链滴

A 星寻路算法代码 --java 版（A* 搜索算法）

作者：sunze19920620

原文链接：https://ld246.com/article/1603932487644

来源网站：链滴

```
import java.util.*;

/**
 * @author sunze
 * @date 2020/10/26 4:55 下午
 */
public class Astar {

    /**
     * 公式： F=G+H
     *
     * 含义：G：起点到当前点的距离 H ： 当前点到终点的距离。
     *
     * F：等于前面两者之和，用于判定下一节点该往哪里走。如下图，此时，我们先不考虑障碍物先。
     */

    /**
     * 初始化地图
     */
    public static final String[][] MAP={
    //            0  1  2  3  4  5  6  7  8
            /*0*/ { "0", "0", "0", "0", "0", "0", "0", "0", "0" },
            /*1*/ { "0", "0", "0", "0", "0", "0", "0", "0", "0" },
            /*2*/ { "0", "0", "0", "0", "0", "0", "0", "0", "0" },
            /*3*/ { "0", "0", "0", "1", "0", "0", "0", "0", "0" },
            /*4*/ { "0", "0", "0", "1", "0", "0", "0", "0", "0" },
            /*5*/ { "0", "0", "0", "1", "0", "0", "0", "0", "0" },
            /*6*/ { "0", "0", "0", "1", "0", "0", "0", "0", "0" },
            /*7*/ { "0", "0", "0", "1", "0", "0", "0", "0", "0" },
            /*8*/ { "0", "0", "0", "1", "0", "0", "0", "0", "0" }
    };

    /**
     * 横着走所需能量
     */
    public static final Integer HORIZONTAL_STEP= 10;
    /**
     * 斜着走所需能量
     */
    public static final Integer OBLIQUE_STEP= 14;

    /**
     * 可使用列表
     */
    public static List<Node> OPEN=new ArrayList<>();
    /**
     * 不可使用列表
     */
    public static List<Node> CLOSE=new ArrayList<>();

    private static final Node END_NODE=new Node(6,7);
    private static final Node START_NODE=new Node(7,1);

//    private static void
```

```
//            0  1  2  3  4  5  6  7  8
    //    /*0*/ { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    //    /*1*/ { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    //    /*2*/ { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    //    /*3*/ { 0, 0, 0, 1, 0, 0, 0, 0, 0 },
    //    /*4*/ { 0, 0, 0, 1, 0, 0, 0, 0, 0 },
    //    /*5*/ { 0, 0, 0, 1, 0, 0, 0, 0, 0 },
    //    /*6*/ { 0, 0, 0, 1, 0, 0, 0, 0, 0 },
    //    /*7*/ { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
    //    /*8*/ { 0, 0, 0, 0, 0, 0, 0, 0, 0 }
    public static void main(String[] args) {
        //将开始节点加入可使用列表
        START_NODE.parent=START_NODE;
        OPEN.add(START_NODE);
        while (CLOSE.indexOf(END_NODE)<0){
            //查找open中的F值最小的节点
            Node minFNodeInOpenList = OPEN.remove(0);
            //将这个节点放到close节点中
            CLOSE.add(0,minFNodeInOpenList);
            //获取这个节点周围的可移动节点并放入到open中
            findNeighborNodes(minFNodeInOpenList);
        }

        //回溯查找最短路径
        Node temp = CLOSE.get(0);
        List<Node> result = new ArrayList<>();
        while (temp.parent!=null&&temp!=temp.parent){
            result.add(0,temp);
            temp=temp.parent;
        }
        result.add(0,START_NODE);
        String[][] resultMap = MAP;
        for (Node node : result) {
            resultMap[node.x][node.y]="$";
        }
        showResult(resultMap);
        System.out.println("--------------------");
        String[][] tempMap = MAP;
        for (Node node : CLOSE) {
            System.out.println(node.x+":"+node.y);
            tempMap[node.x][node.y]="*";
        }
        showResult(tempMap);
    }


    private static void showResult(String[][] arr){
        for (int i = 0, j = 0; i < arr.length;) {
            System.out.print(arr[i][j]+" ");
            j++;
            if (j >= arr[i].length) {
                i++;
                j = 0;
                System.out.print("\n");
```

```
            }
        }
    }

    /**
     * 包括斜着周围一共有8个节点
     * @param currentNode
     */
    public static void findNeighborNodes(Node currentNode) {
        List<Node> temp = new ArrayList<>();
        int x = currentNode.x;
        int y = currentNode.y;
        //
        for (int i=x-1>=0?x-1:0;i<=(x+1<=8?x+1:8);i++){
            for (int j=y-1>=0?y-1:0;j<=(y+1<=8?y+1:8);j++){
                //去除当前节点并且此节点不在close列表中
                if(!(i==x&&j==y)&&CLOSE.indexOf(new Node(i,j))<0&&OPEN.indexOf(new Node(i,j
)<0){
                    //除去障碍物
                    if(MAP[i][j]!="1"){
                        //计算F值
                        Node node = new Node(i, j);
                        node.parent=currentNode;
                        int g = calcG(currentNode, node);
                        int h = calcH(END_NODE, node);
                        node.H=h;
                        node.G=g;
                        node.calcF();
                        temp.add(node);
                    }
                }
            }
        }
        OPEN.addAll(temp);
        Collections.sort(OPEN);
    }

    private static int calcG(Node start, Node node) {
        if(start.x==node.x||start.y==node.y){
            return HORIZONTAL_STEP+node.parent.G;
        }else{
            return OBLIQUE_STEP+node.parent.G;
        }
    }

    /**
     *     这个是最简单粗暴的计算H值得方法，当然还有其它方法，这里先理解AStar的思想先，以后
以自己改进这个计算H值得方法
      */
    private static int calcH(Node end, Node node) {
        int step = Math.abs(node.x - end.x) + Math.abs(node.y - end.y);
        return step * HORIZONTAL_STEP;
    }
```

```
    public static class Node implements Comparable<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 boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            Node node = (Node) o;
            return x == node.x &&
                    y == node.y;
        }

        @Override
        public int hashCode() {
            return Objects.hash(x, y);
        }

        @Override
        public int compareTo(Node o) {
            if(this.F > o.F){
                return 1;
            }
            if(this.F < o.F){
                return -1;
            }
            return 0;
        }
    }

}
```

demo：https://github.com/sunzeJAVA/AStar.git