A\*算法（路径）

import java.util.ArrayList;

import java.util.List;

public class AStar {

public static final int[][] maps = {

{0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0, 0},

};

public static final int[][] direct = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};

public static final int step = 10;

private ArrayList<Node> openList = new ArrayList<Node>();

private ArrayList<Node> closeList = new ArrayList<Node>();

public Node findMinFNodeInOpneList() {

Node tempNode = openList.get(0);

for (Node node : openList) {

if (node.F < tempNode.F) {

tempNode = node;

}

}

return tempNode;

}

public ArrayList<Node> findNeighborNodes(Node currentNode) {

ArrayList<Node> arrayList = new ArrayList<Node>();

for (int i = 0; i < 4; i++) {

int newX = currentNode.x + direct[i][0];

int newY = currentNode.y + direct[i][1];

if (canReach(newX, newY) && !exists(openList, newX, newY))

arrayList.add(new Node(newX, newY));

}

return arrayList;

}

public boolean canReach(int x, int y) {

if (x >= 0 && x < maps.length && y >= 0 && y < maps[0].length) {

return maps[x][y] == 0;

}

return false;

}

public Node findPath(Node startNode, Node endNode) {

openList.add(startNode);

while (openList.size() > 0) {

Node currentNode = findMinFNodeInOpneList();

openList.remove(currentNode);

closeList.add(currentNode);

ArrayList<Node> neighborNodes = findNeighborNodes(currentNode);

for (Node node : neighborNodes) {

if (exists(openList, node)) {

foundPoint(currentNode, node);

} else {

notFoundPoint(currentNode, endNode, node);

}

}

if (find(openList, endNode) != null) {

return find(openList, endNode);

}

}

return find(openList, endNode);

}

private void foundPoint(Node tempStart, Node node) {

int G = calcG(tempStart, node);

if (G < node.G) {

node.parent = tempStart;

node.G = G;

node.calcF();

}

}

private void notFoundPoint(Node tempStart, Node end, Node node) {

node.parent = tempStart;

node.G = calcG(tempStart, node);

node.H = calcH(end, node);

node.calcF();

openList.add(node);

}

private int calcG(Node start, Node node) {

int G = step;

int parentG = node.parent != null ? node.parent.G : 0;

return G + parentG;

}

private int calcH(Node end, Node node) {

int step = Math.abs(node.x - end.x) + Math.abs(node.y - end.y);

return step \* step;

}

public static void main(String[] args) {

Node startNode = new Node(5, 1);

Node endNode = new Node(2, 4);

Node parent = new AStar().findPath(startNode, endNode);

printMap(maps);

ArrayList<Node> arrayList = parent != null ? getPaths(parent) : null;

printPaths(arrayList);

}

private static void printMap(int[][] maps) {

for (int i = 0; i < maps[0].length; i++) {

System.out.print("\t" + i + ",");

}

System.out.print("\n-----------------------------------------\n");

int count = 0;

for (int i = 0; i < maps.length; i++) {

for (int j = 0; j < maps[0].length; j++) {

if (j == 0)

System.out.print(count++ + "|\t");

System.out.print(maps[i][j] + ",\t");

}

System.out.println();

}

System.out.println();

}

private static ArrayList<Node> getPaths(Node endNode) {

ArrayList<Node> arrayList = new ArrayList<Node>();

Node pre = endNode;

while (pre != null) {

arrayList.add(new Node(pre.x, pre.y));

pre = pre.parent;

}

return arrayList;

}

public static void printPaths(ArrayList<Node> arrayList) {

for (int i = 0; i < maps[0].length; i++) {

System.out.print("\t" + i + ",");

}

System.out.print("\n-----------------------------------------\n");

int count = 0;

for (int i = 0; i < maps.length; i++) {

for (int j = 0; j < maps[0].length; j++) {

if (j == 0)

System.out.print(count++ + "|\t");

if (exists(arrayList, i, j)) {

System.out.print("@,\t");

} else {

System.out.print(maps[i][j] + ",\t");

}

}

System.out.println();

}

System.out.println();

for (int i = arrayList.size() - 1; i >= 0; i--) {

if (i == 0)

System.out.print(arrayList.get(i));

else

System.out.print(arrayList.get(i) + "->");

}

System.out.println();

}

public static Node find(List<Node> maps, Node point) {

for (Node n : maps)

if ((n.x == point.x) && (n.y == point.y)) {

return n;

}

return null;

}

public static boolean exists(List<Node> maps, Node node) {

for (Node n : maps) {

if ((n.x == node.x) && (n.y == node.y)) {

return true;

}

}

return false;

}

public static boolean exists(List<Node> maps, int x, int y) {

for (Node n : maps) {

if ((n.x == x) && (n.y == y)) {

return true;

}

}

return false;

}

}

class Node {

public Node(int x, int y) {

this.x = x;

this.y = y;

}

public int x;

public int y;

public int F;

public int G;

public int H;

public void calcF() {

this.F = this.G + this.H;

}

public Node parent;

@Override

public String toString() {

return "(" + x + "," + y + ")";

}

}

