++){
            Nodo vecino = nodo.vecinos.get(i);

            if(!containsPosition(vecino.x, vecino.y, listaInterior)){
                double auxG = vecino.g + heuristicaEuclidea(vecino, nodo);
                //System.out.println("auxG: " + auxG + ", vecino.g: " + vecino.g + ", " + nodo.g);
                if(!containsPosition(vecino.x, vecino.y, listaFrontera)){

                 vecino.g = nodo.g + 1;
                 vecino.h = heuristicaEuclidea(nodo, nodoFinal);
                 vecino.f = vecino.g + vecino.h;
                 vecino.padre = nodo;
                 listaFrontera.add(vecino);

                }else if(auxG <= nodo.g){
                 vecino.padre = nodo;
                 vecino.g = auxG;
                 vecino.h = heuristicaEuclidea(vecino, nodoFinal);
                 vecino.f = vecino.g + vecino.h;
                }
            }
        }
    }
```

IMAGEN 3

En la última parte del bucle, lo que hacemos es comprobar las demás casillas de alrededor para ver si una de ellas es más prometedora que la casilla que analizamos en la primera parte del bucle (Imagen 2). En caso de ser más prometedora elegiremos esta casilla para seguir buscando nuestro camino.

Así finaliza el bucle y el Algoritmo A* implementado para los fantasmas.

La implementación del algoritmo A* para PacMan, es básicamente la misma, sólo que hay que invertir los puntos de inicio y objetivo. En este caso el inicio es la posición que ocupa PacMan y el objetivo es moverse contrariamente al fantasma más próximo.

Por eso voy a enseñar las partes del código que cambian del algoritmo A* para los fantasmas.

```
//Inicializo las dos listas que voy a utilizar.
ArrayList<Nodo> listaInterior = new ArrayList<>();
ArrayList<Nodo> listaFrontera = new ArrayList<>();

//Guardo la posición de Inicio y el objetivo(Pacman)
int[] posPacman = laberinto.obtenerPosicionPacman();
Nodo nodoInicial = new Nodo(null, posPacman[0], posPacman[1]);

//Calculo que fantasma está más cerca. Para alejarme de él.
Nodo nodoFinal = null;
double distMin = 1000;
for(int i = 1 ; i <= laberinto.numFantasmas ; i++){
    int[] posFantasma = laberinto.obtenerPosicionFantasma(i);
    if( nodoInicial.calcularDistancia(new Nodo(null, posFantasma[0], posFantasma[1])) < distMin){
        nodoFinal = new Nodo(null, posFantasma[0], posFantasma[1]);
        distMin = nodoInicial.calcularDistancia(new Nodo(null, posFantasma[0], posFantasma[1]));
    }
}
```

IMAGEN 4

Esta es la inicialización de los valores para el algoritmo A* para PacMan. Pero lo más destacable de esta parte es el bucle for que sirve para buscar de entre los fantasmas existentes cuál está más cerca. El fantasma escogido nos servirá de referencia para alejarnos de él en esa instancia. El resto de código para PacMan es idéntico que para Fantasma. Solo que la moverse se moverá para el lado contrario del la posición del fantasma, intentando alejarse de este. Esto se consigue con la siguiente función que hace las comprobaciones pertinentes.

```
private int moveToPacman(ArrayList<Nodo> path ,Laberinto lab){

    Nodo nodo = new Nodo(path.get(path.size() - 1));
    Nodo moveNodo = new Nodo(path.get(path.size() - 2));
    //System.out.println("..................................................Mover");

    int x = nodo.x - moveNodo.x;
    int y = nodo.y - moveNodo.y;


    if(x == 0 && y == 1){
        //Arriba.

        if(lab.obtenerPosicion(nodo.x, nodo.y+1) != 1){ //Abajo
            return Laberinto.ABAJO;
        }
        if(lab.obtenerPosicion(nodo.x+1, nodo.y) != 1){ //Derecha
            return Laberinto.DERECHA;
        }
        if(lab.obtenerPosicion(nodo.x-1, nodo.y) != 1){ //Izquierda
            return Laberinto.IZQUIERDA;
        }

        return Laberinto.ARRIBA;
    }
```

IMAGEN 5

Análisis Comparativo.

Vamos a analizar cómo se comporta el algoritmo según la heurística que utilicemos. Recordamos que cuanto mejor estimemos la función heurística menos nodos generamos, lo que se traduce en mejor eficiencia.

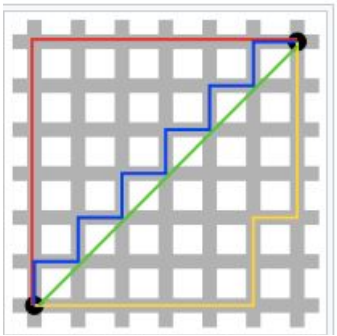
Vamos utilizar tres heurísticas:

```
private double heuristicaUno(){
    return 1;
}

private double heuristicaEuclidea(Nodo inicio, Nodo destino){
    //Euclidea
    double x = (double)Math.pow(destino.x - inicio.x, 2);
    double y = (double)Math.pow(destino.y - inicio.y, 2);
    return (double)Math.sqrt(x + y);
}

private double heuristicaManhattan(Nodo inicio, Nodo destino){
    //Manhattan
    return (double)(Math.abs(inicio.x - destino.x) + (Math.abs(inicio.y - destino.y)));
}
```

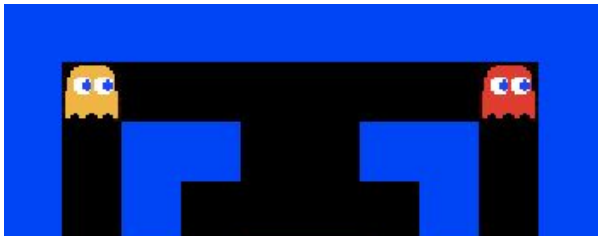
La primera “heuristicaUno” la utilizaremos para ver cómo funciona el algoritmo sin una función heurística. La segunda función “heuristicaEculidea” se basa en la fórmula típica para calcular la distancia entre dos puntos en un plano xy. Viene dada por la fórmula:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

La tercera y última función “heuristicaManhattan”, en principio es la que más se adapta a nuestro problema, pues sirve para calcular la distancia entre dos puntos, en un mapa de “bloques” (como se haría en una ciudad). De forma visual se vería de esta manera.



Para comparar vamos a utilizar estas tres heurísticas en un solo laberinto para ver los resultados.



El laberinto utilizado es “prueba.txt”. Que vemos al lado.

1. Prueba sin heurística.

```
NO MODIFICAR ESTE FORMATO DE SALIDA
Coste del camino: 10.0
Nodos expandidos: 33
Camino
. . . . . . . . . .
. X . . . . . . . .
. X . . . . . . . .
. X . . . . . . . .
. X X X . . . . . .
. . . X . . . . . .
. . . X X X . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
Camino explorado
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  0  1  3  5  7 11 17 22 -1
-1  2 -1 -1  8 12 -1 -1 26 -1
-1  4 -1 19 13 18 23 -1 30 -1
-1  6  9 14 -1 -1 27 31 -1 -1
-1 10 15 20 -1 -1 32 -1 -1 -1
-1 16 -1 24 28 -1 -1 -1 -1 -1
-1 21 -1 -1 -1 -1 -1 -1 -1 -1
-1 25 29 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Esta es una captura del fantasma 1 buscando el camino más corto hacia la posición de PacMan. Lo que podemos entender de no utilizar la función heurística, es la gran cantidad de nodos que se generan. Ya que no puede ver otros nodos más prometedores.

El no tener una función bien acotada nos conlleva realizar más cálculos innecesarios o que se podrían evitar.

2. Prueba con Heurística Euclídea.

```
NO MODIFICAR ESTE FORMATO DE SALIDA
Coste del camino: 10.0
Nodos expandidos: 22
Camino
. . . . . . . . . .
. X . . . . . . . .
```

Desde el primer movimiento vemos la mejora. El fantasma 1 encuentra el mismo resultado que antes, pero expandiendo 11 nodos menos. En total expande 22 nodos.

Es una mejora notable y que hace del problema más escalable para resolver problemas más grandes.

3. Prueba con Heurística Manhattan.

```
NO MODIFICAR ESTE FORMATO DE SALIDA
Coste del camino: 10.0
Nodos expandidos: 16
Camino
. . . . . .
. . . . . .
. X . . . . .
. X . . . . .
. X X X . . .
. . . X . . .
. . . X X X .
. . . . . .
. . . . . .
. . . . . .
Camino explorado
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1  2 -1 -1 -1 -1 -1 -1 -1 -1
-1  0 -1 -1 -1 -1 -1 -1 -1 -1
-1  1 -1 10 -1 -1 -1 -1 -1 -1
-1  3  4  6 -1 -1 -1 -1 -1 -1
-1  5  7  9 -1 -1 -1 -1 -1 -1
-1  8 -1 12 13 14 -1 -1 -1 -1
-1 11 -1 -1 15 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Con Manhattan por el momento vemos la mejor solución. Con solo 16 nodos expandidos para llegar a la solución.