**A\*算法的实现**

1.A\*算法，是一种在图形平面上，有多个节点的路径，求出最低通过成本的算法。常用于游戏中。通过二维数组构建的一个迷宫，竖线表示墙壁，S为起点，E为终点，“#”代表障碍物，“\*”代表算法计算后的路径。

2.算法的核心公式为：F=G+H

把地图上的节点看成一个网格。G=从起点S,沿着产生的路径，移动到网格上指定节点的移动消耗，在这个例子里，我们令水平或者垂直移动的耗费为10，对角线方向耗费为14。我们取这些值是因为沿对角线的距离是沿水平或垂直移动耗费的的根号2，或者约1.414倍。为了简化，我们用10和14近似。既然我们在计算沿特定路径通往某个方格的G值，求值的方法就是取它父节点的G值，然后依照它相对父节点是对角线方向或者直角方向(非对角线)，分别增加14和10。

H=从当前格移动到终点B的预估移动消耗。使用曼哈顿方法，它计算从当前格到目的格之间水平和垂直的方格的数量总和，忽略对角线方向。然后把结果乘以10。

F的值是G和H的和，这是我们用来判断优先路径的标准，F值最小的格，我们认为是优先的路径节点。

3.

**package** a;

**public** **class** Node {

**private** **int** x; //x坐标

**private** **int** y; //y坐标

**private** String value; //表示节点的值

**private** **double** FValue = 0; //F值

**private** **double** GValue = 0; //G值

**private** **double** HValue = 0; //H值

**private** **boolean** Reachable; //是否可到达（是否为障碍物）

**private** Node PNode; //父节点

**public** Node(**int** x, **int** y, String value, **boolean** reachable) {

**super**();

**this**.x = x;

**this**.y = y;

**this**.value = value;

Reachable = reachable;

}

**public** Node() {

**super**();

}

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

**public** String getValue() {

**return** value;

}

**public** **void** setValue(String value) {

**this**.value = value;

}

**public** **double** getFValue() {

**return** FValue;

}

**public** **void** setFValue(**double** fValue) {

FValue = fValue;

}

**public** **double** getGValue() {

**return** GValue;

}

**public** **void** setGValue(**double** gValue) {

GValue = gValue;

}

**public** **double** getHValue() {

**return** HValue;

}

**public** **void** setHValue(**double** hValue) {

HValue = hValue;

}

**public** **boolean** isReachable() {

**return** Reachable;

}

**public** **void** setReachable(**boolean** reachable) {

Reachable = reachable;

}

**public** Node getPNode() {

**return** PNode;

}

**public** **void** setPNode(Node pNode) {

PNode = pNode;

}

}

**package** a;

**public** **class** Map {

**private** Node[][] map;

//节点数组

**private** Node startNode;

//起点

**private** Node endNode;

//终点

**public** Map() {

map = **new** Node[7][7];

**for** (**int** i = 0;i<7;i++){

**for** (**int** j = 0;j<7;j++){

map[i][j] = **new** Node(i,j," ",**true**);

}

}

**for** (**int** d = 0;d<7;d++){

map[0][d].setValue("\_");

map[0][d].setReachable(**false**);

map[d][0].setValue("|");

map[d][0].setReachable(**false**);

map[6][d].setValue("\_");

map[6][d].setReachable(**false**);

map[d][6].setValue("|");

map[d][6].setReachable(**false**);

}

map[3][1].setValue("S");

startNode = map[3][1];

map[3][5].setValue("E");

endNode = map[3][5];

**for** (**int** k = 1;k<=3;k++){

map[k+1][3].setValue("#");

map[k+1][3].setReachable(**false**);

}

}

**public** **void** ShowMap(){

**for** (**int** i = 0;i<7;i++){

**for** (**int** j = 0;j<7;j++){

System.***out***.print(map[i][j].getValue()+" ");

}

System.***out***.println("");

}

}

**public** Node[][] getMap() {

**return** map;

}

**public** **void** setMap(Node[][] map) {

**this**.map = map;

}

**public** Node getStartNode() {

**return** startNode;

}

**public** **void** setStartNode(Node startNode) {

**this**.startNode = startNode;

}

**public** Node getEndNode() {

**return** endNode;

}

**public** **void** setEndNode(Node endNode) {

**this**.endNode = endNode;

}

}

**package** a;

**import** java.util.ArrayList;

**public** **class** AStar {

/\*\*

\* 使用ArrayList数组作为“开启列表”和“关闭列表”

\*/

ArrayList<Node> open = **new** ArrayList<Node>();

ArrayList<Node> close = **new** ArrayList<Node>();

/\*\*

\* 获取H值

\* **@param** currentNode：当前节点

\* **@param** endNode：终点

\* **@return**

\*/

**public** **double** getHValue(Node currentNode,Node endNode){

**return** (Math.*abs*(currentNode.getX() - endNode.getX()) + Math.*abs*(currentNode.getY() - endNode.getY()))\*10;

}

/\*\*

\* 获取G值

\* **@param** currentNode：当前节点

\* **@return**

\*/

**public** **double** getGValue(Node currentNode){

**if**(currentNode.getPNode()!=**null**){

**if**(currentNode.getX()==currentNode.getPNode().getX()||currentNode.getY()==currentNode.getPNode().getY()){

//判断当前节点与其父节点之间的位置关系（水平？对角线）

**return** currentNode.getGValue()+10;

}

**return** currentNode.getGValue()+14;

}

**return** currentNode.getGValue();

}

/\*\*

\* 获取F值 ： G + H

\* **@param** currentNode

\* **@return**

\*/

**public** **double** getFValue(Node currentNode){

**return** currentNode.getGValue()+currentNode.getHValue();

}

/\*\*

\* 将选中节点周围的节点添加进“开启列表”

\* **@param** node

\* **@param** map

\*/

**public** **void** inOpen(Node node,Map map){

**int** x = node.getX();

**int** y = node.getY();

**for** (**int** i = 0;i<3;i++){

**for** (**int** j = 0;j<3;j++){

//判断条件为：节点为可到达的（即不是障碍物，不在关闭列表中），开启列表中不包含，不是选中节点

**if**(map.getMap()[x-1+i][y-1+j].isReachable()&&!open.contains(map.getMap()[x-1+i][y-1+j])&&!(x==(x-1+i)&&y==(y-1+j))){

map.getMap()[x-1+i][y-1+j].setPNode(map.getMap()[x][y]);

//将选中节点作为父节点

map.getMap()[x-1+i][y-1+j].setGValue(getGValue(map.getMap()[x-1+i][y-1+j]));

map.getMap()[x-1+i][y-1+j].setHValue(getHValue(map.getMap()[x-1+i][y-1+j],map.getEndNode()));

map.getMap()[x-1+i][y-1+j].setFValue(getFValue(map.getMap()[x-1+i][y-1+j]));

open.add(map.getMap()[x-1+i][y-1+j]);

}

}

}

}

/\*\*

\* 使用冒泡排序将开启列表中的节点按F值从小到大排序

\* **@param** arr

\*/

**public** **void** sort(ArrayList<Node> arr){

**for** (**int** i = 0;i<arr.size()-1;i++){

**for** (**int** j = i+1;j<arr.size();j++){

**if**(arr.get(i).getFValue() > arr.get(j).getFValue()){

Node tmp = **new** Node();

tmp = arr.get(i);

arr.set(i, arr.get(j));

arr.set(j, tmp);

}

}

}

}

/\*\*

\* 将节点添加进”关闭列表“

\* **@param** node

\* **@param** open

\*/

**public** **void** inClose(Node node,ArrayList<Node> open){

**if**(open.contains(node)){

node.setReachable(**false**);

//设置为不可达

open.remove(node);

close.add(node);

}

}

**public** **void** search(Map map){

//对起点即起点周围的节点进行操作

inOpen(map.getMap()[map.getStartNode().getX()][map.getStartNode().getY()],map);

close.add(map.getMap()[map.getStartNode().getX()][map.getStartNode().getY()]);

map.getMap()[map.getStartNode().getX()][map.getStartNode().getY()].setReachable(**false**);

map.getMap()[map.getStartNode().getX()][map.getStartNode().getY()].setPNode(map.getMap()[map.getStartNode().getX()][map.getStartNode().getY()]);

sort(open);

//重复步骤

**do**{

inOpen(open.get(0), map);

inClose(open.get(0), open);

sort(open);

}

**while**(!open.contains(map.getMap()[map.getEndNode().getX()][map.getEndNode().getY()]));

//知道开启列表中包含终点时，循环退出

inClose(map.getMap()[map.getEndNode().getX()][map.getEndNode().getY()], open);

showPath(close,map);

}

/\*\*

\* 将路径标记出来

\* **@param** arr

\* **@param** map

\*/

**public** **void** showPath(ArrayList<Node> arr,Map map) {

**if**(arr.size()>0){

Node node = **new** Node();

node = map.getMap()[map.getEndNode().getX()][map.getEndNode().getY()];

**while**(!(node.getX() ==map.getStartNode().getX()&&node.getY() ==map.getStartNode().getY())){

node.getPNode().setValue("\*");

node = node.getPNode();

}

}

}

}

**package** a;

**public** **class** MainTest {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Map map = **new** Map();

AStar aStar = **new** AStar();

map.ShowMap();

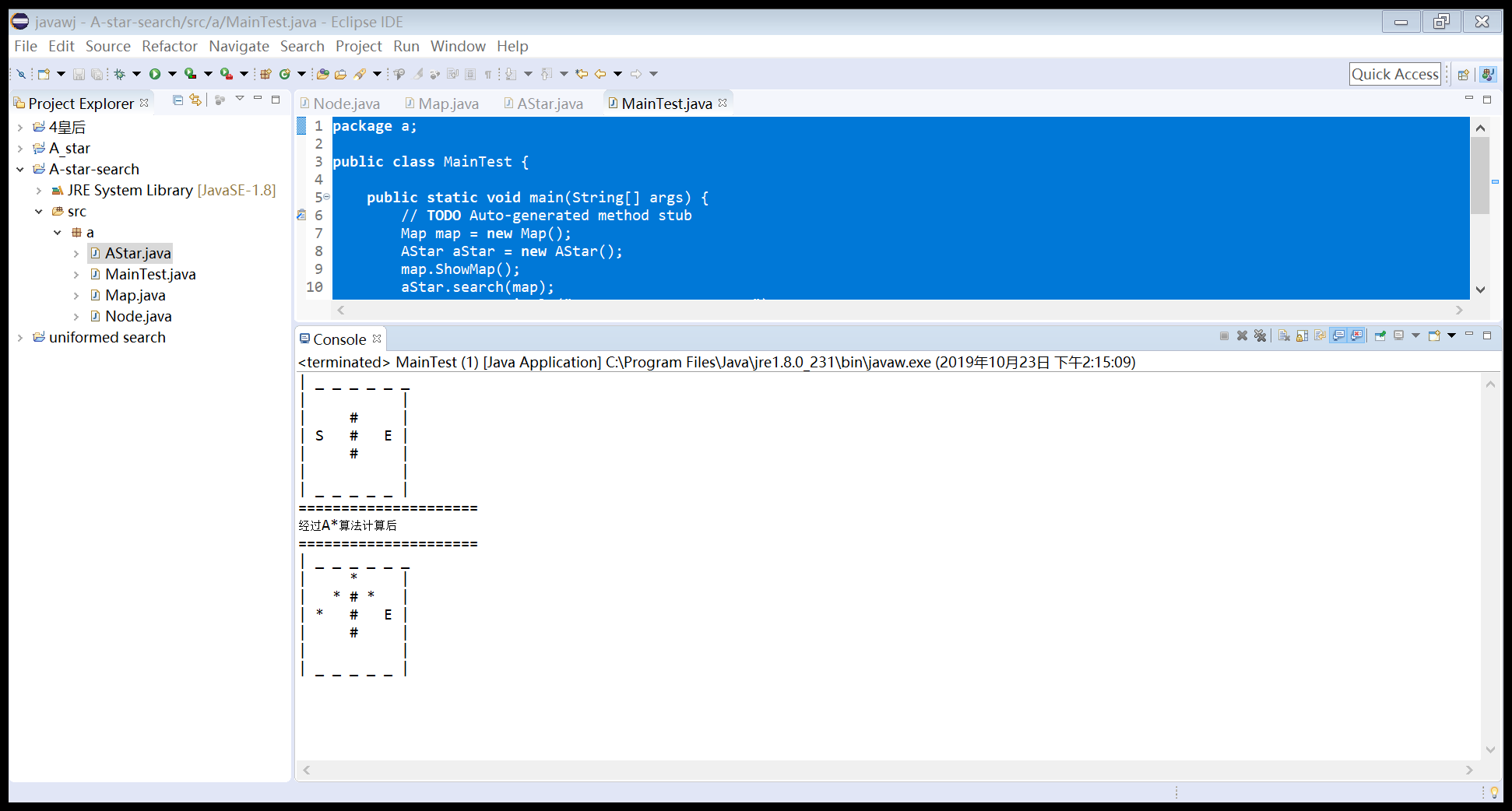
aStar.search(map);

System.***out***.println("=====================");

System.***out***.println("经过A\*算法计算后");

System.***out***.println("=====================");

map.ShowMap();

 }

}

4.