nt[][] 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 },

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

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

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

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

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

};

public class Coord

{

public int x;

public int y;

public Coord(int x, int y)

{

this.x = x;

this.y = y;

}

public boolean equals(Object obj)

{

if (obj == null) return false;

if (obj instanceof Coord)

{

Coord c = (Coord) obj;

return x == c.x && y == c.y;

}

return false;

}

}

public class Node implements Comparable<Node>

{

public Coord coord;

public Node parent;

public int G; // G：

public int H; // H：

public Node(int x, int y)

{

this.coord = new Coord(x, y);

}

public Node(Coord coord, Node parent, int g, int h)

{

this.coord = coord;

this.parent = parent;

G = g;

H = h;

}

public int compareTo(Node o)

{

if (o == null) return -1;

if (G + H > o.G + o.H)

return 1;

else if (G + H < o.G + o.H) return -1;

return 0;

}

}

public class MapInfo

{

public int[][] maps;

public int width;

public int hight;

public Node start;

public Node end;

public MapInfo(int[][] maps, int width, int hight, Node start, Node end)

{

this.maps = maps;

this.width = width;

this.hight = hight;

this.start = start;

this.end = end;

}

}

public final static int BAR = 1;

public final static int PATH = 2;

public final static int DIRECT\_VALUE = 10;

public final static int OBLIQUE\_VALUE = 14;

Queue<Node> openList = new PriorityQueue<Node>();

List<Node> closeList = new ArrayList<Node>();12

private boolean isEndNode(Coord end,Coord coord)

{

return coord != null && end.equals(coord);

}

private boolean canAddNodeToOpen(MapInfo mapInfo,int x, int y)

{

if (x < 0 || x >= mapInfo.width || y < 0 || y >= mapInfo.hight) return false;

if (mapInfo.maps[y][x] == BAR) return false;

if (isCoordInClose(x, y)) return false;

return true;

}

private boolean isCoordInClose(Coord coord)

{

return coord!=null&&isCoordInClose(coord.x, coord.y);

}

private boolean isCoordInClose(int x, int y)

{

if (closeList.isEmpty()) return false;

for (Node node : closeList)

{

if (node.coord.x == x && node.coord.y == y)

{

return true;

}

}

return false;

}

private int calcH(Coord end,Coord coord)

{

return Math.abs(end.x - coord.x) + Math.abs(end.y - coord.y);

}

private Node findNodeInOpen(Coord coord)

{

if (coord == null || openList.isEmpty()) return null;

for (Node node : openList)

{

if (node.coord.equals(coord))

{

return node;

}

}

return null;

}

private void addNeighborNodeInOpen(MapInfo mapInfo,Node current)

{

int x = current.coord.x;

int y = current.coord.y;

addNeighborNodeInOpen(mapInfo,current, x - 1, y, DIRECT\_VALUE);

addNeighborNodeInOpen(mapInfo,current, x, y - 1, DIRECT\_VALUE);

addNeighborNodeInOpen(mapInfo,current, x + 1, y, DIRECT\_VALUE);

addNeighborNodeInOpen(mapInfo,current, x, y + 1, DIRECT\_VALUE);

addNeighborNodeInOpen(mapInfo,current, x - 1, y - 1, OBLIQUE\_VALUE);

addNeighborNodeInOpen(mapInfo,current, x + 1, y - 1, OBLIQUE\_VALUE);

addNeighborNodeInOpen(mapInfo,current, x + 1, y + 1, OBLIQUE\_VALUE);

addNeighborNodeInOpen(mapInfo,current, x - 1, y + 1, OBLIQUE\_VALUE);

}

private void addNeighborNodeInOpen(MapInfo mapInfo,Node current, int x, int y, int value)

{

if (canAddNodeToOpen(mapInfo,x, y))

{

Node end=mapInfo.end;

Coord coord = new Coord(x, y);

int G = current.G + value;

Node child = findNodeInOpen(coord);

if (child == null)

{

int H=calcH(end.coord,coord);

if(isEndNode(end.coord,coord))

{

child=end;

child.parent=current;

child.G=G;

child.H=H;

}

else

{

child = new Node(coord, current, G, H);

}

openList.add(child);

}

else if (child.G > G)

{

child.G = G;

child.parent = current;

openList.add(child);

}

}

}

private void drawPath(int[][] maps, Node end)

{

if(end==null||maps==null) return;

System.out.println("总代价：" + end.G);

while (end != null)

{

Coord c = end.coord;

maps[c.y][c.x] = PATH;

end = end.parent;

}

}

public void start(MapInfo mapInfo)

{

if(mapInfo==null) return;

// clean

openList.clear();

closeList.clear();

openList.add(mapInfo.start);

moveNodes(mapInfo);

}

private void moveNodes(MapInfo mapInfo)

{

while (!openList.isEmpty())

{

if (isCoordInClose(mapInfo.end.coord))

{

drawPath(mapInfo.maps, mapInfo.end);

break;

}

Node current = openList.poll();

closeList.add(current);

addNeighborNodeInOpen(mapInfo,current);

}